Designing Tools for the Invisible Art of Game Feel

Mads Johansen
IT University of Copenhagen
Denmark

March 2022

A dissertation submitted in compliance with
the requirements for the degree

Doctor of Philosophy
at the Creative AI Lab,
Center for Computer Games Research,
Digital Design Department

Main Supervisor
Assoc. Prof. Martin Pichlmair
Center for Computer Games Research
Digital Design Department
IT University of Copenhagen

Second Supervisor
Prof. Sebastian Risi
Center for Computer Games Research
Digital Design Department
IT University of Copenhagen

page intentionally left blank

Acknowledgements

I would like to thank the people who made this project possible. A Ph.D. is not created in a vacuum; family, friends and colleagues have contributed their time, knowledge and questions to make this project a reality.

First, I would like to thank my main supervisor Martin Pichlmair for his support, guidance and patience with me and my work. I have traveled many winding paths over these past three years, and your guidance, insights and discussions have been invaluable over the years. Without your support, I surely would not have managed to streamline my work to fit in an academic context by tuning my creative output toward concrete tasks and juicing up the papers we published together. I also want to thank my second supervisor Sebastian Risi for his dedication to keeping my research on track with clear goals and academic conciseness. I have enjoyed the benefit of your knowledge about the academic environment in our field and the vast amounts of research carried out within creative computing.

Second, I want to thank Mike Cook for our collaboration and discussions throughout my Ph.D. Although circumstances did not allow me to do a physical stay abroad, I thoroughly enjoyed my virtual stay abroad. I also want to thank you and the research group Knives and Paintbrushes, which continues to provide inspiration and feedback about any and all projects I work on.

I would like to thank my colleagues in the Center for Computer Games Research for interesting discussions about the implications of my work in larger perspectives. I also want to thank both the Creative AI Lab and the Robotics, Evolution and Art Lab for the practice presentations you endured and the project discussions that followed. I would also like to thank the Game AI group at the Queen Mary University of London for the discussions during my virtual visit.

I want to thank the participants in our user studies, without whom we could not have conducted our research. I also want to express my gratitude to the WUDT community for support, feedback, motivation and honest discussions.

Additionally, I would like to thank my family and loved ones who have supported me with work-life balance, amazing questions and the flexibility that allowed me to complete this Ph.D. I would also like to thank Joleen Blom for being a great roommate and colleague, for great discussions about all things game related and for offering great insights into the humanities side of game research. I also want to thank Djordje Grbic and Miguel Gonzalez Duque for our Amager hangouts that were invaluable before and especially during the pandemic. For

troll hunts and shut-up-and-write sessions I would like to thank Mille Edith Nielsen and Nina Stausholm. I would also like to thank Lukasz Paczkowski for game nights and industry insights.

Finally, I would like to thank the people who have shown me kindness and patiently listened to me talking about my work, my motivation and terrible jokes over the years.

Abstract

Over the past decade the related terms and concepts of *game feel*, *juice*, *polish*, *oil* and *feedback* have become more common among game developers. Although books, blog posts, and videos on the subject exist, the terms are still vague, and practitioners do not fully agree on what each concept entails.

This thesis presents a study of game feel design and how design tools can support game developers in designing game feel. The project touches many subject areas and resides in the intersection between game feel design and mixed-initiative tools.

The overarching problem addressed is *how can design tools support game feel design?* This project was motivated by curiosity about game feel, a lack of focus on the topic within game generation research, and missing tools for prototyping game feedback, also known as *juice*. The aim is to provide tools that speed up juice prototyping and help novice game developers understand the process.

This thesis combines practice-based research methods and user testing to explore the topic and solutions to address the above questions. The author draws on practical game design experience and methods and knowledge from research through design, evolutionary algorithms, procedural content generation, and human–computer interaction to explore the subject. Research questions with practical implications are posed, with knowledge acquired over five years as a professional game developer. Experiments and software are designed to evaluate those questions through practice-based design research methods.

This thesis consists of five research publications, bound together by an introduction. The presented research examines game feel from various perspectives, including psychology, human–computer interaction, and game design. It explores game generation, AI evaluation frameworks, and building mixed-initiative tools for game developers. The tools are evaluated through qualitative user testing, focusing on functionality.

The practical result of this Ph.D. is the development and release of several open-source frameworks and tools. The tools were developed and described to conduct experiments, inspire others to explore the topic, and support future research.

The three main practical contributions of this thesis are:

1) The release of the open-source *UnityVGDL*, an adaptation of the GVGAI framework in Unity.

2) A survey and collection of the current state of the art in game feel design, with our attempt at solidifying subjects for individual design tasks, their verbs, and actions.

3) Introducing the unexplored topic of game feel to the mixed-initiative and automated game design research community through several publications, in addition to the release and description of the open-source tool *Squeezer*. *Squeezer* is a mixed-initiative juice prototyping design tool that has been evaluated through two rounds of user testing and has been integrated into an automated game design system.

The projects presented in this thesis seek to inspire game developers and game researchers alike to take advantage of the tools and knowledge in their games and frameworks. *Squeezer* has been open-sourced to allow anyone to incorporate, modify or expand the project. Additionally, full access to the source code allows other developers to analyze and understand how to build new and perhaps better tools for similar purposes.

This thesis sheds light on new areas for mixed-initiative, creative computing, and automated game design research by describing, presenting, and evaluating the tools and the approaches used to create them. The thesis and the related open-source projects are a body of work with the potential to contribute to research, game design practices, and learning environments.

Resumé

I det seneste årti er de relatered udtryk og koncepter *game feel*, *juice*, *polish*, *oil* and *feedback* blevet mere gængse blandt spiludviklere. Selvom der både findes bøger, blogs og videoer om disse emner, er der ikke konsensus om hvad de enkelte udtryk og koncepter dækker over blandt udviklere.

Denne afhandling afdækker en udforskning af game feel og hvordan man kan designe værktøjer der assisterer spiludviklere i processen med at designe game feel. Projektet bevæger sig ind på mange forskellige områder, men befinder sig i overlappet mellem game feel design og mixed-initiative værktøjer.

Det overordnede problem som adresseres er *hvordan kan design værktøjer assistere med game feel design?* Dette projekt er motiveret af nysgerrighed omkring game feel, et manglende fokus på området inden for spil genererings forskning og manglende værktøjer til designing af feedback, også kendt som juice, i prototype-fasen af spiludvikling. Målet med projektet er at tilgængeliggøre værktøjer der kan lette prototype-fasen og hjælpe nye spiludviklere med at forstå processen.

Gennem brug af praktikbaserede design forskningsmetoder opsættes eksperimenter og software løsninger til at evaluerer de ovenstående spørgsmål. Forfatteren udnytter praktisk spiludviklingserfaring samt metoder og viden fra research through design, evolutionary algorithms, procedural content generation og human-computer interaction til at udforske emnet. Med viden opnået gennem fem år som professionel spiludvikler stilles forskningsspørgsmål med praktiske opgaver for øje. Ved at kombinere praktik baseret forskningsmetoder med brugerundersøgelser, udforsker og adressere denne afhandling de ovenstående emner ved at lave og teste forskellige løsninger.

Denne afhandling består af fem forskningspublikationer, bundet sammen af en introduktion. Den præsenterede forskning undersøger game feel fra forskellige perspektiver inklusiv psykologi, human-computer interaction og spildesign. Den udforsker spil generering, evalueringsværktøjer til kunstig intelligens og at bygge mixed-initiative værktøjer til spiludviklere. Værktøjerne er evalueret gennem kvalitative brugerundersøgelser, med et fokus på funktionalitet.

De praktiske resultater af denne PhD er udviklingen og udgivelsen af flere open-source værktøjer. Udviklet og beskrevet for at kunne udfører eksperimenter, inspirere andre til udforskning af emnet og tillade fremtidig forskning baseret på disse projekter.

De tre største forskningsbidrag i denne afhandling er:

1) Udgivelsen af open-source værktøjet *UnityVGDL*, en adaptering af GVGAI frameworket i Unity.

2) En undersøgelse og samling af moderne game feel design, og et forsøg på at sammenkoble emner med individuelle design opgaver, samt deres verber og handlinger.

3) Udgivelsen og beskrivelsen af open-source værktøjet *Squeezer*, et mixed-initiative juice prototype design værktøj. *Squeezer* er blevet evalueret gennem to runder af brugerundersøgelser, integreret i et automatiseret spiludviklingssystem, hvilket har åbnet det hidtil uudforskede game feel emne til forskningsområdet inden for automatiseret spiludvikling.

Projekterne præsenteret i denne afhandling forsøger at inspirere både spiludviklere og spilforskere, til at drage nytte af værktøjer og viden i deres egne spil og forskning. Squeezer er tilgængeligt open-source, for at tillader enhver at integrere, ændre eller udvide projektet. Desuden tillader fuld adgang til kildekoden at andre udviklere kan analysere og forstå hvordan de kan bygge nye og eventuelt bedre værktøjer til lignende formål.

Ved at beskrive, præsentere og evaluere værktøjerne, samt de tilgange brugt til at skabe dem, belyser denne afhandling nye forskningsområder indenfor mixed-initiative, creative computing og automatiseret spiludvikling. Den resulterende afhandling og de relaterede open-source projekter udgøre derfor et værk med et potentiale til at bidrage ikke bare til forskning, men også til spiludvikling og spilundervisning.

Contents

Chapter 1

Introduction

Since the earliest graphical computer interfaces, computer-aided design (CAD) [207] has been a stable research field within human–computer interaction (HCI) research. Sutherland's early CAD system Sketchpad [207] was a system that leveraged computers to aid designers in ways that answered some "What if...?" questions from pen and paper design. Questions like "What if, instead of having to erase a line and redraw it, I could just move the endpoint?" and the logical follow-up questions, "What if, when we move that point, any other connected points also followed it?" or "What if instead of carefully drawing two lines exactly perpendicular, we could constrain them to be exactly perpendicular always?" Sutherland launched a completely new way of using computers with the Sketchpad system, showing people the potential of a collaboration between humans and computers and pioneering early HCI research.

Focusing on past achievements like Sutherland's Sketchpad and logic-based programming languages such as Prolog[1], Bret Victor imagines an alternative "future of programming" [228]. Victor questions how software design is conducted today, reminding us that interacting with computers was a very different task in the not-so-distant past. It involved directly manipulating data, specifying constraints and goals, and letting the computer figure out how to do it. Victor's research and demonstrations focus on dynamic alternatives to data visualization [226, 225] and using the computer as a new visual art medium [227]. The ideas presented have been around for a very long time. However, Victor proposes bringing them back and changing modern interaction schemes to a more designer-friendly variety based on direct manipulation and instant previews. Designing software programs, for instance, is a largely asynchronous task alternating between "blindly" manipulating symbols and evaluating the resulting program. Some languages or interactive development environments (IDE) provide live "previews" of the graphical user interface (GUI), but very few provide the option for editing the application while it is running. Although some IDEs provide intelligent programming suggestions or smart visual programming representations, the task is still inherently disconnected from the result. Victor argues that designing with computers does not necessarily mean providing the computer with the exact instructions for doing the task. Instead, a user should be able to directly visualize data, design programs, or build games without worrying about how the computer does it. Instead of sole

[1] https://en.wikipedia.org/wiki/Prolog

responsibility resting on the user, the system carries the responsibility of accomplishing the result the user specifies.

In mixed-initiative research, which this thesis is concerned with, splits the responsibility or "initiative" in this manner. Mixed-initiative tools assist their users by taking responsibility for specific sub-tasks, providing suggestions, or taking turns alongside the user when manipulating an artifact. Like much modern software, mixed-initiative tools can be viewed as CAD systems. However, by sharing responsibilities in the creative process, they bring back that "magic" feeling computer-aided design had in the early days. Many mixed-initiative tools employ procedural content generation (PCG) systems to provide their input. PCG systems have existed since the early days of computing and have been a staple in game development for almost as long. PCG, as the name implies, generates content based on procedures. These procedures and any data they might use act as constraints and parameters to produce the generated output. In a sense, the PCG system acts as a CAD tool generating a range of artifacts based on constraints the designer adds, removes, or adjusts. The cycle of generation and adjustment continues until the generator provides the desired output. Where most CAD tools rely on the users for the creative part of the process, and PCG systems rely on the computer for the creative part of the process, mixed-initiative systems allow both the computer and users to assist in parts of the creative process. The tool Danesh [45] attaches to and adjusts the parameters of a PCG system to achieve certain meta goals, moving the design exploration responsibility away from the user, essentially building a mixed-initiative interface for PCG systems. On the game-design-practitioner side, there are mixed-initiative tools like SFXR [154] and direct manipulation tools like Doodle Studio 95! [161] and Flickgame [121]. Mixed-initiative research has resulted in tools that generate artifacts ranging from textures, 3D-models, and levels, to sounds, music and even game rules. Examples of such systems like Tanagra, Sentient Sketchbook and Maestro Genesis [193, 128, 86] have been designed for various purposes related to game development. Some systems even generate or assist in generating entire games, such as Game-o-Matic, Interactive Game-Design Assistant and ANGELINA [221, 139, 39, 40].

Most of these tools are concerned with generating specific artifacts common to games. However, there is more to games than their assets; there is an affective side to games that is often neglected or overlooked by "non-practitioners." Designing the affective moment-to-moment interactions in a game is called game feel design. Game feel design is often overlooked because it is almost invisible to the player when it works as intended. Good game feel design essentially makes every interaction feel intuitive, natural, or gritty so that the player forgets to notice the details and instead becomes immersed in the experience. Designing game feel takes many iterations and experience with coalescing all the details that make the experience coherent. Game feel design is a detail-oriented endeavor, one aspect of which is called "juicing". In the talk entitled "Loving Your Player With Juicy Feedback," Hunicke [88] describes it: *juiciness can be applied to abstract forms and elements and it is a way of embodying arbitrarily defined objects and giving them some aliveness, some qua, some thing, some tenderness.* As the title of the talk also suggests, the art of game feel is pouring love into minute details of a game to

ensure it feels just right. Juice is there to give objects and the world life, becoming invisible to and accepted by the player as the reality of the game. Another seemingly invisible aspect of game feel is designing support for the player through streamlining it. *The goal of streamlining is to make rough edges of the game disappear to provide a smooth player experience. Most of the time, the player does not want to realize how much the game is supporting them. 'If you do this right, then the player won't suspect a thing," says Pulver [160]. Disc Room's designer Nijman explains that their use of Coyote Time "has a bunch of good side effects that make it seem like the game knows your intentions" [231].* As we described in [155], solving problems so the player can remain blissfully unaware by interpreting their intentions and helping them along the way is the reason why excellent game feel design remains almost invisible.

This Ph.D. project has been a journey to discover how to design tools for game feel design. Tools that support the game feel design process while developing game prototypes.

The research methodology of this project is based on prior experience as a game developer and akin to practice-based design research [223, 163]. *Software* is the explored "material" much like in software studies [67]. The process is rooted in experimentation, formulating questions, and exploring them through an iterative development process. Primarily focused on improving the learning curve for novice game designers and tools that allow experienced users faster iteration and prototyping in real working environments (e.g., off-the-shelf game engines and fantasy consoles). Exploring game feel through software development using real-world practices and academic knowledge. The resulting tools and prototypes act as both showcase and learning examples. The practical usage of the tools has provided feedback for the tool design process throughout the project.

In practice-led design research, the research's frame of a project is called the "program". In this case, the program is defined by the main research question: *How can design tools support game feel design?* This places the research in the overlap between game feel design and mixed-initiative tools. The main research question is divided into more precise questions for which "experiments" are designed. Experiments in this project consist of an iterative development process followed by a test phase. From each experiment, two layers of results are derived. First, the resulting software is described and released, allowing others to reuse the approach or the resulting software. Second, the test results, indicating how well the software performed its task and whether the approach is feasible for further research. The experiments and their results then yield new research questions to be investigated through new experiments. Lastly, the dissertation appears as a collection of experiments and their results, how they fit into the overarching program and how the Ph.D. project addressed the main research question.

The contributions of this dissertation and an outline of how each chapter addresses the research questions (see Section 1.1) can be found in Section 1.4.

1.1 Research Questions

The main research question of this dissertation can be summarized as:

*How can **design tools** support **game feel design**?*

This question is derived from the larger question, "How can the computer assist in the game design process?" A question others have wrestled with through the exploration of how to make tools that support, suggest, and take part in various aspects of the game design process [194, 192, 126]. These questions are themselves part of the even larger quest in computer-aided design research asking "How can the computer assist in the design process?" [131]

Others have also explored moving past the idea of the computer assisting humans directly in the process, instead focusing on "How can the computer design certain aspects of games?" [203, 111] or "How can the computer create games without human assistance?" [41, 79]. The research revolving around questions like these often leads to new techniques and analytical challenges in game design. In this dissertation, exploring procedural content generation and automated game design is part of uncovering how tools can support the process of game feel design.

The main research question only addresses game feel design instead of all (game) design processes. Besides limiting the scope to game feel design, the focus is on *supporting* the game feel design process through tools. This subtle shift in focus moves the goal from the computer assisting in the game design process to providing essential support in the design process with the computer.

However, the question remains too big to be answered in full within the limits of a single Ph.D. project.

To understand how design tools can support the game feel design process and begin answering the main question. The main question needed to be divided into a set of narrower questions that could be tackled in this dissertation:

- RQ1 *"What is game feel design?"*
- RQ2 *"How can game feel design support get implemented in existing workflows?"*
- RQ3 *"What is the impact of design tools on the game feel design process?"*
- RQ4 *"How does the addition of game feel design tools alter the qualities of games?"*

This dissertation investigates the process of game feel design and how adding tools to support it changes both the process and the resulting games. What game feel design is *(RQ1)* is explored through a survey [155]. Reconciling more than a decade's worth of game design research, industry terms, and practices, with research from several other fields like psychology and human-computer interaction stretching back much further in time. The remaining questions are tackled through the creation and adaptation of software prototypes. Squeezer is the central piece, a design tool that allows designers to explore game feel design. Through several user tests, Squeezer serves to answer both what the impact of design tools is on the game feel design process *(RQ3)* and how they alter the qualities of games built with them *(RQ4)*. While the implications of adding design tools to the design process are at the core of what the impact of design tools is *(RQ3)*, several approaches to providing support in the design process are explored through different interfaces as well as through generative and algorithmic means. The full set of tool design explorations in various prototypes throughout this project serve as examples of potential answers to how game feel design support can get implemented in existing workflows *(RQ2)*. This project focuses on providing support during design exploration related to game feel design. Though design explorations can be approached from various angles, it happens most commonly as part of the prototyping phase of game development. Both common and generative approaches to game prototyping are explored in the context of potential implementations of game feel design support in existing workflows *(RQ2)*.

While it is hard to claim that the questions above have been answered fully, this dissertation forms a foundation for further exploration of game feel within the research areas of game design, procedural content generation, mixed-initiative design tools, computational creativity, and computational intelligence. Highlighting both solutions and challenges related to making design tools that support the game feel design process.

1.2 List of Papers

The papers are available in Chapter 9 Publications.

Paper 1
M. Johansen, M. Pichlmair and S. Risi, "Video game description language environment for Unity machine learning agents" in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8,
DOI: https://doi.org/10.1109/CIG.2019.8848072

Paper 2
M. Johansen, M. Pichlmair, and S Risi, "Squeezer - A tool for designing juicy effects" in *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play* (*CHI PLAY '20*). Association for Computing Machinery, New York, NY, USA, 2020, 282–286.
DOI: https://doi.org/10.1145/3383668.3419862

Paper 3
M. Johansen, M. Pichlmair, and S. Risi, "Squeezer - A mixed-initiative tool for designing juice effects" in *The 16th International Conference on the Foundations of Digital Games (FDG) 2021* (FDG'21), August 3–6, 2021, Montreal, QC, Canada. ACM, New York, NY, USA, 11 pages.
DOI: https://doi.org/10.1145/3472538.3472575

Paper 4
M. Pichlmair and M. Johansen, "Designing game feel. A survey." in *IEEE Transactions on Games*, 2021,
DOI: https://doi.org/10.1109/TG.2021.3072241

Paper 5
M. Johansen and M. Cook. 2021, "Challenges in generating juice effects for automatically designed games" in *The 17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-21)*, 2021.
Best Student Paper Nominee at AIIDE 2021

1.3 List of Open-Source Projects

In addition to this dissertation and the included papers, the source code for several projects and experiments is freely available online. Releasing frameworks and projects has been a big part of this Ph.D. project. They are providing researchers and practitioners interested in continuing, extending, or learning from the source code with the option to do so. For instance, *UnityVGDL* provides a starting point for anyone wishing to work with VGDL in Unity. While VGDL has been explored in many different projects and papers [175, 176, 152, 150, 151, 113], the topic of game feel design [155] is relatively new. The availability of a project like *Squeezer* allows researchers to continue work on exploring interface design, algorithmic assistance, and new generative approaches to game feel design tools. *Squeezer* is primarily concerned with event signification through juice effect generation and design. Making *Squeezer* freely available to practitioners for prototyping juice effects increases the chances that it will be used, adapted, and improved upon by others.

Additionally, *Squeezer* is an excellent way for novice designers to learn event signification through the playful exploration of juice effects. Lastly, two experiments with particle effect generation and a lightweight version of Squeezer for the fantasy console Pico-8[2] are also included here. The two experiments represent future research opportunities and attempt to generalize ideas included in Squeezer to platforms beyond Unity.

Project Status and links below:

- UnityVGDL
  - Available at: https://github.com/pyjamads/UnityVGDL
  - Run VGDL games in Unity
  - Machine learning support using ML-agents [101, 100]
- Squeezer
  - Available at: https://github.com/pyjamads/Squeezer
  - Effect sequence designer and generator for prototyping juice effects in Unity.
  - Proof-of-concept interactive evolution mode
  - Novelty search for interactive evolution using tree-edit distance (experimental)
  - API for effect generation and design (used for automated game design system [93])
  - ***Squeezer*** *passed Artifact Evaluation at AIIDE 2021 in connection with [93]*
- Fantasy Console Experiments (Pico-8)
  - Squeezer for Pico-8, available at: https://pyjamads.itch.io/squeezer-pico8
  - Procedural Particle Puffs, available at: https://pyjamads.itch.io/pcg-puffs-pico8

[2] Available at https://www.lexaloffle.com/pico-8.php

1.4 Contributions & Outline

The following outline is based on a collection of results, contributions, and partial answers to the research questions posed in Section 1.1. The curious reader should be warned that this section contains spoilers and provides an overview of the rest of the dissertation.

Chapter 2 presents related work and background knowledge, including various technologies, theories, and practices at work in this project. The chapter is neither part of the contributions or answers, but it provides a necessary foundation for understanding the rest of the dissertation.

Chapter 3 explores state of the art within game feel design, addressing the first research question: *"What is game feel design?" (RQ1)*

The results and contributions in the chapter are the identification and classification of a long list of design elements commonly associated with designing game feel. Additionally, game feel design is subdivided into three domains, each with a respective goal and a described polishing task. Polishing in this circumstance refers to adjusting the game to exactly match the experience intended by the designer. The three design domains related to the feeling of moment-to-moment interaction are *Physicality*, *Amplification* and *Support*. Each domain contains a polishing task, namely **tuning**, **juicing** and **streamlining**. The feeling of moment-to-moment interactions can be adjusted by tuning the physicality, juicing the amplification, and streamlining the support. Each domain and design element is described and situated within game design and connected to related research fields. The survey [155] gathers resources from practitioners and academia in an area that has seen little research in the decade since the original release of Swink's book on Game Feel [214].

Chapter 4 looks at the different game development workflows touched upon during this Ph.D. and describes possible ways to implement game feel support in them. These descriptions partially answer the second research question: *"How can game feel design support get implemented in existing workflows?" (RQ2)*

The chapter explores the implementation of game feel design support in three existing workflows. First, a section related to game generation using description languages and in particular the video game description language (VGDL) [125, 175]. Second, a section describing how a generalizable game development pattern lead to the implementation of Squeezer [95, 94]. Third, a section describing potential approaches to, and the process of, integrating Squeezer into an automated game design (AGD) system [93].

For game description languages, three potential solutions for implementing game feel support are identified and described. The particular game description language explored is VGDL [175, 125, 152, 113, 150, 151]. The first solution is language modifications based on unpublished experiments with VGDL descriptions during the development of UnityVGDL [96], exploring potential ways of building game feel support into the framework. The solution requires adding

a library of juice effects (amplification), more tuneable player classes (physicality), and making collision handling more adjustable (support). The second solution is to expand descriptions to include options that allow a third-party library to handle the execution in conjunction with the game execution. The final proposed solution is to generate entire game prototypes from the VGDL descriptions in a game engine like Unity. This approach allows a game designer to quickly explore rule and level changes while maintaining the ability to improve the aesthetics and feel of the game in an external engine. This approach of testing the initial design and generating game prototypes using a description language has even been used in the development of commercial games[3].

Most importantly, this section paved the way for generalizing the approach used in VGDL to designing juice effects. In VGDL, reacting to objects' interactions (read: collisions) is used for executing "effects". VGDL "effects" are the game mechanics and rules that interactions trigger. As juice effects are concerned with the amplification of game events and game events primarily happen during interactions between objects (or as a result of player input). Additionally, the language modification experiments (using a nested tree structure) showed a powerful concept for sequence design, later used as the basis for visual sequence design in Squeezer.

The contributions concerning Squeezer in Chapter 4, is the description of the implementation of an interactive sequence design system, methods for executing said sequences, and a method for algorithmic generation of effect sequences. Additionally, the chapter describes methods for shortening the iteration loops, allowing designers to explore the design space more effectively. Short iteration loops and real-time previews are an attempt to "stop drawing dead fish" [227] during juice design. However, the immediate contribution is making Squeezer available open-source as a reference for other researchers and developers.

Lastly, the chapter shows how the approaches to integrating game feel design into an automated game design (AGD) system are similar to those available for other game generation systems (e.g., VGDL)—specifically providing the implementation details for integrating Squeezer into an AGD system. Additional contributions include identifying and describing challenges related to adding game feel design into AGD systems and outlining fascinating problems that require future research and computational creativity solutions.

Chapter 5 explores how using *Squeezer* impacts the game feel design process in order to answer the third research question: *"What is the impact of design tools on the game feel design process?" (RQ3)*

The chapter begins by outlining current approaches to game feel design and describing the design philosophy behind Squeezer. The outlined approaches are compared to the results from user studies performed in the two Squeezer papers [95, 94]. At the same time, Squeezer is evaluated against the design philosophy and how well it allows for playful exploration and design of juice effects.

[3]Presented by Alan Hazelden at Game Developers Conference 2016 in the Tech Toolbox panel. https://www.gdcvault.com/play/1023377/Tech

The chapter's contributions are primarily based on user studies, indicating that Squeezer's visual sequence design represents an excellent approach to designing juicy effects. Squeezer is a tool the participants claim to be interested in using during prototyping and game jams. The user studies indicate that designers with less experience designing juice engage with interactive evolution longer. The experiment is based on a simple implementation of interactive evolution and might indicate that interactive evolution needs to provide more variation and diverse examples to keep experienced designers interested. Additionally, the chapter describes the potential benefits of sequence generation as an evaluation first approach to juice design. In this approach, the designer generates an effect sequence and evaluates how well it fits the interaction. The approach contrasts with the typical approach where the designer has to envision the juice effect and then create it using an asynchronous process, usually through code or visual scripting. The chapter argues that the evaluation first approach side-steps the issue of the designer staring at a "blank page" when they lack inspiration.

Chapter 6 explores the impact of adding support for game feel design into the game design process in order to explore the fourth research question: *"How does the addition of game feel design tools alter the qualities of games?" (RQ4)*

The section begins by reiterating the purpose of game feel design, and especially the purpose of juice. Then followed by a reflection on how the test participants would use Squeezer and their focus and interest while working with the tool. Then it is wrapped up by a deeper analysis of the qualitative results from adding Squeezer to an AGD system and allowing users to play different variations of those games with different effects [93]. The results appear in line with those found by Kao [105] in a similar study. They are indicating that adding juice effects to games (even when those effects are randomly generated) will at the very least not hurt the perception of the game, as long as they are not too extreme. This study exposes the potential advantages of adding juice effects or game feel design support to AGD systems and other game description and generation frameworks such as VGDL. In the study, the generated effect sequences were tailored towards the general event type but lacked alignment with the interaction they were meant to amplify. The alignment of the effect sequences with the interaction and the overall aesthetics of the game is also one of the core challenges identified in Chapter 4 for adding game feel support to AGD systems.

Chapter 7 reviews the contributions extensively with respect to the research questions. Lastly, the chapter ends with a discussion of potential future research on topics related to this dissertation.

I seldom end up where I wanted to go, but almost always end up where I need to be.[4]

[4]From "The Long Dark Tea-Time of the Soul" by Douglas Adams, 1988

Chapter 2

Background

This chapter outlines relevant work and background knowledge for the research conducted in this project. The first section presents practice-based design research, a research methodology that inspired how this project was conducted. Especially evident in Chapter 4 that is concerned with the practical implementation experiments that have led to most of the publications created for this dissertation. The second section presents the subject of game development, which is the primary focus of this dissertation. Chapter 3 acts as an extension of the game development background outlining the primary focus of this dissertation. The sections following game development are procedural content generation and evolutionary algorithms. These sections present generative approaches that have been applied and explored through user testing (see Chapter 5). A section on computational creativity follows these two sections, as it has been influential both to this project and to the topic of automated game design (explored throughout the dissertation, but more prominent in Chapter 6). Lastly, related work on mixed-initiative tools is outlined, in general, and in relation to game development.

2.1 Practiced-Based Design Research

Practice-based design research [223] aims to study materials or practices through designing artifacts, methods, frameworks, and tools. Practice-led research is shaped by the design practice which informs it. Design theory is inherently temporal, incremental, and derived from ever-changing examples and practices. Redström [163] describes how to derive design theory from practices and how design theory of this kind is temporal. Two main types of design theory arise from practice-led design research, methods, and practical evaluations. The methods are often accompanied by reflection on the practice and the results, leading to practice-oriented insights valuable for "non-practitioners". One such example is the design of a chair in Redström's book, which in practitioner reflections becomes designing the act of sitting or designing an object intended for the act of sitting. This reflection allows the practitioner to play around with the idea of how the act of sitting can be done differently, resulting in chair designs that were radically different from the norm (seen in Figure 2.1). By describing the process, intent, and creation of the chair, the design practitioner contributes a new method for designing chairs.

Additionally, performing evaluations of the chair might reveal certain benefits that increase the value of the design and might incentivize others to follow a similar design path.

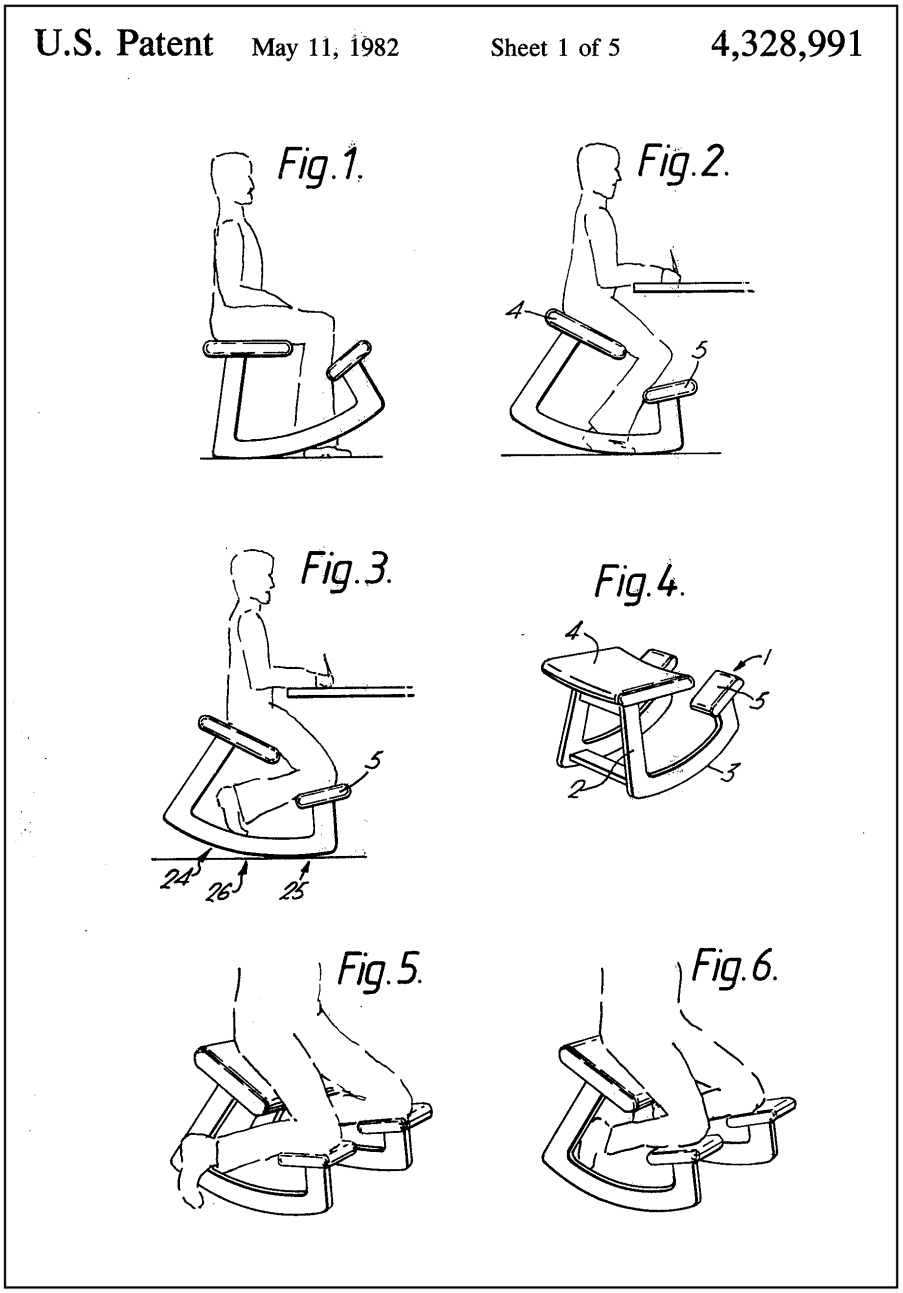


Figure 2.1: Illustration from patent no. US 4328991 A for a "sitting device" by Hans C. Mengshoel and Peter Opsvik, 1982.

Both Redström, Vaughan, and Brandt et al. [17] describe the notion of a "Program" within which design research is conducted. This program is based on a central research question that makes up the frame of the project. Research projects explore the program through experiments based on sub-question. The descriptions of experiment, method, and results become the knowledge produced by and pushing the boundaries of the program. Design research is temporal because the program boundaries are moving both in terms of contemporary methods and materials and because other designers push the boundaries of what is possible and accepted.

2.2 Game Development

Videogames have existed for more than six decades. They began as simple vector drawings on an oscilloscope, in Tennis for Two[1]. They have since been developed for almost every conceivable and sometimes inconceivable device on the planet. Today game development is a multifaceted endeavor involving many different and sometimes opposing disciplines to create a wide variety of interactive experiences. Disciplines spanning various art forms, different types of engineering, several facets of design in addition to business and project management. Whether your approach is analytical [177] or playful [71], or you just believe play matters [183] and people have a need to play, games begin with ideas. Testing is a big requirement for success from the first ideas until release and beyond. Testing takes many forms in game development, but the primary way is making prototypes. As Fullerton [71] puts it *prototyping lies at the heart of good game design. Prototyping is the creation of a working model of your idea that allows you to test its feasibility and make improvements to it. Game prototypes while playable, usually include only a rough approximation of the artwork, sound and features. They are much like sketches whose purpose is to allow you to focus on a small set of the game's mechanics or features and see how they function.*

Prototypes allow designers to expand the scope of their idea to figure out the best solutions throughout development. Game development often goes through several phases of expanding ideas and narrowing down through testing (An example of this flow can be seen in Figure 2.2). Prototypes let designers explore possible options, but they need to be cheap and quick to make, wasting as little time as possible building and testing them because most will be discarded. Sketching out prototypes allows designers to *think more clearly and openly about their ideas* and *create abundant ideas without worrying about their quality* [76]. They are meant to verify idea feasibility or show the potential of an idea, but prototypes can also be used to document the development process.

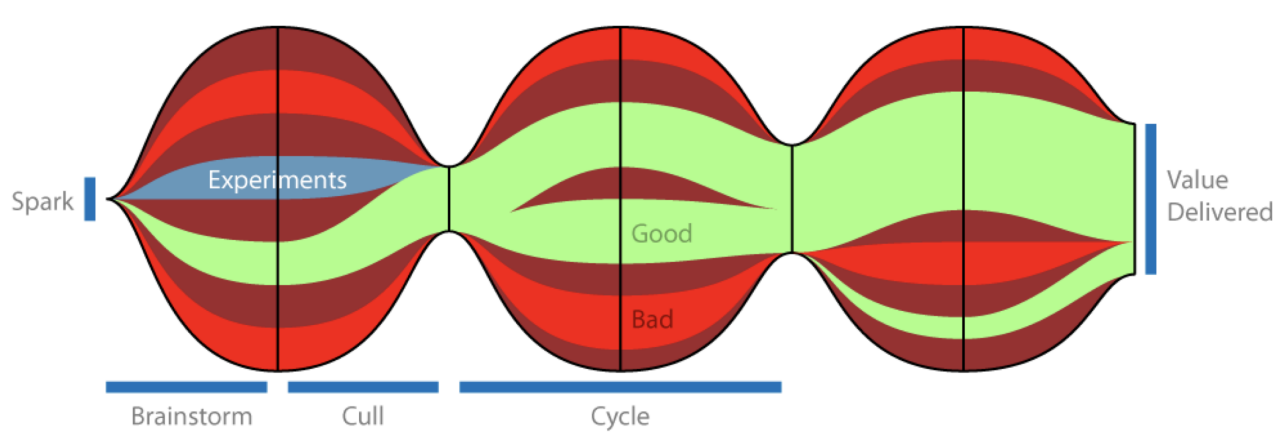


Figure 2.2: A common flow of a game development process, with alternating phases of design exploration and culling based on testing.

[1] https://en.wikipedia.org/wiki/Tennis_for_Two

Prototypes can be created in many different ways, depending on the aspect they are meant to test. Early on, developers may use pen and paper to test game mechanics, but as the game idea grows more complex and gets further in development, often new ideas need to be tested in the context of the game. This is where prototyping tools come into play. Prototyping tools ideally allow many ideas to be created and tested in the game context. Prototyping tools support designers in exploring the design space [129], expanding possibilities before evaluation narrows it down again. Examples include generating sound samples with SFXR [154], designing shaders through Shader Forge [2] or drawing quick sprites with animations in Doodle Studio 95! [3] [161]. Often tools are selected, and workflows are built around them to maintain fast iteration times and ease the development process.

2.3 Procedural Content Generation

Procedural content generation (PCG) is the generation of a variety of different artifacts by applying a specified procedure. The generated artifacts can serve as "content" in various contexts, ranging from rug weaving patterns to the entire written lineage of a single dwarf in *Dwarf Fortress*[4]. PCG has been used in games for a long time, and the algorithms have been used to generate content for many different purposes [181, 182]. In games, PCG systems have often afforded the replay-ability and surprising elements that keep players engaged and coming back again and again. PCG systems replace a content design task with a system design and an evaluation task. As Emily Short[5] says PCG *isn't a substitute for designing content. It's a way of designing content — one that is often at least as labor-intensive as other ways, and that also demands a strong capacity for abstraction and the ability to characterize one's aesthetic goals.*

PCG systems are sometimes viewed as a magic bullet. However, they just transform a design task into a different kind that might be easier to understand and create for some people. Generating assets using a PCG system can also make a game world seem more natural, as people are used to minor variations in almost all materials they experience. People experience slight imperfections in almost anything they interact with due to wear and tear, production defects, or natural variations. Elements like rocks, trees, or clouds all exhibit these natural variations. The human brain is very good at recognizing patterns, and if a game repeats a texture or element too many times, most people would notice it and find it uncanny[6]. The idea behind using PCG is often to replace manual labor in creating large enough amounts of assets to avoid the issue of repeating patterns.

PCG systems are often implemented either dynamically as part of the shader code[7] or to

[2] https://acegikmo.com/shaderforge/

[3] https://fernandoramallo.itch.io/doodle-studio-95

[4] A notoriously deep simulation game, with a rich history for a generated "living" world, https://www.bay12games.com/dwarves/ see also https://www.moma.org/collection/works/164920

[5] https://emshort.blog/2016/09/21/bowls-of-oatmeal-and-text-generation/

[6] Similar to the Uncanny Valley https://en.wikipedia.org/wiki/Uncanny_valley

[7] A shader is the piece of software the graphics processing unit performs to determine the color of pixels

generate a variety of assets. However, as stated earlier, assets or artifacts generated by a PCG system can be many different things. Similar to issues with textures repeating patterns, patterns of textual output have to be obfuscated enough that the player cannot discern which parts are generated and which are authored by designers. When using PCG systems to generate content, the amount of content is proportional to the design space and the granularity of the parameters used in the generation. However, while using floating-point numbers versus integers to define, for instance, a color value would allow the PCG system to generate almost infinitely more colors, the actual generated artifacts might still look incredibly similar[8]. With that in mind, creating a PCG system means defining an obfuscated design task with parameters that can create meaningful variations. One strength of PCG systems is that if used in a way where the asset is only encountered once, when one particular asset is bad or surprising, with enough content, it does not take away from the whole experience.

With the explosion of machine learning (ML) research in the past decade, it is no surprise that PCG and ML have been explored and coined as PCGML [206]. An area that has also been explored as part of Khalifa's dissertation on general procedural level generation [111]. Similarly, generative adversarial networks (GANs) [49] have become synonymous with machine learning-based PCG. One example uses GANs for PCG in puzzle games [83] adjusting the generation through parameterized condition vectors.

One way to get around the issue of less optimal content is to search the design space using search-based PCG algorithms [219, 218]. Instead of the PCG system generating content directly, search-based approaches (such as evolutionary algorithms) search the content space using evaluation functions to guide the search toward good solutions effectively.

2.4 Evolutionary Algorithms

Evolutionary algorithms (EA) [59] are algorithms that search a solution space using an approach derived from biological natural selection. EA optimizes a population of individuals by allowing those who score highest in a fitness evaluation to repopulate the next generation and remove the rest. Each individual contains genetic material known as the *genotype*[9] which can be used to create the *phenotype*[10] that is evaluated with a fitness function. The individuals with the highest fitness scores are then selected, and their genotype is either copied, mutated, or crossed with other individuals in the population to create new individuals. Mutation and crossover create new individuals that retain elements of the previous best individuals. Sometimes that leads to individuals who perform poorly and sometimes individuals that perform better than previous generations. By repeating the process of removing individuals who perform poorly and generating offspring based on the best performing individuals, EA is incredibly effective at

displayed on a screen https://en.wikipedia.org/wiki/Shader

[8]This leads to the "10000 bowls of oatmeal" problem https://galaxykate0.tumblr.com/post/139774965871/so-you-want-to-build-a-generator

[9]The genetic material of the individuals which will be used to create offspring.

[10]The instructions, agent behavior or artifact that will be evaluated.

searching a space of possibilities. With each generation of the population, the average fitness improves, and for tasks that can be solved, the evolution usually stops when a "good enough" solution is found. One pitfall in task evaluation is that some tasks may have local optima in their solution space, meaning sub-optimal solutions exist (such as taking the right exit in a roundabout but going around it multiple times). To avoid this and find the global solution optima, the mutations have to be large enough that a single individual can escape the local optimum and find a better search space outside the local optimum.

2.4.1 Interactive Evolution

Interactive evolution [185] covers implementations of evolutionary algorithms with the fitness function fully or partially replaced by an outside agent (usually a human being), selecting the individuals that are used for populating the next generation. The most common way implementation can be seen in systems like Picbreeder[11] [179] or Endless Forms[12] [32], where a user is presented with a side-by-side list of artifacts to choose from. The user then selects one or more artifacts for the algorithm to seed the repopulating of the next generation. Interactive evolution implementations sometimes run a few generations internally or apply quality diversity algorithms [159] in order to get appropriate amounts of change between displayed generations. Getting appropriate amounts of change is done to avoid user fatigue[13], resulting in users abandoning the process too quickly. In systems like Picbreeder and Endless Forms, the initial populations displayed will feature artifacts selected and created by other users. Preselection continues the overarching design exploration of the search space across multiple user sessions. Similar to how machine learning has entered the procedural content generation space with PCGML [206] and GANs [83] interactive evolution is being combined with machine learning. One example is combining interactive evolution with latent space exploration of generative adversarial networks (GANs) [178].

2.5 Computational Creativity

In 2012 Colton and Wiggins [36] introduced their definition of the field of computational creativity (CC). Stating that it is *the philosophy, science, and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviors that unbiased observers would deem to be creative.* The thing that defines the field is its interest in the processes involved in making systems appear creative, not just the output of the systems and how creative they might be. One example of this focus can be found in the earlier computational creativity theory (CCT) by Colton et al. [35]. CCT was an attempt to describe the assumptions and theories of software development on how creative acts can be implemented.

[11] See https://picbreeder.org/

[12] See http://endlessforms.com/

[13] User fatigue is a common issue in interactive evolution, where users become bored or frustrated with the process.

Notable examples of CC research include projects like "The Painting Fool" [34, 37], and Cook's dissertation on cooperative co-evolution [41], that included the automated game designer ANGELINA [39, 40].

However, computational creativity research has been around for decades, and early work was inspired by Margaret Boden's work on defining creativity [16, 15, 13, 14]. Boden was also inspired by early work in AI-based generative systems. Boden describes the creative mind and three types of creativity [16], but lends from AI research [14] to provide examples that explain some of her points. Through her three types of creativity, Boden compared human creativity to computer creativity and posed the question: "can computers truly be creative?" While Boden was primarily interested in what made computers appear creative, Ritchie [166] presented ideas for attributing creativity to computer programs.

Boden notes that there are two types of creative ideas [13] p-creative and h-creative ideas. P-creative ideas are new to the person (or system) who makes them. H-creative ideas are new historically, and no one else has had them before. While h-creative ideas are the most valuable on a larger scale to the person making them, they are no different from p-creative ideas. Boden describes the three types of creativity a person (or system) can have: combinatorial, exploratory, and transformational. The three types of creativity form a sort of hierarchy. Combinatorial refers to combining known ideas and concepts to form new ideas. Exploratory refers to ideas formed by exploring conceptual spaces, coming up with new ideas by applying a known style of thinking, and exploring the possibility space through it. Transformational refers to transforming the conceptual space in some way that allows new combinations that were previously impossible to occur, forming previously unthinkable ideas. Transformational creativity can include exploratory and combinatorial creativity as well. Exploratory creativity can include combinatorial creativity. Combinatorial creativity, however, combines other known concepts and ideas and none of the other forms of creativity.

While Boden's work is not a part of CC research, her ideas have been explored and discussed in the context of CC [147]. Understandably, Boden's dimensions of creativity [15] and types of creativity [16] are interesting to CC research, as both combinatorial and exploratory creativity play into strengths of computer programs. As stated earlier, CC research is interested in the creative process itself, and pioneers like Ritchie [166] have put forward ideas for evaluating that process in computer programs. Colton discussed the creativity versus the perceived creativity of computational systems [33]. Jordanous dedicated an entire dissertation to the subject of evaluating computational creativity [99] and Lamb et al. performed an interdisciplinary study of computational creativity evaluation [117].

The field has grown a great deal over the past decade [65] following the growth in machine learning, but also heavily featuring evolutionary computation techniques. Computational creativity research takes on many forms, but one area that combines well with it is mixed-initiative research, where humans and machines share the "burden" of creativity. The primary conference on computational creativity, the International Conference on Computational Creativity

(ICCC)[14], has also explicitly called for co-creative systems, in which two or more agents work together (e.g., several AI systems collaborating or AI and human collaboration). The conference also showcases an *"exhibition of creative artifacts created using computational means, either primarily or as support for a human creator,"* further indicating this mixed creative ownership idea. Even early on, Boden recognized the potential of this collaboration, exemplified by this quote from the opening introduction for her book [16].

The answer to our opening question, then, is that there are many intriguing relations between creativity and computers. Computers can come up with new ideas, and help people to do so. Both their failures and their successes help us think more clearly about our own creative powers.

Among other areas, computational creativity research has spilled into research for games, where Liapis et al. [127] describe games as the "killer app" for computational creativity research. They argue that game development encompasses a large variety of artistic and creative tasks and, as such, provides the perfect platform for computational creativity research. Video games' subjective and complex nature puts an even stronger focus on CC research into co-creative and collaborative systems.

2.6 Mixed-Initiative Tools

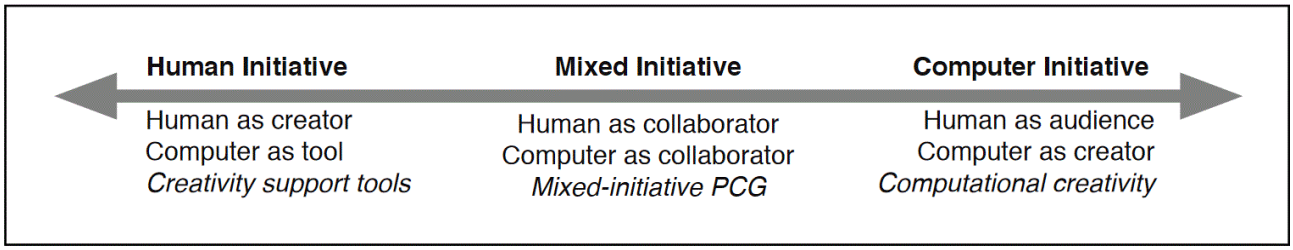


Figure 2.3: The spectrum of agency and initiative distribution in creative interfaces [55]

In the late 90s, Novick and Sutton asked the question "*What is Mixed-Initiative Interaction?*" [144] distinguishing between interactions with pure initiative and mixed-initiative, building on research from the past decade [132]. Mixed-initiative interaction means two or more parties take turns leading the "conversation" or process. This type of exchange is common when people work together to solve various tasks. Mixed-initiative tools thus strive to allow two or more agents, where at least one is a computer system, taking turns to drive the process forward. However, Mixed-initiative agency is a scale (see Figure 2.3), ranging between purely user initiative and purely system "initiative" (after an initial prompt). Any tool where the user and system alternate driving the process forward exhibits mixed-initiative interaction. The definition of tool influences which systems are Mixed-Initiative Tools and the amount of initiative taken by system and user may vary depending on the task and implementation. In

[14]See https://computationalcreativity.net/iccc21/

[233] Yannakakis et al. argue that mixed-initiative tools can foster a beneficial co-creativity, by alternating initiative in a creative process, both user and system take part in the creation and continually "inspire" each other[15].

2.6.1 Mixed-Initiative Tools for Game Development

Creative mixed-initiative tools have been researched in various media. In 2014 Liapis et al. [127] pointed out that games are a medium filled with opportunities for computational creativity and building mixed-initiative co-creativity tools for game development [233] have several attractive benefits. These benefits include wider exploration of design spaces, ideation, and improved iteration. Several of these tools have been created, such as Tanagra [193], Ropossom [180], Sentient Sketchbook [128, 233] all focused on level design with different levels of game specificity. Where Ropossum is targeting the game "Cut the Rope"[16] specifically, Tanagra targets the infinite runner genre, with a focus on the rhythm the player has to tap. Sentient Sketchbook assists in designing 2D-level sketches that it can generate for multiple different game types. Level and puzzle design has had the most attention in this field, combining various PCG forms with different user interaction methods, like interactive evolution, goal specification, and other forms of constraint setting.

Projects like Danesh [45] provide a sort of mixed-initiative interface for the meta-design of PCG systems. Danesh analyses the output of a system and allows the user to define constraints or criteria they want the output to meet. Danesh then adjusts the parameters of the PCG system, attempting to meet the constraints while providing visual feedback about the progress. While the approach could be used on any PCG system, the demonstrations, like much of the mixed-initiative research on games, are mainly based on optimizing level generators.

Newer projects like Lode Encoder [10], or Baba is Y'all[27] combine game simulators and mixed-initiative level generation into online tools, where you can both design and playtest your levels. These tools also allow users to play and redesign each other's levels, making them platforms for co-creative development.

While some projects include other aspects of game design than pure level generation, the tools are still heavily centered around generating levels like SuSketch [136] a sort of spiritual successor to Sentient Sketchbook. SuSketch uses machine learning agents that play a multiplayer first-person shooter level to generate gameplay predictions and suggest alternative level designs. In SuSketch, the collaboration between human and machine fills multiple roles. On one side, the AI agents are used as playtesters in a game simulation, and on the other, the application provides design suggestions and analysis similar to a game designer.

It is obviously hard to separate game mechanics like jumping and movement from level design, which can be seen in projects like Mechanic Maker [171], which builds on Morai Maker [78] (a mixed-initiative AI-driven level editor), to automatically co-create game mechanics for platforming levels. Similarly, Cicero [133] is built as a mixed-initiative interface for game design,

[15] A curated overview of the breadth of tools available on http://mici.codingconduct.cc

[16] https://www.cuttherope.net

based on GVGAI [152, 113, 150, 151] and VGDL [125, 175, 176]. Cicero builds the game using VGDL game descriptions and assists both in designing game and levels. To enhance the design experience, Cicero also provides statistics and context to the otherwise purely text-based representations in GVGAI or VGDL.

Expanding AI assistance to more areas of game design is one of the goals for future mixed-initiative research that is often argued and is also supported by results indicating that assisting designers is a worthwhile cause [134].

Chapter 3

Designing Game Feel

In cases of major discrepancy it's always reality that's got it wrong. This was the gist of the notice. It said "The Guide is definitive. Reality is frequently inaccurate."[1]

Related Papers
Pichlmair and Johansen, *Designing game feel. A survey.*

This chapter addresses the research question *"what is game feel design?" (RQ1)*

In the strive to support the game feel design process through tools, there is a need to establish a language around the task and determining the state of the art within the design area. This Chapter coalesces the nebulous term game feel into three design domains and their purpose. Establishing and presenting a foundation for discussing and analysing aspects of game design through a collection of state of the art design elements within game feel design.

To answer what game feel design is *(RQ1)* we conducted a survey of the relatively new term game feel. Exploring its roots, stretching back far beyond its origin in the late 2000s [215, 214], into several related research fields beyond games and game design. Collecting design knowledge from from practitioners as well as academia and relating it to the act of designing game feel.

[1] From "The Hitchhikers Guide to the Galaxy" by Douglas Adams, 1979

3.1 What is Game Feel?

We began [155] with the definitions put forth in Swink's book *Game Feel: A designers guide to virtual sensation* [214], and from those, we explored and expanded the subject with references from several fields beyond game design, such as psychology, human–computer interaction and other aspects of design research. Swink defines "game feel" as a specific type of embodied experience reachable when the Venn diagram of real-time interaction, spacial simulation and what he calls "polish" is sufficiently well designed that the player experiences the actions of the player character as their own actions. Swink expands his definition, noting that great feeling games convey five kinds of experiences:

- The aesthetic sensation of control
- The pleasure of learning, practising and mastering a skill
- Extension of the senses
- Extension of identity
- Interaction with a unique physical reality within the game

Swink's "Game Feel" is a specific form of game feel that designers may strive to achieve in certain situations and is also inherently defined as "good" game feel. However, game feel design is not limited to games that do not contain real-time interaction or spacial simulation. For instance, Doug Wilson [230] points this out and distinguishes between "Game Feel" and "game feel," with the former focusing on what makes games feel good in the setting described by Swink, and the latter being any feeling a game communicates. The vocabulary for game feel arises both from how the player perceives it and from the designer's intentions. Journalist and game designer Tim Rogers coined the term "friction" in an exhaustive article [168] on how games feel from a player perspective. Interestingly, when Anthropy & Clark [4] defined their "game design vocabulary," they approach friction from the designer's perspective and call it designing the "resistance." Resistance then defines the amount of friction experienced by the player as the feeling of inertia working against the player in the game. Game feel can also be linked with the reflections of practitioners to aesthetic theories of games [58]. Larsen [118] even attempts to define an "aesthetics of action," building on knowledge presented by Swink [214] and Nijman [140] to determine which components of a game contribute to "thrilling experiences." Lastly, Yang [232] expands game feel to include metaphorical aspects of game objects and their connection to the player. By building on queer theory, he connects the feel of games to political aspects to communicate the diversity of the gameplay experience with diverse players. These expansions all provide game developers with a richer set of design tools and lenses [177] with which to view and design how a game feels.

In our survey, we concern ourselves mainly with the haptic and visual aspects of game feel, aware that other aspects such as narrative content, sound, music, and art can influence how a game feels. We provide points of entry for other aspects, where they exist, such as designing

the feel of the story of a game [229]. However, we mainly focus on the moment-to-moment interactivity [115, 188, 215], microinteractions [170], and the interactions with the core loops [184] and their design. More broadly than Swink's precise but narrow definition of game feel, we looked at game feel as the affective aspect of real-time interactivity.

3.2 From Flow to Feel

We first looked at how the physicality of interactivity has been studied by presenting an overview of the history, context and state of the art of the understanding of game feel and how to design it. We based this on research in the field and publications by practitioners, capturing both conceptual and practical knowledge. Years before Swink wrote "Game Feel" in 2009, research on the connection between emotions and gameplay was often linked to flow theory [50], which presented the results of a global research project about experiences that are "so gratifying that people are willing to do it for its own sake, with little concern for what they will get out of it, even when it is difficult or dangerous." The study found tasks that elicit flow fulfill eight criteria:

1. a task that can be completed;
2. the ability to concentrate on the task;
3. that concentration is possible because the task has clear goals;
4. that concentration is possible because the task provides immediate feedback;
5. the ability to exercise a sense of control over actions;
6. a deep but effortless involvement that removes awareness of the frustrations of everyday life;
7. concern for self disappears, but sense of self emerges stronger afterwards; and
8. the sense of the duration of time is altered.

Sweetser and Wyeth [213] connected flow theory to games, and Juul [103] criticizes the theory's relevance when describing enjoyable challenges in games. Ciccoricoo [30] discusses flow in relation to Mirror's Edge [56], contrasting the game's fluidity of movement with the feminist concept of fluidity. Chen [28] famously based not only his graduation thesis but also three successful games released by his studio, That Game Company, on his understanding of the concept.

3.3 Juice the Feel Amplifier

We argue that of the eight elements that allow for a flow experience [50], game feel is affected by the clarity of the goals (3.), how the feedback is provided (4.) and the correlation between input and action (5.). Game feel is focused on the role of interactivity in the process rather than the dynamics of immersion from flow theory. A concept that often appears when discussing the interactivity and its intensity is "juice." Juice amplifies the interactivity by providing excessive amounts of feedback in relation to player input [104]. We note that *the goal of juice is to make actions feel significant. It is superfluous from a strictly mechanical perspective but turns interacting with the system into a more pleasurable experience.* However, there is an adequacy to juice [105], in that adding too many effects on the screen can make it difficult to learn which aspects of interactivity have mechanical importance [57] while adding too few can make the game feel dull. However, because games are a diverse medium, games that are almost purely made of juice exist; they are referred to as "toys" or "autotelic experiences" [183]. These toys present playful experiences based on the feedback amplified by juice.

The overarching goal of applying juiciness is to enhance the feedback when interacting with game objects. In the hope of determining the correct amount of juice in video games, Hicks et al. [84] presented a framework for analyzing juiciness in games, building on Juul's [104, 102], Schell's [177] and Deterding's [54] work on juice in video games. However, juice can also exist as a spectacle for the audience watching the game, in addition to the player's experience. Rogers [168] hints at that, explaining that once the player knows the mechanics intricately, they can ignore the little juicy details that initially draw them in and provides feedback about the interaction initially. While this might seem to make the feedback irrelevant later on, Gage [69] describes how having a "readable" game can draw people in. Gage notes that if a game is appealing at multiple levels of "subway legibility," the audience—whether invited or a casual passerby—can get a deeper understanding of the game as they focus in on more and more details. In the era of streaming services like Twitch [2], developers have a higher and higher incentive to appeal to spectators, not just within e-sports [25] where the spectator and player interfaces have diverged to accommodate commentators, spectators and players [3].

Coming back to juice, Hunicke [88] says, "juiciness can be applied to abstract forms and elements, and it is a way of embodying arbitrarily defined objects and giving them some aliveness, some qua, some thing, some tenderness." This is strikingly similar to the term "polishing" used by Swink [214], Larsen [118] and Fullerton [68], who describes the act of polishing as "the impression of physicality created by layering of reactive motion, proactive motion, sounds, and effects, and the synergy between those layers." Fullerton views polish as giving physicality to inanimate objects to render them more tangible, similar to Hunicke's reasoning for juiciness. In game development practice, many things use the term "polish," like fixing timings in voice

[2] https://twitch.tv/

[3] Examples are MOBA games like DOTA2 *https://www.dota2.com* or League of Legends *https://www.leagueoflegends.com*, RTS games like StarCraft *https://starcraft2.com*, and FPS games like Counter-Strike *https://blog.counter-strike.net*

cues or fixing bugs in the code [201]. Juiciness and polish are linked in the sense that all juicy elements are polished at some point, but polish is seen mostly as an aesthetic task that stops short of changing the basic rules of the game, the narrative or the core loops. While the intentionality behind Hunicke's and Fullerton's comments is apparent, the comments are not very descriptive. However, Lisa Brown's assertion that "you're not juicing your game – you're actually picking a feeling that your game should communicate and juicing that feeling" [18], makes the intent of juicing more clear.

3.4 Three Design Domains

After looking at the theory, we turned to game feel design elements in practice, gathering blogposts, conference talks and online discussions about various design elements. Some game designers discussed approaches to game feel in general, in podcasts [208, 209, 210, 211] and video analysis [19, 20, 31]. We also gathered material breaking down specific aspects of game feel, and we began forming an overview[4] of the various design elements in game feel design (see Table. 3.2).

We broke down the design elements into five categories: Movement and Actions, Event Signification, Time Manipulation, Persistence and Scene Framing. Each category is subdivided into several different established concepts, and the references where they are explained or more info is available. An example is coyote time, which is named after the coyote in the Road Runner cartoons, as it often finds itself suspended in mid-air for a period of time after running off of a cliff. Coyote time provides the player a grace period after they move off a platform or ledge where they can still jump. The time is usually somewhere around 100–200 ms (though some designers define it through pixel distance [172] or through a framecount [160]), to offset the average human reaction time. In addition to these categories, we also broke down game feel design into three design domains and their purpose (see Table. 3.1). These three domains are Physicality, which is the foundation Swink's [214] entire Game Feel

[4]We do not claim this is a complete overview, but it serves nicely as an initial overview.

Design Domain	*Physicality*	*Amplification*	*Support*
Domain Description	The physical simulation.	Intensification of experience.	Enabling the player.
Polishing Task	*Tuning*	*Juicing*	*Streamlining*
Task Description	Setting parameters to specify the behaviour of objects.	Adding feedback to emphasise, clarify and amplify.	Acting on player intent by interpreting the input in context of the gameplay situation.

Table 3.1: Game Feel Design Domains and their associated Polishing Tasks

concept rests upon. It is the design of the experienced physicality of the system. The second domain is Amplification, which serves two purposes: it empowers the player and communicates the importance of events. This is what Jonasson and Purho show in their "Juice it or Lose it" talk [98]. The third domain is Support; this domain covers techniques for assisting the player in performing their intended actions. This domain can lean towards accessibility [70] and the minute details Doucent discusses in his blog post "Oil it or Spoil it" [57] (a response to "Juice it or Lose it"). In his book, Swink [214] states that the real-time control in a simulated space needs to be amplified by polish. We argue that each of these three design domains has its own polishing task. We call these Tuning the Physicality, Juicing the Amplification and Streamlining the Support. In Table. 3.2, we have noted which design domains each design element belongs to. Our point here is to show how every task and domain is significant and important to various aspects of a game and how they all influence the feel of the game.

We also note that our survey, while extensive, is a launching point for several deep dives into certain aspects, design elements and design domains.

3.5 Discussion and Future Work

In this survey, we drew together every angle we could find related to game feel design. We had some initial ideas about just how extensive the research on this topic was. However, we also knew it was scattered in many different fields and had many different names depending on the perspective of those fields. Games are a medium that pulls together many different disciplines, and only partial segments of game feel design are explored in each of those disciplines. With this paper, we highlight knowledge that could easily be missed when thinking about game feel design and its many origins (see Figure 3.1).

As alluded to earlier, this survey is meant to serve as a general introduction to the current state of the art within game feel design. By defining three design domains and identifying common design elements, the answer to *"what is game feel design" (RQ1)* emerges as tuning the physicality, juicing the amplifications and streamlining the support. These three polishing tasks can be applied in order to push the game experience as close to the design vision as possible.

Some of the design elements we identified in this survey may overlap. Freeze frames and bullet time (see Table. 3.2) are good examples; both fall under time manipulation, and although they both offer support, they each have different affordances [142] and are useful in different circumstances. Future research into these types of overlaps could reveal more design elements or clarify their use further.

We intentionally moved quickly past certain aspects that affect the overall feelings perceived by the player as they play the game, such as narrative, setting, aesthetics, rules, music and sound effects. Future research into these aspects may offer additional design elements that fall under the design domains of physicality, amplification and support. Additionally, some of these aspects contain their own design domains and polishing tasks that are worth pursuing in future

research. For example rules might fall under the design domain of game mechanics, in charge of making the systems interactive and with the polishing task "Balancing" attempting to keep the interactions between dynamic systems from degenerating.

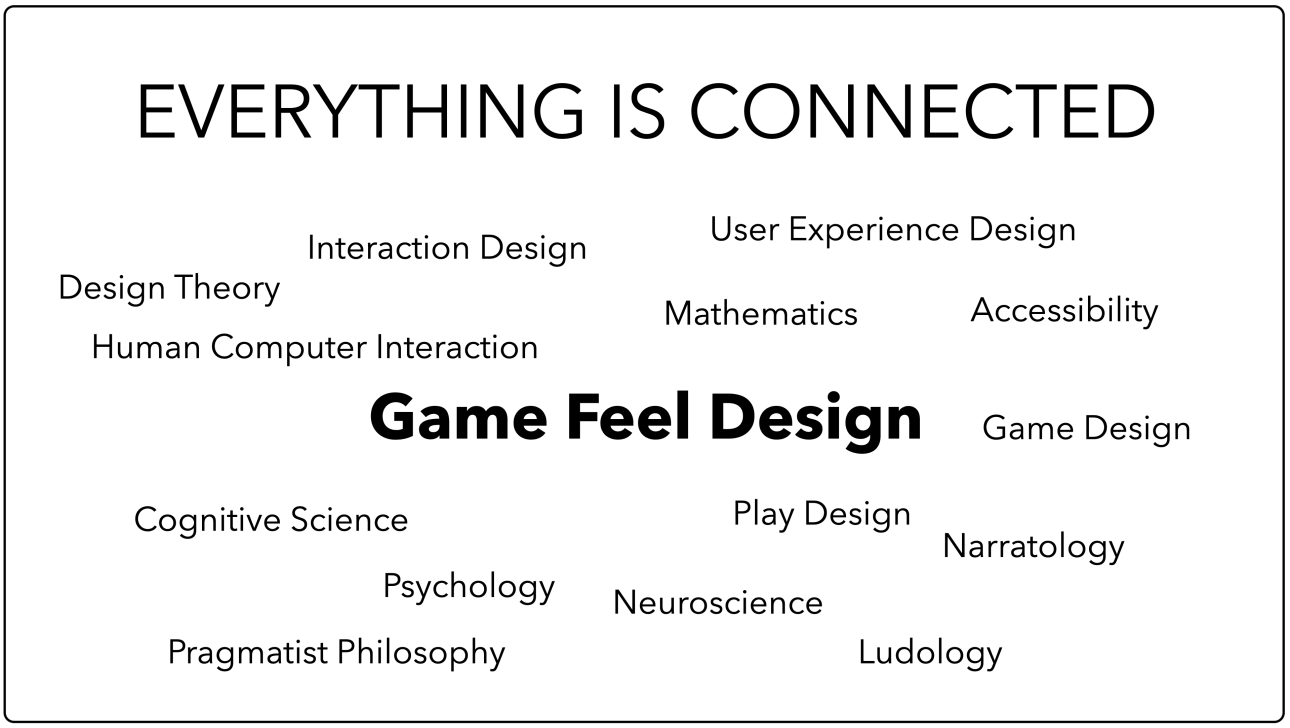


Figure 3.1: A slide from our presentation at the Conference on Games 2021, showing the many topics and contexts from which we collected data used in our paper [155].

With the "what" *(RQ1)* addressed and an understanding of game feel design established, two things become clear. First, the feel of a game is not hinged on a single design element, but rather on a collection of elements influencing the moment-to-moment interaction at any given time. Second, while some design elements of game feel design can be applied relatively superficially without impacting game logic (like screen shake and one-shot particle effects), many design elements are deeply entangled with the physics and core logic of games. The more entangled a design element is, the harder it becomes to support that element through generalized tools. Instead these aspects need specialized tools to deal with the exact aspect in isolation, or customized tools for the specific game. This applies to design elements like basic movement, coyote time and assisted aiming. Aspects such as these are deeply dependent on the game goals and mechanics: is the game 2D or 3D? is the gameworld continuous or turn-/grid-based? does the dangerous elements (in the case of coyote time) move in one or multiple dimensions?

This leads to an insight: the more generalised the game feel design support needs to be, the more it will have to rely on relatively superficial elements. While that is not ideal, the elements that cover most ground in all three game feel design domains are related to event signification, and these are almost all relatively superficial aspects relying heavily on audio-visual feedback. The event signification elements are central to communicating both the state and events more

clearly to the player, as well as signifying relationships and repercussions of certain events. They are at the core of the amplification domain allowing the game to communicate information "between the line", and part of what gives the game some aliveness, some qua, some thing, some tenderness [88]. As such while tuning the physicality and streamlining the support, provides the feel of the material properties and friction of the game. The amplification provides the language the game uses to convey both the material and immaterial properties the game might have. In the following chapters the emphasis rests on (but not limited to) design elements related to event signification and it remains as future work to create and research tools for the remaining aspects of game feel design.

Design Element	Physicality	Amplification	Support	Key References
Movement and Actions				
Basic Movement	•			[51, 143, 156, 62, 157, 38, 205]
Gravity	•			[62, 122, 2]
Terminal Velocity			•	[62]
Coyote Time			•	[173, 224]
Invincibility Frames			•	[189, 137, 187]
Corner Correction			•	[72, 57]
Collision Shapes	•		•	[231]
Button Caching			•	[62]
Spring-locked Modes	•			[162, 97, 4]
Assisted Aiming			•	[57, 108, 236]
Event Signification				
Screen Shake	•	•	•	[140, 98, 149, 186]
Knock-back & Recoil	•	•		[140, 6]
One-shot Particle Effects		•	•	[164, 91, 120, 130, 98, 167, 222]
Cooldown Visualisation			•	[47, 77, 114, 5]
Ragdoll Physics	•	•	•	[92, 214]
Colour Flashing		•		[153, 106, 140]
Impact Markers		•	•	[199, 196]
Hit Stop		•	•	[52, 90, 196, 20]
Audio Feedback	•	•	•	[9, 66, 138]
Haptic Feedback	•	•	•	[146, 196]
Time Manipulation				
Freeze Frames		•	•	[196]
Slow Motion		•	•	[196]
Bullet Time		•	•	[169, 158]
Instant Replays		•	•	[196]
Persistence				
Trails			•	[164, 9, 145]
Decals & Debris			•	[11, 9]
Follow-Through	•			[217]
Fluid Interfaces	•		•	[107, 73, 26]
Idle Animations			•	[3, 48, 217]
Scene Framing				
Highlighting			•	[110, 109, 135]
Dynamic Camera		•	•	[29, 80, 81, 82, 23, 234, 153]

Table 3.2: Game feel design elements overview.

Chapter 4

Supporting Game Feel Design

We are stuck with technology when what we really want is just stuff that works.[1]

Related Papers
Johansen, Pichlmair, and Risi, *Video game description language environment for Unity machine learning agents*
Johansen, Pichlmair, and Risi, *Squeezer - A tool for designing juicy effects*
Johansen, Pichlmair, and Risi, *Squeezer - A mixed-initiative tool for designing juice effects*
Johansen and Cook, *Challenges in generating juice effects for automatically designed games*

This chapter describes various efforts carried out during this thesis in an attempt to answer *"how can game feel design support get implemented in existing workflows?" (RQ2)*

Before asking how support can be implemented in existing workflows, it is interesting to reflect on where this need is coming from. Chapter 5 describes standard practices for designing game feel today. However, it is interesting to think about the lack of inherent game feel design support in many frameworks and game engines before getting to that. While some engines provide fundamental support for a few core effects (often play sound and make simple particles) or provide general ways of implementing it (e.g., through code), developers often use external or custom code and tools. Big engines like Unity provide several different systems to make juice effects, but they are often disconnected and require expert knowledge to wield proficiently. Game feel design is an essential part of designing a game. However, prototyping and game generation frameworks often lack support for making the moment-to-moment interaction feel just right. Getting this part of the interaction right is something that experienced designers will often focus on early in the development process—allowing it to become a foundation to build everything else around. Often called the core loop or the main interactions, they are the tiny interactions the player repeatedly does throughout the game. These interactions need

[1] From "The Salmon of Doubt" by Douglas Adams, 2002

to elicit proper responses aligned with the gameplay to provide the player with the intended experience. If a game has a lot of mouse clicking (e.g., a point-and-click adventure or a digital card game), the designer has to make sure that clicking provides a satisfying response. However, most game creation systems require both knowledge and proficiency before it is even possible to experiment with different response designs.

This chapter outlines experimentation with adding support for game feel design into three different workflows. The experiments were stepping stones in a practice-based design research sense, each informing the next. The "program" or frame of reference for the experiments is defined by the goal of discovering how game feel design support can get implemented in existing workflows *(RQ2)*. The program and different research directions explored in this chapter have been the driving force behind the research in this Ph.D. project. The experiments in the first section are defined by the question, "how can game description languages become better at supporting game feel design?" The second section takes ideas from the design experiments with the video game description language (VGDL). It is based on the underlying question, "how can we leverage game description structures to build support tools for game feel design?" The third section builds on the second through the question, "how can tools for game feel design get implemented in an automated game design system?"

Each section discusses the unique challenges of adding game feel design support within each existing workflow. The entire chapter is wrapped up with a discussion of how each experiment contributed to answering how game feel design support can get implemented in existing workflows *(RQ2)*.

4.1 Game Description Frameworks

The question "how can game description languages become better at supporting game feel design?" Formed after surveying existing mixed-initiative and generative game tools early in the project. The survey was initially performed to discover how much game feel design had been explored through tools. While many projects offered new and insightful ways of interacting and supporting game designers [55], the focus often revolved around game development aspects such as level design, game mechanics (rules and goals of the game), and sound or music generation. The design of aesthetics and perception of the game remained largely unexplored within this branch of research.

The overall focus of this Ph.D. project is design exploration and, in particular, prototyping within the subject of game feel design. Two projects inspired the first experimental direction. The first one being the Video Game Description Language (VGDL) [175, 125, 152, 113, 150, 151] a framework for generating and playing the described games. The concise nature of VGDL descriptions makes them easy to understand, and their ability to generate relatively complex games makes them an interesting starting point for design research. The second project is the sound effect tool SFXR [154] for its generative properties allowing novice game designers or game jammers with limited time to create sound effects there are "good enough" quickly. How

SXFR influenced the development is described more in the next section on building generalized tools for a game engine, as well as in Chapter 5 concerning sequence generation.

Combining the idea of design exploration with VGDL to support game feel design led to the question, "how can game description languages become better at supporting game feel design?" The experiments in this section are based on a hypothesis that can be summarized as: *It is possible to describe and execute event-based feedback (event signification), similarly to how VGDL describes and executes event-based game logic.*[2]

Answering how game feel design support can be implemented *(RQ2)* in game description languages (GDLs) is done through two distinct avenues. The first is based on transcribing effect descriptions in industry talks such as "Juice it or lose it" [98] and "The Art of Screenshake" [140], to derive an initial event signification ontology. The second part was the experiments to determine the viability of combining event-based feedback with a GDL. VGDL was selected as the primary platform for the second part of the exploration based on prior knowledge and source code available at the time.

It bears mentioning here that VGDL is just one among many game description languages that could have been selected for this project. Frameworks like Puzzlescript[3] by Stephen Lavelle or Ludi/Ludii [22, 21, 200] could also have formed the basis of this exploration. However, Puzzlescript descriptions become verbose very quickly[4]. While Ludii has later been opened to the public, it was not available when these frameworks were explored. Ludii also has a strong focus on board games, and while the platform might benefit from adding more game feel elements, this project lends itself better to the arcade-style action games of VGDL.

4.1.1 Video Game Description Language

The Video Game Description Language (VGDL) described by parts of the game AI research community [125], implemented initially in Python by Tom Schaul [175, 176] and later adapted to Java for the general video game AI competition (GVGAI) by Perez-Liebana et al. [152, 113, 150, 151].

[2] It was later extended to include generation of feedback by taking inspiration from SFXR.

[3] https://puzzlescipt.net

[4] A fact the author experienced by implementing the game "BABA IS YOU" in Puzzlescript https://pyjamads.itch.io/baba-is-you-demake/ while considering the language.

VGDL was designed to be part of the grand challenge of general video game AI [125], and it was intended to be a stepping stone toward a so-called general artificial intelligence. VGDL is based on real-time interaction, and its goal is to be able to specify 2D arcade games, such as those found on the Atari 2600.[5] VGDL specifications are separated into five sections[6]:

- Sprite Set
- Interaction Set
- Termination Set
- Level Mapping
- Levels

Figure 4.1 shows an example of a VGDL game specification. *Sprites set* defines the objects used by VGDL, including their behavior and properties. *Interaction set* defines what happens when sprites overlap. *Termination set* determine when the game is over. *Level mapping* connects the sprite names to ASCII characters. Last, *levels* in VGDL are defined in separate files. The game does not automatically serve the next level when a level ends. Instead, each level is treated as an individual game, passing the result to the overarching system. While VGDL was created for AI testing, it allows human players to play the games, a feature used mainly for familiarization and testing game rules.

To condense the descriptions and simplify input handling, VGDL uses an "ontology" of predefined classes that define sprite behavior types, interaction types (also called "effects"), and termination types. Sprite behavior types handle input and movement for objects. It is important to note that the player input classes are meant to handle only one player character on screen at a time. Interaction and termination types are similarly based on predefined actions or conditions. Predefined actions and conditions make the descriptions very concise, at the cost of expressiveness. It is possible to adjust the behavior of sprites, interactions, and terminations by adding parameters in the specification, but they still rely on predefined logic.

VGDL loads a set of rules and instantiates a level from a separate file. Separate files for rules and levels make sense for game AI testing, as the system can load levels in any order or use different rules for the same level. However, it also means that creating an overarching experience across multiple levels or screens is not possible with the current VGDL implementations. Any sense of progression beyond collecting wins, losses, or scores has to be controlled by another system.

The GVGAI version of VGDL does allow sound effects to be played by some interactions, which is a curiosity, as the sound feedback is imperceptible to the AI agents. In addition to sounds, the GVGAI framework also allows small animations and has a system for automatic tile

[5]The Atari 2600 games have also been used for machine learning in the Arcade Learning Environment (ALE) [8].

[6]In VGDL, levels are defined in files separate from the game description.

```
BasicGame
 SpriteSet
  hole   > Immovable color=DARKBLUE
  avatar > MovingAvatar
  box    > Passive
 LevelMapping
  0 > hole
  1 > box
 InteractionSet
  avatar wall > stepBack  #stop at wall
  box avatar  > bounceForward #push box
  box wall    > undoAll #wall stops box
  box box     > undoAll  #box stops box
  box hole    > killSprite #destroy box
 TerminationSet
  SpriteCounter stype=box limit=0 win=True
```

Figure 4.1: Game description for a simple Sokoban game in VGDL. Objects defined in the *SpriteSet* can be used to define interactions, terminations, and level mapping. The VGDL Effects such as *stepBack* have notes like "#stop at wall" to explain what they do when the two object types collide. The game is won by pushing boxes into holes until no boxes are left.

```
wwwwwwwwwwww
w          w  w
w   1         w
w   A 1 w 0ww
www w1  wwwww
w         w 0 w
w 1           ww
w             ww
wwwwwwwwwwww
```

Figure 4.2: VGDL Sokoban level description. 'A' denotes the Avatar (controlled by player), 'w' denotes walls, '1' denotes boxes and '0' denotes holes.

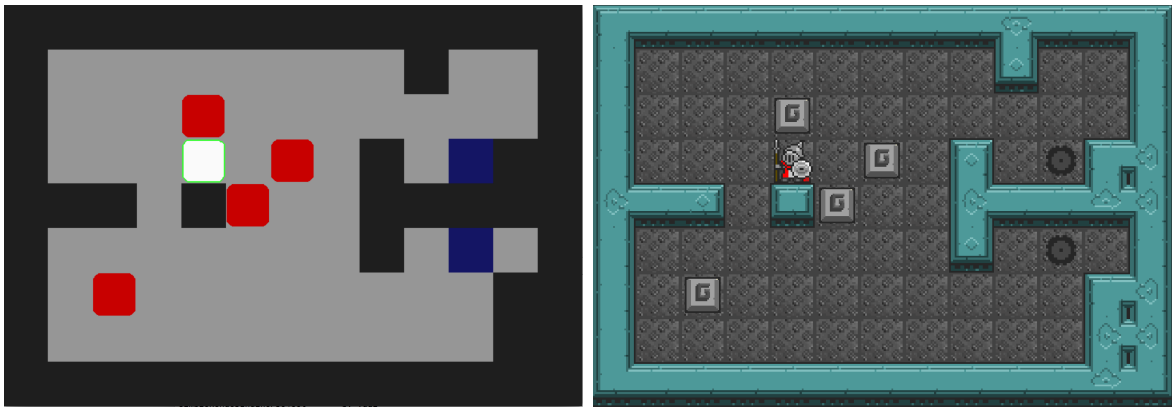


Figure 4.3: Visualizations of VGDL Sokoban: left PyVGDL, right GVGAI

placement. With all of the above in mind, VGDL could be a fantastic prototyping tool for game designers because of its very condensed and human-readable descriptions. However, it would need modifications to allow proper level progression and potentially data persistence between levels. Additionally, adding more game feel elements like better player controllers and visual feedback would allow the games to communicate actions through event signification better.

4.1.2 Extending VGDL with Game Feel Elements

In order to answer how game feel design can be supported *(RQ2)* through GDLs, we first identify what type of extensions would be interesting to make. Based on the earlier hypothesis, an unpublished experiment was defined. The hypothesis states that it is possible to describe game feel elements for event signification like VGDL describes interaction effects. This experiment required two steps. The first was the transcription of every event and accompanying effect demonstrated in the two talks "Juice it or lose it" [98] and "The Art of Screenshake" [140]. The second step was to explore how these effects could be described and parameterized similar to other interactions in VGDL.

The transcription of the two talks provided around 10-15 types of effects, ranging from screenshake and particle explosions to color changes and sounds. In Table 3.2, most of them belong under event signification. Although the transcriptions have not been published or used directly, they did contribute a lot to the list of game feel design elements seen in Table 3.2 in Chapter 3.

"Juice it or lose it" also provides an excellent test case for the descriptions, as the example game used in the demonstration already existed among the VGDL game example. The game is a Breakout clone, and it features a single screen with destructible blocks, walls, a ball, and a paddle controlled by the player.

It took several iterations to include the additional effects and features in the altered descriptions. An example of the imaginary extension of VGDL can be seen in Figure 4.4. The example describes the interaction and event signification to be executed when the ball and a block in Breakout collide. This example features the two primary examples of format changes

in *"VGDL+"*[7], one is defining multiple effects for a single interaction, the second is the *"Other"* feature, which changes the target from ball to block for the same interaction. This nesting removes two issues, the need to specify the interaction multiple times and any potential confusion about the execution order of the effects. For instance, the description in Figure 4.5 contains two interactions for the ball and block collision. The first definition specifies that when a block gets hit by a ball, it should be destroyed. However, the second definition states that when the ball hits a block, it should bounce off it. If executed in the order the interactions are defined, the block would be destroyed, and the ball would never bounce. In reality, VGDL executes them in reverse order, allowing the ball to bounce before the block is destroyed. In *VGDL+*, the order would be determined by the list of within a single interaction. Additionally, the *Other* feature allows a designer to inject when effects are executed on the opposite target.

Looking back to Figure 4.4 we see that the basic "bounceDirection" (which changes the ball's trajectory) remains in place in this description. However, the bounce impact is emphasized by screenshake, sound effects, color changes, size changes. In addition to that, instead of simply destroying the block, the block is transformed, and a nested list of effects is executed on the transformed object, making it fall off the screen while rotating, scaling down, and changing color. Though *VGDL+* has never been implemented, description experiments like those in Figure 4.4 make it easy to get an overview of the resulting interaction effects. Just like the description of Breakout in Figure 4.5, it makes it easy to understand the rules and mechanics involved (once you know what building blocks like "bounceDirection" do).

```
ball block >
    bounceDirection #VGDL bounce off the wall effect
    #### VFX / SFX BELOW ####
    playSound id=XXXXXXXXXX volume=0.5 fadetime=0.1 fade=INOUTLINEAR    #blockhit creates
    different sound
    screenShake duration=0.2 intensity=0.3 #shakes the whole screen

    #Impact
    colorYoyo duration=0.2 color=WHITE easing=INOUTSINE
    colorYoyoAll stype=background duration=0.1 color=WHITE easing=INOUTEXPO
    scaleYoyo duration=0.2 relative=TRUE x=2 y=2 easing=INOUTSINE
    scaleYoyoAll stype=block duration=0.1 easing=INOUTBACK relative=TRUE x=1.5 y=1.5

    #Aftereffect
    scaleYoyo delay=0.2 duration=0.5 relative=TRUE x=1.2 y=1.2 easing=INOUTBACK repeat=5 #
    repeat means loop, default is 1, infinite=-1

    Other > # VGDL+ feature, to reverse the sprite1 and sprite2 parameters to effects
        transformTo stype=fallingBlock > #nesting feature, transformTo will call the effect
        list on the newly generated fallingBlock
            killSprite delay=5 #New VGDL+ feature, destroying it after a delay (seconds)
            scaleSprite relative=FALSE x=0 y=0 duration=5 easing=OUTLINEAR
            rotateSprite duration=5 rotation=5 #rotation in degrees per second
            colorSprite duration=0.1 color=DARKBLUE
```

Figure 4.4: Sample description for the imaginary extension of VGDL, here noted as *"VGDL+."* It describes the interaction between ball and block in breakout, with the juice effects presented in "Juice it or Lose it" [98].

[7]More feature ideas and format changes were explored to provide better design choices related to prototyping richer game experiences. However, those remain out of scope in this context.

From the experiments with *VGDL+*, we can see that it is indeed possible to describe specific game feel elements related to interaction events. As discussed in Chapter 3, the amplification domain of game feel design (also known as juicing) often involves multiple layers of effects when emphasizing game events. The extended *VGDL+* format shows that with minor modifications to the format, VGDL could attach multiple effects to a single interaction event. Besides listing multiple effects, another critical feature in *VGDL+* is delaying effects a certain period after the interaction occurred. Delays are essential to adjust each effect's relative timing, so all animations, sounds, and particle effects complement each other perfectly. Additionally, the concept of waiting for other events to finish (seen after the "transformTo" effect in Figure 4.4) builds a foundation for the tree structure used in Squeezer (see Section 4.2).

In a more general sense, the structure allows event signification effects to be listed in an easily readable format, like the rest of VGDL. Nested lists of effects can be used to describe sequences of effects that occur simultaneously, sequentially, or both.

Comparing the size of the extended interaction description in Figure 4.4, to the description of the entire interaction set in Figure 4.5. It becomes clear that the game descriptions would grow significantly in size with this amount of granularity in the feedback effects. One way to counteract that would be implementing overarching concepts like "impact" as an effect type that executes a set of juice effects like scaling, color-changing, screenshake, and sounds. However, keeping an extended format allows every effect to be easily understood and linked to a collision event. The granularity of the ontology of effects determines the designer's freedom and ability to fine-tune, juice, and streamline the game feel.

```
InteractionSet
    ballStart block wall > transformToAll stype=ballStart stypeTo=ball

    block ball > killSprite scoreChange=2

    ball avatar > bounceDirection
    ball wall > wallBounce
    avatar wall > wallStop

    ball EOS > subtractHealthPoints stype=avatar value=1
    ball EOS > transformTo stype=ballLost
    ball block > wallBounce
```

Figure 4.5: The entire *InteractionSet* for the basic Breakout game in the VGDL library.

The experiment shows is that there is a tradeoff between designer expressiveness and description conciseness. The granularity of the effect ontology impacts the designer's expressiveness. The more granular, the more expressiveness, but the less concise the descriptions become. Besides the granularity, the nesting and sequencing options play a crucial role in supporting game

feel design. Without this nesting, it can be hard to tell the order of operations when an interaction is specified multiple times. Sequencing effects through delays and nesting allows the designer to layer multiple effects to emphasize the event how they want.

The exploration of *VGDL+* shows one possible way to support game feel design *(RQ2)* in game description languages. This experiment only covers event signification on a single game description. A more comprehensive range of games and design elements should be used in future explorations of game feel support for game description languages.

4.1.3 Implementation Options

The hypothesis also includes the possibility of executing event signification effects based on interaction events. These effects require a few things not present in current VGDL implementations. The first is the ability to execute effects over a duration, updating them a little at a time. While the GVGAI framework does allow sound effects, they are handled through a "fire and forget" system. The only effect close to working in this way in GVGAI is the "Flicker", which creates an object for a few frames before removing it again. Manipulating objects over time by changing their color to white and back, or altering their size or position for a split second, is vital to emphasize events like getting hit by a projectile. Manipulating the passage of the game time itself, using freeze frames or bullet time is a common way to allow the player to soak up the event. The VGDL implementations also lack the concept of a camera, which means an effect like screenshake would have to move all the elements in the game world around to achieve the same effect as moving the camera. Screenshake comes from film, where shaking the camera simulates the effect of shock waves from explosions or the movement of driving a car.

Two reasonable options for implementing event signification support are extending a previous implementation of VGDL or building game prototypes based on VGDL descriptions for a workflow that already supports it.

Extending the General Video Game AI Framework

Through working with the GVGAI framework, during the development of UnityVGDL [96] and through the VGDL language modification experiments, a few things became clear. The GVGAI framework is heavily optimized towards forward planning (also known as search-based AI). Interaction effects are never updated after execution. Updates are tick-based, meaning they are indifferent to the actual time that passes between each update. Lastly, the rendering system is built to handle simple sprite rendering.

These features make GVGAI a highly efficient system for game AI research, but they make implementing non-trivial time-based audio-visual effects harder. Making certain effects possible would require the rendering system to be adjusted, but more importantly, many visual effects are executed over a duration. Executing effects is complicated (and potentially inconsistent) in a tick-based update system. But even worse in VGDL, updates are tied to "Sprites" (game

objects), and effects are only executed upon interactions. In order to execute and adjust juice effects over a duration, the system would need to handle them separately from other effects. Lastly, to support time manipulation effects, such as freeze frames or bullet time, the system would need to handle updates at alternative time scales.

While it would be possible to make these changes to GVGAI, it is likely infeasible to alter the framework that much. Part of the motivation behind UnityVGDL was moving VGDL into a modern game engine and moving away from the competition framework in GVGAI. The other option of extending an existing VGDL implementation would be the original VGDL [175, 176] implementation in PyGame [8]. Unfortunately, the PyGame implementation of VGDL has not been updated for a long time and is missing the newer ontology elements available in GVGAI. All-in-all, this makes both GVGAI and PyVGDL relatively unattractive as a foundation for supporting game feel design in game generation frameworks.

Using a Game Engine

One alternative to extending and modifying the GVGAI framework to support various visual effects is to move the VGDL core into a fully functioning game engine, where these types of visual effects are easy to add. Similarly, in 2016 game developer Alan Hazelden presented a talk[9] about his workflow, prototyping Puzzlescript games and importing them into Unity. In Unity, the objects could be re-skinned, decorated and the game feel could be adjusted to match the gameplay (see Figure 4.6 for an example). In the talk, Hazelden describes how he uses the Puzzlescript interpreter in Unity to import game logic and levels. Embedding the Puzzlescript game in Unity allows Hazelden to quickly try out new mechanics and levels by modifying the underlying descriptions. Puzzlescript and VGDL have many similarities in their description languages, and Hazelden's workflow demonstrates that interpreting descriptions in Unity has some elegant design advantages. Figure 4.6 shows a Puzzlescript game jam prototype by Hazelden, on which the mobile game *Cosmic Express* [61] is based.

4.1.4 UnityVGDL

UnityVGDL developed as a starting point for working with VGDL in Unity [96]. UnityVGDL directly implements the GVGAI core in Unity and C#. Making it possible to interpret and run VGDL games in Unity, as well as using the newly created MLAgents framework [101, 100] within Unity for game testing and playing. In [96] we compared MLAgents to machine learning and planning agents in GVGAI [220]. We also believe that in the spirit of general video game playing AI, the GVGAI competition could be extended with a transfer learning track, where an agent is trained to play a VGDL game in GVGAI while getting tested on a different interpretation (such as UnityVGDL).

[8] see https://www.pygame.org/

[9] Presented at Game Developers Conference 2016 in the Tech Toolbox panel. https://www.gdcvault.com/play/1023377/Tech

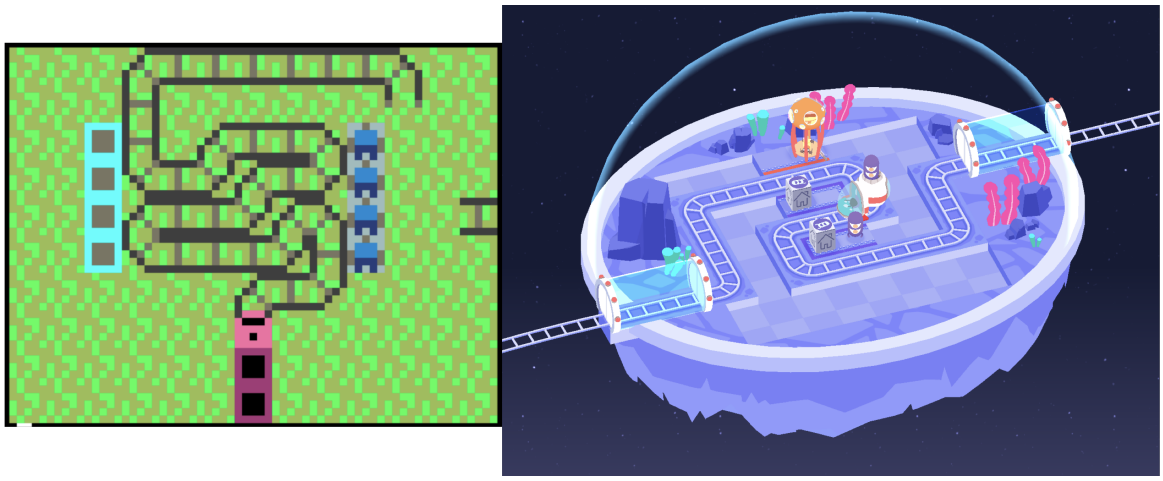


Figure 4.6: Left *Train Brain* game jam prototype. Right the mobile game *Cosmic Express* by Cosmic Engineers [61]

4.1.5 Building Unity prototypes from VGDL

The secondary purpose of UnityVGDL was to build Unity prototypes. An unfinished version of the workflow was created by Luis Fernando Laris Pardo (an MSc student at the IT University of Copenhagen). Pardo managed to generate Unity prototypes from VGDL descriptions, attaching scripts to game objects and using the collision system to drive the interactions between otherwise self-contained objects. Later we tested adding juice effects to the generated games[10] using an early version of Squeezer (see Section 4.2).

4.1.6 Handling VGDL Interactions in Unity Prototypes

To determine the object types and the interactions to apply when collisions happen. The game objects were tagged with the VGDL sprite type, using the Tag system in Unity. In Unity, this approach to event handling is a commonly recurring pattern for game prototypes, as it is convenient and straightforward to use.

Unfortunately, the approach used to build the prototypes was a bit messy as it relied on the same interaction classes translated from GVGAI to UnityVGDL. Using the interaction classes directly created a ton of friction in the implementation. The VGDL game system drives all game logic from a central location, and the objects in the Unity prototypes were using a decentralized script system. With that in mind, a VGDL interpreter in Unity would benefit from custom implementations of the entire ontology. Such a reimplementation could potentially also benefit the game feel of the generated VGDL games. Implementation details and additional adjustable parameters could support designers by allowing them to adjust objects and interactions more meaningfully.

[10] An example of VGDL Sokoban with simple juice effects can be found here: https://github.com/pyjamads/Squeezer/blob/master/Showcase/misc/tinysokoban.gif

4.1.7 A Generalizable Pattern Appears

While the prototype generation experiment was less successful, the analysis of how to handle VGDL interactions for the generated Unity prototypes led to a realization. The generated prototypes used the collision and tag system to determine which interactions should occur when a collision happened. This recurring pattern in Unity prototypes presented an opportunity for bypassing the game generation process entirely and skipping right into supporting game feel design in Unity. By taking advantage of this typical pattern, a tool could listen for interactions and apply juice effects to the objects involved.

4.1.8 Discussion

This section discussed how VGDL descriptions could be altered to include specific game feel elements. It also touched upon ways to execute this type of interaction feedback in GVGAI and how UnityVGDL was used as a stepping stone to build VGDL games in Unity. Most importantly, a generalizable pattern for detecting interactions in Unity was found. This pattern moved the work on extending workflows with game feel support into the next phase. Namely, making a tool for combining the power of effect sequence descriptions from the *VGDL+* experiments with detecting specific interactions in Unity prototypes.

4.2 Squeezer

Squeezer [95, 94] is arguably the most developed and user-friendly project in this thesis, and it is revisited in Chapter 5 and Chapter 6. Squeezer is a tool for designing juice effects in Unity. Initially based on the generalizable approach of reacting to collisions based on object tags in Unity. This approach of detecting and providing feedback to events based on descriptions in a format similar to VGDL, attempts to answer how to implement support for game feel design *(RQ2)* on a much broader level. Exploring how to support game feel design in Unity prototypes, including ones generated from VGDL descriptions.

This section describes the structure of Squeezer, how it combines ideas explored in relation to VGDL. How interactive descriptions and sequencing of effects lie at the core of the tool. This section also describes the different ways in which the very practical task of integrating Squeezer into game prototypes has been designed. The ways in which description visualisation plays a big role in working with the tool, and how the tool supports the iterative design process. Lastly, the actual detection and execution system is described, explaining how game feel design tools can be implemented without the need for big changes in the game implementation itself.

Figure 4.7: A juice squeezer by UrbanBar

4.2.1 Interactive Description Design

The initial inspiration for Squeezer came from working on UnityVGDL and adding juice effects to VGDL descriptions, as described in the previous section. This work clearly indicated that a general system for taking advantage of a recurring pattern in Unity games and use it to apply juice effects, was possible. Splitting the juice descriptions from VGDL and applying them to a much broader range of Unity projects.

Experienced game development professionals like the author, know that designing and implementing juice effects is a time-consuming task. Juice effects apply to many aspects of games, and each aspect is usually needs to be implemented as a separate system. Combining the most common aspects in a single tool would allow a designer to save a lot of implementation time while designing and testing effects. Squeezer is targeted at the prototyping phase, because creative exploration happens mainly in that phase and because performance requirements are usually lower at that stage[11].

One downside of writing descriptions like VGDL in basic text files, is the limited amount of support a basic text editor provides in the design process. The designer has to make sure the file format and syntax is implemented correctly. Further the designer has to know the ontology by heart, including the parameters of each sprite, interaction and termination type, to avoid looking up details constantly. By making Squeezer GUI based, the juice descriptions could become an interactive design tool. Allowing the designer to select from lists of options, and showing options and parameters for each type of effect in the ontology.

Early on, Squeezer was called "GameFeelDescriptions," because of its roots in description languages. However, the descriptions mainly cover the amplification aspect of game feel described in [155]. This design task is called juicing, which is why the project was given a more fitting name for a juicing tool (see Figure 4.7).

While the underlying data structure will be discussed later in this section, the goal of the GUI was to present an interactive version of the earlier juice descriptions in *VGDL+* (see Figure 4.4). Although the VGDL descriptions lacked support while designing them, the resulting descriptions are very concise and easy to comprehend. Because designing game feel is a complex task involving many different moving parts, keeping the representation as concise as possible is key. However, each effect has multiple parameters that the designer might want to adjust. Showing every parameter for each effect at once counteracts the conciseness of the description. Squeezer avoids that by collapsing each element in the descriptions, hiding their parameters until the designer needs to adjust it (see Figure 4.8 for an example). This combines the power of concise descriptions with the ability to view each parameter and its options, when an element is expanded. The amount of parameters was kept as low as possible to allow as much expressiveness as possible while maintaining the ease of use.

A core part of juicing your game is layering multiple effects on top of each other. The specific game feel arises from adjusting the timing and intensity of each effect. While the intensity is handled through the adjustment of effect parameters, the timing needs to be adjusted relatively between the effects. This is usually handled by setting delays for each effect, from the occurrence of a game event, until the effect is executed. In Squeezer the task of sequencing effects, is handled through nesting and delays. Nesting an effect below another effect in the description (see the indented effects in Figure 4.8), implies the effect will be delayed until the previous effect is complete. Besides the nesting individual effects at the same indentation, can be delayed independently as well. This nesting gives the designer visual representation of the sequence

[11] Lower performance requirements allows greater focus dedicated to expanding functionality instead.

they are building. Additionally, each effect also notes its duration or duration range when randomization is enabled, to assist with sequencing.

In Squeezer reordering and copying effects is handled through dragging and dropping them on other effects. Additionally, the context menu allows explicit copying allowing a designer to copy effects (including their sub-trees) between different descriptions in a project.

Though designing the effect sequences is the main design task. The descriptions like the VGDL descriptions also describe two other aspects. The first is defining which objects should be used for detecting the event to set off the effects. This is done through specifying a tag, a component type or through a list of objects. The tag and component types can be used to dynamically detect if the object is created at runtime. This is useful for game elements that are created as the game is played. The second part of the descriptions define which events trigger the effect sequences. Each description only listens to the defined set of objects, but multiple triggers can be defined for those objects. The effect sequences are usually executed using the object that detected the event as the target (either using the object or its location). However, it is also possible to define a different set of objects as the target using the effect groups.

To recap the descriptions in Squeezer, each description targets a set of detector objects, defines which events to detect, defines the objects to target in case of an event and lastly defines the effect sequences to execute on those targets (see Figure 4.8. The descriptions are kept collapsed in a format similar to VGDL, to make them more comprehensible. Each element in the description can be expanded to display all of its parameters and options.

The effects available in Squeezer were initially based on the analysis [95] of "Juice it or Lose it" [98] and "The Art of Screenshake" [140]. This set of effects has later been expanded with more advanced particle effects as well as effects for additional sequencing options (such as delaying, branching and looping groups of effects). The ontology contains 25 different effects. However, Squeezer uses reflection to look up effects and show them in the GUI, which makes it highly extensible.

4.2.2 Architecture

Getting from conceptual descriptions of effects to executing effect sequences in a game prototype, the underlying system in Squeezer needs to handle updating high numbers of effects simultaneously. From practical experience, the author knew tweening [149] systems are often for implementing juice effects.

The core of Squeezer is based on modern tweening systems, which are used in many games and are a fairly standard component in game engines. The term tween is derived from a task called "in-betweening," where junior artists would draw images between two key-frames created by senior artists. This term was later carried over to computer graphics [165] and animation systems [149]. An example of tweening is scaling an object to twice the size in half a second. Tweens can make changes linearly (fixed step-size per frame), or they can use easing

functions[12] (variable step-sizes, allowing speeding up and slowing down changes over time). Easing functions were popularized in tweening systems by Robert Penner [149]. Tweening systems are often used for creating simple juice effects in games, so they serve as a great foundation for creating a juicing tool. This type of system has a centralized class in charge of controlling and updating active tweens and a set of tweens that manipulate values.

While some tweening systems allow sequencing effects one after another, some juice effect sequences require scheduling effects both sequentially and in parallel at the same time. Consider an explosion: several effects go off initially, such as sound, a shock wave, and a flash of light. Afterward, some smoke and embers might appear, and each of those effects might fade and change color simultaneously. The embers might even flicker a little as they disappear. These effects must be sequenced correctly and applied to the correct targets. These requirements mean the Squeezer core is effectively an extended tweening system for a domain-specific purpose.

There are three basic types of effects in Squeezer: one-shot effects, durational effects, and spawning effects. These three are complemented by a set of sequence control "effects," designed to modify the execution flow of effect sequences. *One-shot effects* are commonly used to play sound effects, activate or destroy objects, and trigger custom game events. *Durational effects* consist primarily of simple tween effects (with the ability to change various aspects of game objects). However, they also have other standard juice effects such as color changes, shake effects (used for both objects and screen shake), and specialized effects like the squash-and-stretch effect, which is used to amplify the impact of events [217]. The *spawning effects* consist primarily of custom particle effects in Squeezer, but they also include a simple copy target effect. The copy target effect can create trails behind objects or emphasize where objects have impacted. Squeezer's particle effects use full game objects instead of usual particle systems that run highly optimized simulations. The benefit of using full game objects is that Squeezer can treat them like any other object it manipulates. This trick means that any effect a designer can apply to a game object with Squeezer can be applied to a particle it creates. Using full game objects is a powerful feature, and it creates much less confusion for novice game designers because they can manipulate particles just like other game objects.[13]

The review of juice effects for *VGDL+* provided a list of both common effects and the events that trigger them. Initially, we built Squeezer to handle only collisions (similar to VGDL interactions). After the review, however, it was clear we needed a few more triggers, for instance: starting the game; creating an object; when an object moves or changes direction; when objects are destroyed or disabled; when the game sends custom events or when a value changes on a game script. Any such event could be amplified using juice effects. To detect these different events, we created trigger scripts attached to objects defined by the descriptions. *Trigger scripts* are scripts that get attached to the defined objects and react to state changes or interactions the object experiences. They leverage Unity's component-based architecture to

[12] See examples of easing functions at https://easings.net/

[13] The fact that particles are not full game objects often confuses novice game designers, especially when particles start flying through other objects, due to simulation limitations.

capture the events and then call the Squeezer effect execution system to execute the attached effect sequence.

4.2.3 Executing the Descriptions

Squeezer does not rely on building effects in Unity sub-systems. Not relying on other systems makes designing with Squeezer fast without requiring expert knowledge about Unity. We wanted Squeezer to be as non-invasive as possible, so it would be easy to add and remove. This requirement means that the descriptions handle adding and removing trigger scripts automatically. The descriptions are the links between the game and the designer user interface (UI). Through the descriptions, the designer defines and attaches triggers and specifies groups of effect sequences (and their targets) and the effect sequences and their parameters. Each description attaches its triggers to a collection of objects, usually defined as objects with a specific tag.

The descriptions have a hierarchy called the **Effect Tree** (seen at the top of Figure 4.8). At the first level (at the top), the triggers are defined, and under them is a list of effect groups. Effect groups are used to control the target of the effect sequence, which is mostly the object that triggered the effect. However, they can also target other game objects, allowing events on one object to trigger effects on other objects. Each effect group contains a single multilayered effect sequence. Effects are indented like in VGDL descriptions, creating a tree structure indicating when effects are executed. Figure 4.8 describes an explosion effect which can be seen in Figure 4.9. The description also shows that it is attaching a single trigger to anything with the tag "ball" (noted next to the **Trigger List** in Figure 4.8). The trigger is a collision trigger, and it reacts to anything (symbolized by a *) when a collision is detected. Beneath that, we see the effect group, which defines the target as "Self," which means the ball itself.

The descriptions are fully serializable to and from JSON.[14] This allows designers to transfer them between projects and game scenes as needed. Though the JSON serialization describes the objects accurately, the descriptions are relatively opaque. Creating a serializer for a custom format like VGDL or the visual sequences in the Squeezer UI would make them more accessible. In the Squeezer UI, the effects can be expanded and collapsed to view and edit details. The expanded view can be seen in step three in Figure 4.10. Adding effects to a sequence can be achieved by pressing the "+" button or right-clicking and selecting add effect. The "+" button and context menus function similarly for the triggers and effect groups. On the left side of each effect, a checkbox controls whether the item is disabled or not, allowing the designer to control which parts of the effect sequence are executed (helpful, for example, when a sequence contains multiple particle effects). The "mutate" button and sequence generator will be explained further in Chapter 5.

[14] For the JavaScript object notation, see https://www.json.org/

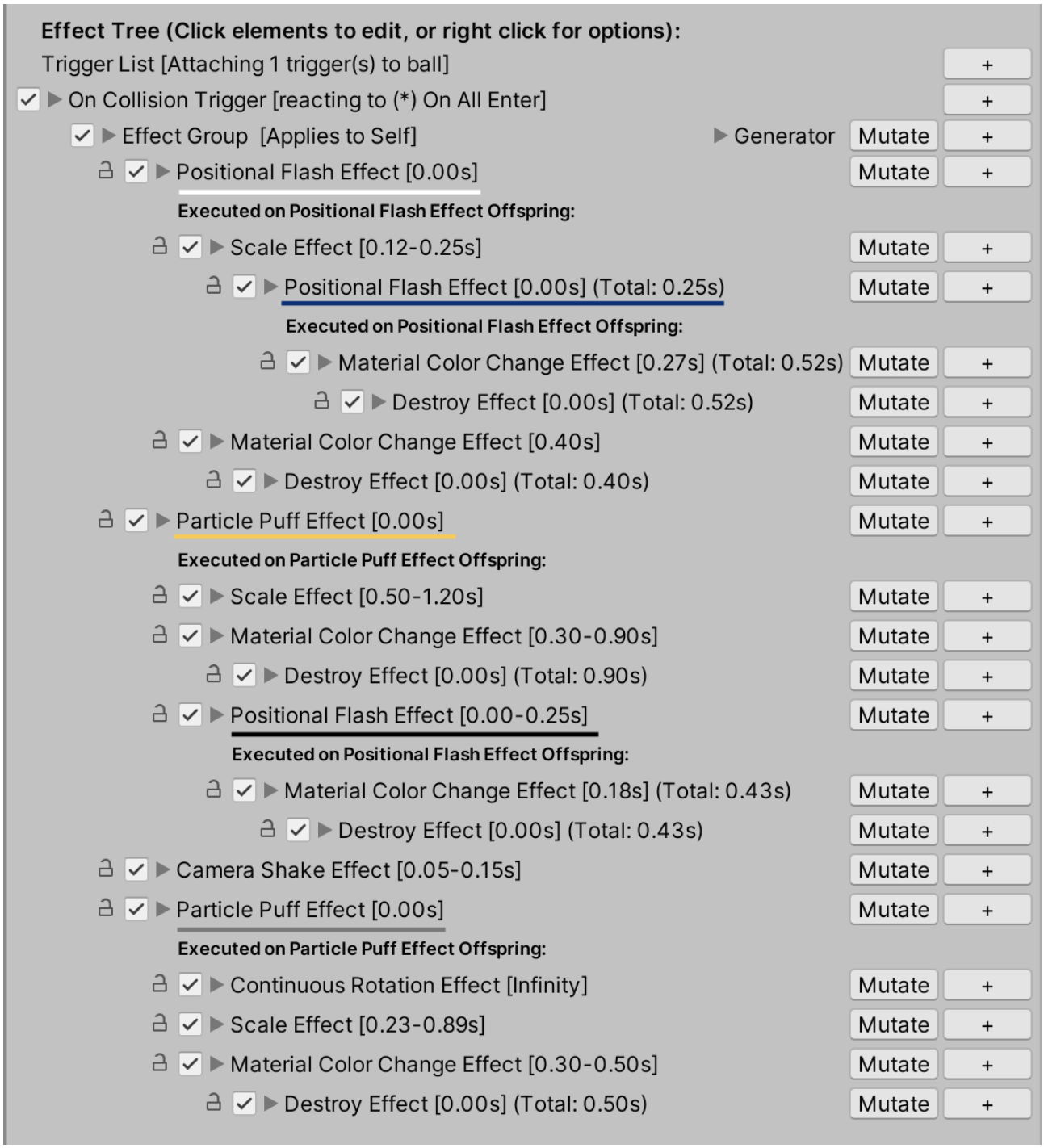


Figure 4.8: The description of an explosion effect sequence. Underlines added externally to associate descriptions and visuals with the animation seen in Figure 4.9.

Figure 4.9: The "synthesized" effect shown at 5fps by executing the description in Figure 4.8. The big white square is the breakout ball (i.e., not part of the effect). There are three visible parts of the effect sequence in the description (the "Camera Shake" is not visible here). The first is the "Positional Flash" (white underline in Figure 4.8). It shows up as a white circle in the first three frames, scaling up and fading out. After the scale-up finishes, a blue flash (blue underline in Figure 4.8) appears and fades out. The second part of the sequence is the "Particle Puff" (yellow underline in Figure 4.8). It controls the slowly expanding yellow circles, each spawning a quick black flash (black underline in Figure 4.8). The combination of particles and flashes creates the illusion of smoky dust as the circles expand and fade away. The third part of the sequence is the "Particle Puff" (gray underline in Figure 4.8). It simulates debris being flung in all directions with small, fast-moving gray boxes.

4.2.4 Designing the Setup Procedure

Squeezer is a tool designed to be dropped into a Unity project. Unfortunately, it requires some amount of setup to get started. However, supporting game feel design also includes supporting the designer while setting up the system. It seems relevant to point out the importance of ease-of-use when implementing tools that support the designer *(RQ2)*. Just as game feel design includes streamlining (see Chapter 3), the task of designing support for the player, building tools requires streamlining the user experience. The benefits of using the tool need to outweigh the perceived friction of use. In Squeezer, the base setup is relatively simple, requiring only the creation of an empty object and adding the description script to it to get started. However, setting the detector objects and defining the triggers can be confusing at first.

In the first iteration of Squeezer [95], an experimental feature called the *step-through mode* was added. Step-through mode is based on ideas from the Klik'n'Play[15] feature of the same name. Initially, step-through mode seemed like a great idea, pausing the game upon any event and asking the designer to design an effect sequence for the event before moving on. Step-through mode was meant to remove the need to manually set up descriptions, allowing the designer to focus on designing the effect sequences. However, designing on the fly does not properly allow iterating the design as game feel design requires. Instead, step-through mode was replaced by a setup wizard, allowing a designer to set up all the potential events they would like to handle, either one by one or all at once. The setup wizard can be used at any point if new objects and events need to be set up (a process that experienced designers can do manually). Figure 4.10 shows the workflow of adding Squeezer into a new game project and beginning the design process.

[15]Klik'n'Play by Clickteam https://knpforschools.webs.com/

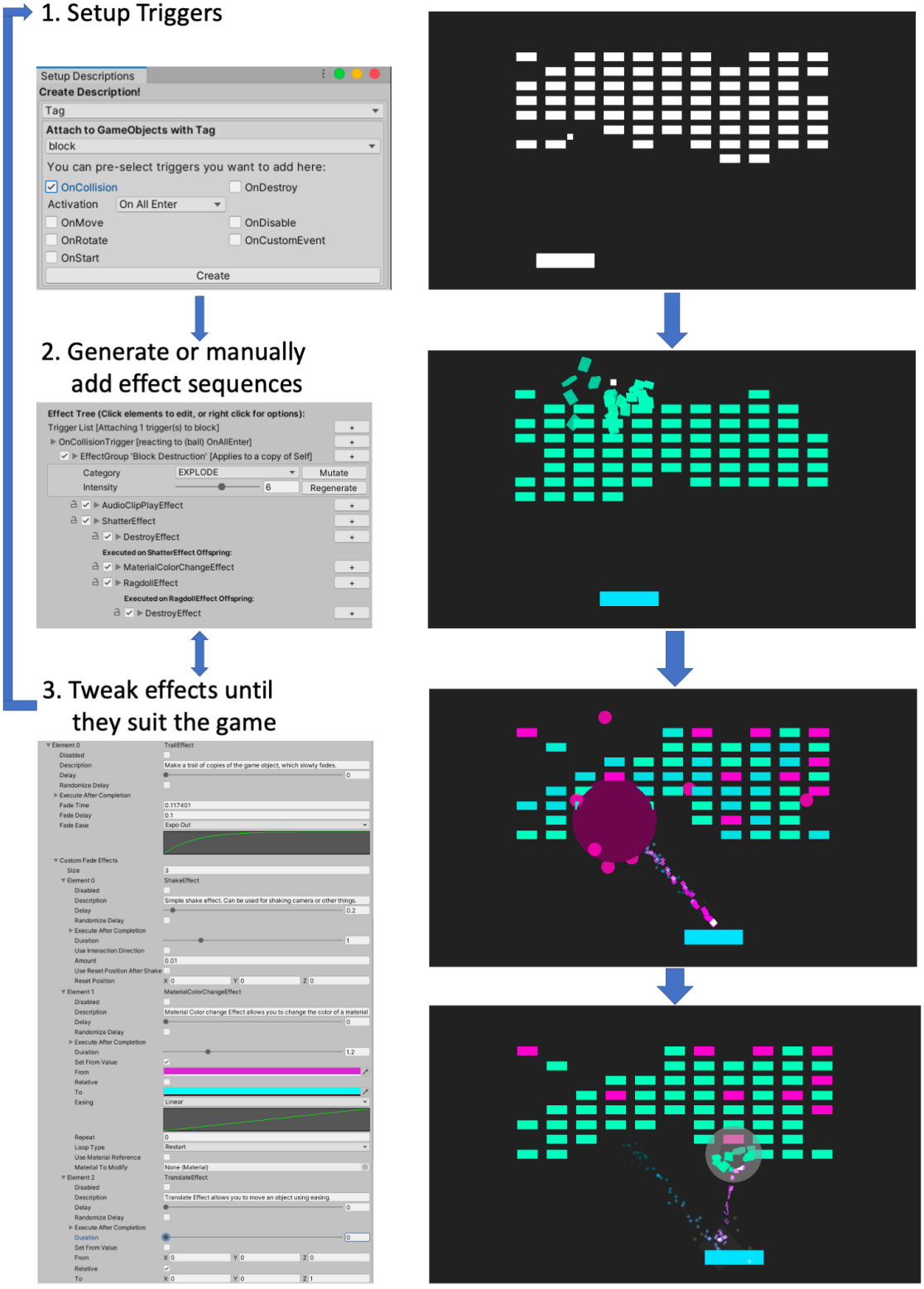


Figure 4.10: Squeezer Workflow. Step (**1**): Take a game prototype and set up the Squeezer event triggers using the setup window. Step (**2**): Add some color and generate or manually add initial effect sequences. Step (**3**): Explore, mutate or design parameters of individual effects until they suit the game mechanics. Repeat steps (**2-3**) until the effect sequence is done, then go back to step (**1**) to add new triggers and start again.

Besides the setup wizard and UI interface, Squeezer provides an API for defining and executing effect sequences through code. This approach allows programmers to integrate, design, and execute effect sequences exactly where they want in code.

4.2.5 Visualising the Effect Sequence

The interactive description system in Squeezer allows designers to change descriptions easily. However, it is an asynchronous task without the ability to view the effect executed in real-time. Initially, Squeezer relied on the game running to bring the descriptions to life. However, relying on the game running creates a cumbersome and semi asynchronous workflow. The workflow required designers to pause the game to alter effect sequences and trigger the same event after resuming play.

Shortening the iteration loop of the game feel design process has been explored throughout the project. Aiming to build tools with reactive and as direct manipulation as possible like Victor calls for [227]. Several experiments and features have been implemented to streamline the design workflow further. One of these features was the step-through mode. Step-through mode functioned as a setup utility and a method for designing on the fly. However, the step-through mode still required the designer to play the game unless they implemented an AI agent to play the game for them. While it allowed the designer to design effect sequences and see them in the game immediately, it can still be problematic to trigger sparse or hard to achieve events. Though designing on the fly reduces the waiting when entering Playmode[16], associated with the usual asynchronous design process, pausing and playing to trigger the events still cause considerable agitation.

Editmode Previews

Editmode previews were developed for the effects in Squeezers descriptions as an alternative to only displaying effects at runtime. Each description can enable or disable previewing the effect sequences. The preview can be displayed directly in the scene (in the context of the targeted object), or it can be executed in a separate preview scene with a copy of the original description target. The preview scene executes the effect sequence related to that object in an otherwise empty scene. When previewing is active, the effect sequence can be triggered manually or executed automatically at an interval of choice. Preview mode is an excellent tool for designing effects, allowing the designer to see any changes they make immediately. It shortens the iteration loop as much as the temporal nature of effect sequences allow (i.e., the duration of the effect sequence). Preview mode removes the asynchronous step of entering Playmode and the need for navigating the game to trigger the event. In addition, preview mode does not suffer from making changes in play mode; a common mistake even seasoned

[16]Playmode is the mode Unity enters as you hit the play button, in contrast, Editmode is when the game is not running.

developers make[17]. Although with Squeezer, it is possible to save changes made in Playmode by enabling it on the description.

Unfortunately, the previews were developed after testing Squeezer with experts. However, preliminary experimentation when building demos and demonstrating Squeezer indicates that the workflow is much easier and faster—allowing the designer to explore a broader range of parameters and effect sequences in the same amount of time.

Editor visualizations

Another feature that supports the designer while using Squeezer is editor visualizations. Editor visualizations show duration, color shifts (a gradient underlining the effect), and particle count next to the names of the effects to mitigate some of that behavior. In the first round of testing, users would open and close effect details to find the effect they were looking for among the collapsed elements. Sometimes simply to check the duration, colors, and other essential parameters.

The editor visualizations make it easier to distinguish the elements of an effect sequence. The visualizations allow users to navigate the collapsed effect sequences more quickly, especially when the same effect is used multiple times in a sequence. A few examples can be seen in Figure 5.3 where the color effects are underlined by the color changes they make to objects. The particle puff effect has a particle count noted in curly brackets, and all effects note their duration in hard brackets.

[17] https://twitter.com/McFunkypants/status/1448363198046818308

4.2.6 Discussion

This section covered how Squeezer is designed to reduce iteration time and improve the juice design process. It describes one possible way of addressing the question of how game feel design can be implemented in existing workflows *(RQ2)*. Specifically how a third-party tool can support game feel design within a game engine like Unity. Chapter 5 will focus on the process of juicing games with Squeezer and cover the results from two sets of expert user tests in [95, 94].

A core part of the experimentation in this chapter related to the reduction of iteration time to increase the efficiency of game feel design exploration. Squeezer reduces iteration time in several significant ways.

1. Squeezer removes the need (while prototyping) for implementing each effect system manually.
2. Squeezer simplifies the event detection and triggering of effects.
3. Squeezer allows designing with concise and comprehensible interactive descriptions, presenting all available parameters and options (initialized with reasonable randomized default values).
4. Squeezer allows real-time design iteration through Editmode previews removes the need for playing the game while evaluating the effect sequence.

Briefly described in the intro of Chapter 5, there are alternative ways to implement juice and game feel within Unity. It is worth mentioning MMFeedbacks [63, 64] in that regard. MMFeedbacks[18] is built on ideas similar to Squeezer, drawing upon similar references and industry experience. MMFeedbacks addresses the question of how to implement support for game feel design *(RQ2)* in a slightly different way than Squeezer. While Squeezer is mostly self-contained, MMFeedbacks relies on several Unity sub-systems to perform various effects. A technical artist developed MMFeedbacks, and while it makes designing and activating juice sequences simple, it relies on expert knowledge of the underlying sub-systems to create many of the actual effects. Squeezer and MMFeedbacks share many basic support elements, but they are differentiated by Squeezers Editmode previews and design support features like sequence generation and interactive evolution. In addition, sequences built-in MMFeedbacks are structured in a single list, with timing adjusted for each effect as needed. The descriptions, including event detection and triggering, also allow Squeezer to be less invasive. The main difference is that MMFeedbacks is designed as a production tool, and Squeezer is designed as a creative prototyping tool.

[18] Recently rebranded into FEEL: https://feel.moremountains.com

4.3 Automated Game Design Systems

The third game creation workflow explored in this chapter is an automated approach, where a piece of software generates new games with little to no interference by human designers. It is interesting to explore how game feel design support can get implemented for an automated system *(RQ2)*. The previous two sections have focused on implementing support for game feel design with human designers in mind. This section explores game feel design support for AI or algorithmic designers.

4.3.1 Automated Game Design

Automated game design (AGD) is a relatively young sub-field of computational creativity and game AI research. AGD research is concerned with building AI systems that either design and develop games independently or take part in a larger game development process. Until recently, AGD research has been focused primarily on rules, goals, and systems, which biases the research towards certain kinds of games, and certain kinds of evaluations, as noted in [46]. Commonly, this results in AI systems capable of generating games, such as ANGELINA [39, 40, 44], the Game-o-Matic [139], Ludii [200], or Gemini [204].

While AGD research stretches back to the early 1990s [148], where the author generated chess variants, the bulk of AGD research has taken place since 2010, with systems like Ludi[19] [22], a system which designed abstract board games. Ludi games were designed to be played with physical game sets.

Around the same time as Ludi was designing board games, videogame-based AGD systems like ANGELINA 1 or the Game-o-Matic were created. ANGELINA 1 focused purely on ruleset generation with little or no consideration for aesthetics. The Game-o-Matic did allow users to apply visual themes based on natural language processing of input prompts connected to a corpus of game art [139]. In all of these cases, the aesthetic responsibility of the AGD system is to choose assets for individual game elements. No attempt is made to emphasize or model any aspect of the game experience through aural or visual feedback effects. One noteworthy exception is ANGELINA 3 system did venture into the area of aesthetics [44] by dynamically finding visual assets and triggering sound effects in accordance with specific game actions.

There are AGD systems that produce games with excessive amounts of positive player feedback (also known as juice [104], see also Chapter 3). However, these rely almost exclusively on the system designers adding juice to the AGD system's templates. *Variations Forever* is the best example of this, a juicy and engaging game that redesigns its rules using constraint-based programming [191]. In the AGD system, Smith and Mateas added juice to the templates, so even if the AGD system could not control it, the resulting games benefit from it.

As part of this Ph.D. project, I enjoyed a virtual stay abroad[20] collaborating with Mike Cook at the Queen Mary University of London (QMUL) – Game AI research group. Cook is

[19] Ludi is the predecessor to the Ludii game system [200].

[20] Due to the Covid-19 pandemic, travel and attendance at the university were, unfortunately, not possible.

a leading researcher within automated game design (AGD) [43]. This collaboration presented an opportunity to address how game feel design support can get implemented *(RQ2)* in the context of automated game design.

The ANGELINA [39, 40] AGD systems created by Cook et al. have touched lightly upon the aesthetics of the generated games [46]. However, most AGD systems completely ignore designing how the games feel, with notable exceptions like Variations Forever [191]. In an attempt to rectify this lack of game feel focus in AGD research, our collaboration resulted in combining Squeezer with an AGD system and generating juice effects as part of the automated game design process [93].

4.3.2 AGD Approach

The AGD used in the project [93] is an unreleased system called *Puck*. Puck is the spiritual successor to ANGELINA [39, 40] and similar in nature to the introductory *Bluecap* [21] that was designed as part of an AGD tutorial[22] at the 2020 Conference on Games[23]. Puck is built using principles Cook presented earlier about software engineering for automated game design [42]. In this paper, Cook investigated how to engineer AGD systems in a way that would allow the system to take part in a game development process alongside human developers[24]. This approach pulls AGD in a different direction than most PCG systems, where the focus is to complete tasks as fast as possible. By considering the AGD system a collaborator in the game design process, the system is encouraged to spend as much time as it needs to evaluate and propose changes to the game. The implications for an AGD system are that it has to be split into various phases. Each phase can be revisited at different stages of the development process as many times as is needed.

In Puck, that functionality is implemented as design modes, each with a different purpose in the process, such as generating rules, selecting assets, evaluating gameplay, and designing look and feel. These modes each have different tasks and additional sub-goals. The modes follow a progression, but should the gameplay evaluation meet some obstacles; the system could go back and change or regenerate rules, which in turn requires the reevaluation of every subsequent task. The system is currently run in different modes that are triggered manually[25]. Each mode will result in the generation, alteration, or addition of files in a game folder. Using this approach, one or more instances of Puck and human developers can work on the game files simultaneously.

Like many other AGD systems, Puck relies on specifying its games in a game description language. As discussed in Section 4.1, there are two main options for adding game feel design

[21] https://github.com/possibilityspace/bluecap

[22] https://www.youtube.com/watch?v=dZv-vRrnHDA

[23] https://ieee-cog.org/2020/

[24] Human designers can alter the persisted game files when Puck is not working on them. The fact that Puck persists games through files means that with a few changes, it could become a collaborative co-creative mixed-initiative system [233].

[25] This is one of the reasons it has yet to be released.

support when using a game description language. The first is to extend the description language with the design elements needed and extend the generator or interpreter to handle the design elements. The second option is to use a separate tool like Squeezer for designing, saving, and executing the effects. In this experiment, the latter approach was used.

4.3.3 Squeezer Integration

A new mode called "designing juice effects" was added to Puck for integrating Squeezer. In this mode, Puck uses Squeezer to generate a set of effects based on the game events used by the game it is designing. While this initial integration is relatively simple, the mode is designed to interpret data from play logs. Puck can record logs using observations from the game evaluations it performs. These observation logs can then be analyzed in other modes to adjust game balancing, generate heat maps, or count events.

To accommodate the needs in Pucks juicing mode, the Squeezer API needed a few tweaks for saving and loading effect sequences in a third-party setup. Puck uses this to save the effect descriptions alongside the game descriptions, tying each file to a specific event in the game descriptions. Then when playing the game, Puck loads effect descriptions and triggers them through Squeezers API for queuing and executing effects.

Like a human designer would need to adapt their workflow to using new tools, an automated designer also needs to adapt. When running the games, Puck initially visualized them purely by rendering the state of the game after each player interaction occurred. However, this type of visualization only barely works for the turn-based games Puck currently designs. Seeing where the pieces end up after each turn obscures the game mechanics and quickly becomes confusing, with complex rules in play. To avoid this and allow visual effects (and sounds) to be meaningfully executed, the visualizer in Puck was changed. Instead of executing a complete turn and then showing the final state, the new visualizer is driven by the same game events that make up the game descriptions. This event-driven visualization allows intermediate steps to be shown, and using either effect descriptions from Squeezer or basic tween effects; each step can be animated.

4.3.4 Juicy Challenges

The main challenge of designing game feel in an AGD system is evaluating the result. Unlike game balance, rules, and objectives, evaluating how a game feels is an open question. Knowing this evaluation task would be needed in an AGD implementation, Cook assisted with a series of unpublished experiments for effect sequence analysis and comparison. Cook has previously used expressive range analysis (ERA) [202, 116] in *Danesh* [45] to analyse the output of PCG systems. However, the preliminary results from using ERA and a few other techniques did not even yield good results for comparing different sets of effects, let alone evaluating the feel. While evaluation remains an open task, it presents an exciting new challenge for AGD research.

The benefit of the design mode approach used to build Puck [42] is that it can work as a

collaborative system, where humans can participate in the process (a sort of human-machine collaborative game design). This approach means Puck could mark games ready for human evaluation and wait for a human collaborator to finish the task before working on the game again. Interestingly this process resembles a production line, where despite a considerable focus on automation, humans remain in parts of the process.

4.3.5 Discussion

This section describes the integration of Squeezer into an automated game design process. Presenting one possible way that game feel design support can get implemented *(RQ2)* in an AGD system. Showing different potential options for integrating game feel design in AGDs, as well as the implementation method used in this project [93]. In the project, Puck delegates the responsibility for game feel design to Squeezer while remaining in control of the overarching AGD process. The approach is made possible by the Squeezer API for saving, loading, and executing effect descriptions. Additionally, the task of designing the effect sequences is handled by Squeezer and is made possible by the effect sequence generator, discussed in Chapter 5.

4.4 Discussion

This chapter described different approaches and prototypes created in an attempt to answer: *"how can game feel design support get implemented in existing workflows?" (RQ2)* For game description languages like VGDL, there are multiple ways to add game feel, depending on the outcome or goal. Suppose the goal is creating a prototyping or game creation tool that supports game feel design. The descriptions would need to be altered to accommodate building effect sequences. The underlying game system needs to execute durational effects, even if they are only activated based on collision or timer events. While VGDL could potentially serve as a platform for prototyping games, the entire ontology would have to be re-written to accommodate human players better (streamlining the support). An alternative is making a game description interpreter that translates a GDL game description to an engine-specific prototype. Examples of such an interpreter are the student project extending UnityVGDL and the Puzzlescript importer for Unity used by Alan Hazelden. An interpreter like this does not directly add game feel support. However, the interpreter is free to alter the feel of any elements it generates to make the elements more juicy, streamlined, or provide more tweakable parameters to the generated elements. The real power of generating games in this manner comes from leveraging the entire game engine tool ecosystem. These ecosystems are built around engines like Unreal or Unity to support developers with different aspects of game making. Additionally, tools like Squeezer can add juice effects to the generated games.

That leads us to the second workflow using Squeezer. Squeezer combines the power of concise descriptions like VGDL with common patterns for detecting events in game prototypes. Building on this event-driven foundation, Squeezer can be attached to most Unity prototypes, allowing designers to explore, generate and design game feel by amplifying events with juicy effects. The next chapter will dive into more details on the impact of tools that support the game feel design process and designing game feel with Squeezer. While Squeezer is a tool that can be added to Unity, this chapter also looked at how the support is integrated into a project workflow *(RQ2)*. This chapter explored two semi-automated approaches to setting up Squeezer, which was later discarded called step-through mode. The other is more akin to a setup wizard used in software installations.

Step-through mode was the first step in a series of experiments to shorten the iteration loop while designing the effects, attempting to "Stop drawing dead fish" [227] and moving the design loop closer to real-time iteration. Using the serialization system built for saving changes in step-through mode, the second attempt at shortening this loop was to save any changes made to the effect sequences at run time. While this approach worked, the designer would either have to control the game while designing or create an AI to play for them. Moreover, even with an AI playing the game, sometimes the events the designer is designing for only happen sporadically. It is still possible to design while playing by setting the description to save any changes made in Playmode.

Preview mode, the newest experiment in shortening the iteration loop, allows the designer to execute the effect without starting the game. Preview mode allows the designer to execute

their effect sequence in the current scene or a separate preview scene. The effects can be executed either on-demand or triggered automatically at a specified interval. Unfortunately, this feature was made after our last user test, but the author has used it to design several Squeezer presentations. While this is a biased statement, the lower iteration time and immediate preview make designing effect sequences more fun and way faster. It removes the delay from playing the game until the event occurs, making design iterations very quick. Condensed interactive descriptions, visualizations of specific parameters, and preview modes support the iterative design task. Nesting effects and interaction models like drag-and-drop streamlines the sequencing process. Lowering friction by providing support to the developer removes barriers the user does not care about, similarly to how streamlining the game feel can smooth out the player experience.

The last workflow explored in this thesis is automated game design (AGD), specifically the unpublished framework "Puck". Puck uses its system to execute and render the games instead of generating a fully playable game prototype in the game engine (Unity in this case). Puck was extended in order to answer the question of how to implement game feel design *(RQ2)* into the AGD process and how adding juice would change the perception of games generated. Chapter 6 discusses the actual implications of adding juice to the prototypes and the results from the playtest performed [93]. This chapter discussed how the process had to be altered, concerning how it was executing and rendering the generated games.

Answering how game feel design support can be implemented *(RQ2)* in ADG systems happen to overlap heavily with the suggested solutions to implementing game feel in GDLs. One way is for the execution engine and game descriptions to allow the descriptions and execution of game feel elements directly. Another way is generating a game prototype in an engine, similar to the VGDL and Puzzlescript approaches described earlier. The approach described in this chapter is a middle ground between the other two. Puck uses Squeezer to generate and save effect sequences tied to certain predefined events, and upon game execution, it will ask Squeezer to execute the saved effect sequences. In a sense, this represents one answer to how game feel support can be implemented *(RQ2)* by showing that AGD processes can delegate responsibility for both describing and executing game feel elements to a tool like Squeezer. Generating games with the support of a third-party tool like Squeezer instead of implementing game feel directly in the system itself. In [93] we only allowed Puck to use the generator for generating effect sequences and applying them to predefined events in the system. However, this approach could just as easily allow an AGD system to design effect sequences through Squeezer's API, as well as define the events that trigger the effects. Allowing an AGD like Puck to define the events that trigger effects would make it capable of reacting correctly to more complex events beyond spawning, destroying, and moving game pieces.

To summarize the game feel design implementation approaches discussed in this chapter. There are two main options for game description frameworks like VGDL; one is to extend the framework to allow game feel design and polishing directly in the descriptions. The second one is to use the description-based system as a stepping stone by generating game prototypes in an

external game engine like Unity, which leads to the workflow of using a game engine to build a game. In the next chapter, we will return to the many different approaches in this workflow. The implementation shown in this chapter explores how to implement a custom tool (Squeezer) with support for game feel design and how that tool fits into a typical game development workflow. Extending ideas from description languages like VGDL, the Squeezer implementation shows one way of supporting game feel design by allowing the designer to see live previews of effect sequences while manipulating interactive descriptions. Lastly, the implementation of game feel design support in an automated game design workflow shows how a hybrid approach can be used by leveraging Squeezer. It is shown that AGD systems can potentially rely on third-party tools to implement the game feel design directly. An AGD system can leverage Squeezer's descriptions and sequence generation (see Section 5.3) for designing juice effects, and at runtime, Squeezer can execute the effects based on specific game events.

Chapter 5

A New Game Design Process

A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.[1]

Related Papers
Johansen, Pichlmair, and Risi, *Squeezer - A tool for designing juicy effects*
Johansen, Pichlmair, and Risi, *Squeezer - A mixed-initiative tool for designing juice effects*
Johansen and Cook, *Challenges in generating juice effects for automatically designed games*

This chapter is focused on the experience of working with design tools (mainly Squeezer) for designing game feel. The larger question explored in this chapter is *"what is the impact of design tools on the game feel design process?" (RQ3)*

To answer this question, we first need to explain the most common approach to game feel design. Drawing on the authors experience both from teaching and developing games, descriptions by practitioners [195, 98, 140, 18, 57]. As well as the data from the Squeezer user test surveys, the most common approaches are:

- **Inexperienced designer approaches:** Make rules, levels, and assets (often not including sounds and music). Then once everything is working, attempt to make it "fun" by adding event significations through common juice effects like particles, screenshake, and sound effects. *This approach often suffers from various elements feeling flat and uninteresting. Additionally, often utilizing systems and built-in tools without the required experience, resulting in generic feel and effects*[2].

  The "Unity"[3] **approach:** Use Unity and the most standard effect tools (e.g. the particle system, *Cinemachine*, built-in scripts, *VFXGraphs*, post-processing effects,

[1] From "Mostly Harmless" by Douglas Adams, 1992
[2] It is common for developers to be able to tell which generic tools have been used to create games by other developers. Less experienced developers often end up making more generic-looking games.
[3] Note that this is not unique to Unity and applies to many other game engines as well.

Shader Graph). Unity's particle system, post-processing effects, and built-in scripts often deliver a relatively generic game experience without custom assets and lots of tuning. More experienced developers know how to break out of the generic look and feel, either by manipulating the tools or using more advanced tools from the Asset Store. However, games created with Asset Store tools can also lose their uniqueness as they gain popularity and become the defacto standard. Popularity like this usually happens when a tool has a very narrow focus or the default settings (if static) are "good enough" for many different game types. The same is true for built-in systems like the post-processing effects, VFXGraphs, and the simple default control scripts available in Unity.

The "find it on the internet" approach: Find various elements or tutorials created by experts online. This approach is often used by developers when they need to learn how to do something or get going quickly. The issue is that the reasoning behind the implementations might not be clearly described, which can make them hard to adjust with good results. Like popular tools, games developed by following popular guides and using popular game assets can also result in generic feeling games. However, sometimes the source code for commercial games is released for others to dissect and learn from. An example is the release of the source code for games such as Doom or Quake[4] or the Celeste character controller [5]. These resources can lead to design discussions and new solutions developed by the game development community.

- **Experienced designer approaches:** Make your game step by step. Add the missing components when certain aspects feel flat or are missing support, amplification, or physicality. Mostly achieved by implementing small tricks that the designer has learned improves the game experience. *This approach often requires custom code or homemade tools that fit the style of the game, as well as a lot of game design experience.*

  The "Tweening" approach: Many developers use tweening systems [165, 119] to achieve a large portion of simple animations and juice effects. This approach provides the designer with a highly optimized system for executing all sorts of parameter changes over time. It has a wide range of applications from animation, over time, manipulation to delaying game events or executing side-effects in several stages.

  The "Visual Tools" approach: Some developers will use visual sequencing tools like the *Timeline* in Unity or MMFeedbacks [63, 64]. This approach can trigger game events, manipulate exposed object parameters and execute animations, similarly to the tweening system approach. The difference is that the design task includes a visualized sequence of actions (or effects). However, this approach still relies on expert knowledge with the underlying systems like the particle system, *VFXGraphs* or *Cinemachine*.

  The "Do It Yourself" (DIY) approach: Very experienced developers sometimes

[4] https://github.com/id-Software

[5] https://github.com/NoelFB/Celeste also discussed on Game Maker's Toolkit: https://www.youtube.com/watch?v=yorTG9at90g

prefer custom code implementations of features to tailor the experience precisely to the game they are making. This approach is also the most common when developers develop proprietary game engines, though parts of the implemented systems often have similar characteristics to tweening systems.

The rest of this chapter is divided into five sections. First, the design intent behind Squeezer is explained. Squeezer is intended as a prototyping tool, allowing designers to playfully explore the possibilities when adding audio-visual effects to their prototypes. Second, the impact of visual sequencing and design without the need to implement sub-systems is analyzed through the results of the first user study of Squeezer in [95]. Third, the question *"how can effect sequences be generated through algorithmic means?"* and *"how does sequence generation impact the design process?"* is examined through results of the second user test of Squeezer in [94] and the author's experiences. Fourth, the impact of adding mixed-initiative exploration is also examined through the second user test [94]. The user test results and the author's personal experience indicate the impact tools can have on the design process. Lastly, the chapter is concluded by discussing the implications of introducing a tool like Squeezer. In terms of altering the design process for automated game design (AGD) systems and prototyping game feel. As well as how an introduction and focus on the importance of game feel design in an academic context could lead to better collaboration, understanding, and insight for researchers and practitioners.

5.1 Designing for Playful Exploration and Prototyping

The Squeezer user interface (UI) draws inspiration from multiple sources to build an interface suited for playful exploration and prototyping. The following section will dive deeper into each area. However, it is worth noting the influences for this discussion and how the Squeezer UI attempts to inspire playful design exploration as a prototyping tool. Squeezer benefits from having a playful UI, allowing users the immediate pleasure of exploring and designing visual effects in real-time. As described in Section 4.2, the core representation of effect sequences is the interactive visual sequences. The visual sequence designer is heavily inspired by VGDL descriptions or rather the language modification experiments to support game feel design (see Figure 4.4). Real-time previews are inspired by projects like Doodle Studio 95! [161] and "Stop Drawing Dead Fish" [227]. The effect sequence generator is inspired by SFXR [154], expanding the idea of sound effect generation from categories into larger effect sequences. Lastly, while the interactive evolution interface is not based on any particular system, the interface displays various artifacts the user can select. Other interactive evolution interfaces like Picbreeder [179] present an overview of the artifacts to choose from. However, due to the temporal nature of effect sequences (and potentially overlapping sound effects), the interactive evolution interface in Squeezer only shows a single set of effects at a time.

The primary objective of Squeezer is to remove the need (while prototyping) for building the inner workings of an audio-visual feedback system and providing the designer with the

ability to playfully explore the internal complexity of mixing multiple sub-systems in order to gain creative insights into the resulting audio-visual feedback.

In order to foster multiple types of creativity, [16] and lateral thinking [53] several strategies were employed and iterated on while building the Squeezer UI. The VGDL style descriptions and effects ontology form a set of constraints for the design exploration. Constraints can improve creative thinking [1] and allow clever combinatorial ideas [16] to arise. Additionally, the Squeezer UI is inspired by the idea of immediate feedback[6], both emphasized by Victor [227] and noted by Hobye [85] concerning design and playful exploration interfaces. Squeezer employs several strategies outlined by Hobye in "Designing For Homo Explorens: Open Social Play in Performative Frames" [85] to build a playful design interface. Hobye argues that the playful nature of humans [87, 183] can be leveraged when designing interfaces for design exploration. The ideas tie in well with Boden's exploratory creativity [16] and the idea of prototyping. While the ideas that originate from this type of design exploration might only be p-creative [13], and not necessarily h-creative. The intended outcome of the effect sequence prototyping process is finding the best solution for the "feeling" the designer is attempting to juice [18].

Hobye [85] outlines six design insights for enhancing playful exploration through internal complexity. Many of these insights align very well with Victor's interface ideas [227, 225] and Boden's types of creativity [16]. Below each design insight is listed, along with an explanation of how Squeezer accomplishes it.

Insight 1: *Create an interaction space to explore through nonlinear algorithms*
Squeezer's main interface, the interactive descriptions, functions as the primary interaction space for user interactions. Building sequences of complex behaviors that can interact and influence the underlying objects in nonlinear ways is one of the core strengths of Squeezers interface. Squeezer also allows the designer to create nonlinear designs through the nested effect sequences. The tree structure makes it possible to generate particles that explode in more particles when they disappear or are randomly assigned different sub-branches during execution (allowing multiple colors of particles or different variations of effects).

Insight 2: *Create a 'sweet spot' between predictability and chaos*
Beyond the complex and sometimes unpredictable interactions that the interactive descriptions allow, sequence generation in Squeezer introduces chaotic variations. Squeezer generates sequences by mutating an effect sequence recipe (see Section 5.3). The mutations are severe enough to create variation and interesting combinations but light enough to remain relatively recognizable within certain categories. The variation allows Squeezer to "surprise" the designers with uncommon combinations that are akin to Boden's combinatorial creativity [16]. The creative combinations, in turn, may lead to transformational ideas in the designer, as they evaluate the effects within the context of their game events. A sort of inspirational chaos, partially provided by Squeezer as it executes designed or generated effect sequences in different game contexts.

[6]Though the effect previews have gone through several design iterations as noted in Section 4.2.

Insight 3: *Create a multilayered interaction space while keeping tight coupling*
The description UI in Squeezer creates an overview of the interaction and the audio-visual feedback created for that interaction. The interface provides a clear overview of the effect sequences with only the most critical information shown. In the literal sense, the effect sequences in Squeezer are multilayered by design, and this is used as delaying effects flexibly until the end of another effect. However, in a true sense, the details of each effect can be found by expanding individual elements of the sequence and exposing an additional layer of adjustable information. Besides the effects themselves, the descriptions in Squeezer also describe which game objects will trigger the sequences and the specific conditions that trigger those sequences.

Insight 4: *Create multiple interactive modes*
Squeezer has multiple interactive modes ranging from coding API and visual sequence design to sequence generation and interactive evolution. They each have different affordances [142], and designers can use them separately or in combination, depending on their purpose (e.g., learning the tool or exploring the design space).

Insight 5: *Create interfaces that guide the interaction*
As a domain-specific tool, Squeezer is designed for assisting designers in the process of getting juice effects into their games. The interactive evolution interface or the sequence generation assists the designer with exploring the effect design space in different ways. Sequence generation can be used to get something showing almost immediately, dropping the designer straight into a design iteration loop. On top of that, the interactive sequence design is based on an ontology of carefully curated effects. Each effect is annotated with short descriptions and tooltips to explain how the interface functions and how each parameter alters the effect. The setup wizard helps set up the descriptions to trigger under the right conditions. Likewise, every parameter in the effects will be randomly set within reasonable ranges, though most parameters can be adjusted outside these ranges if the designer wants to.

Insight 6: *Create exploration with rich real-time feedback*
Squeezers preview mode allows generating or designing an effect sequence while previewing it in real-time. Providing real-time feedback is a key point in Victor's talks [227, 225]. Real-time visual feedback enables transformational creativity [16] and lateral thinking [53] by allowing designers to test unusual combinations and ideas that might affect gameplay in unexpected ways. While the exploration of the design space through combinations of known effects enables the other two types of creativity Boden describes [16].

5.2 Visual Sequence Design

As described in the previous chapter, Squeezer is designed around the idea of having interactive descriptions. The interactive descriptions empower the designer by combining concise and comprehensible descriptions with the power of layering information and guiding the design process (see Figure 4.8). Squeezer further removes the need to implement these systems manually by providing the system to execute the descriptions based on the specified interactions. Moving the designers focus away from writing lines of code and keep it on the task of designing visual aspects, thinking about layering effects and how that impacts the experience. This shift allows designers to focus on iteratively testing ideas, making changes in intuitive ways, and seeing the outcome in real-time. Thus a designer using Squeezer can establish the broad strokes of the abstract game feel communication between player and game. By sketching the audio-visual feedback to communicate the game's rules, intent, and actions.

Squeezer is a domain-specific tool focused mainly on the essential aspects when designing audio-visual feedback. Though some effects could arguably be used for other types of event handling in Unity, and the tweening system built into Squeezer could be used for many purposes in games, the domain requirements limit their general flexibility. Selecting how effects are implemented, which parameters to expose, and their range of options determine how expressive the effects are and how comprehensible they appear. Building common design patterns into effects and how they are presented eases the learning and design process.

For instance, slight alterations (pitch shifting) are a common trick in sound effects to make footstep sounds appear more organic. A similar approach can be applied by mutating or randomizing specific parameters of effect sequences. Timing variations is a common trick to make effect sequences seem more natural. In Squeezer, effects can be randomly delayed, and durational effects can be executed with a random duration (set through effect parameters).

5.2.1 User test

Evaluating how well Squeezer accomplishes the task of visual effect sequencing and the potential impact it might have. A qualitative user study was conducted to validate functionality and discover flaws lurking in the design. The test was conducted with four users with between four and fourteen years of experience with game development. The initial findings were published as a work in progress at CHI:PLAY 2020 [95] The article discusses how participants found the tool interesting and fun to work with, some of the issues discovered during the test, and how they envisioned using a tool like Squeezer.

The user test was set up as an open-ended exploration of the tool. After introducing the tool, the users were asked to add whichever effects they saw fit to a simple Breakout clone. They were allowed to ask questions about the interface and clarifications regarding effect names[7]. The users would either use step-through mode or manually add effects in the editor and then

[7]At this point, the UI was still very rough, and it lacked many usability features such as tooltips and drag-and-drop.

play the game to view their designs in action. After working with Squeezer for 30-60 minutes, the users filled out a survey about their experience.

In their feedback, they found the tool novel and engaging, and they noted that, with an improved UI, there would be various uses for it. These included teaching and game jamming. They also noted how much easier it was to add effects that would have otherwise taken them a significant amount of time to implement. The survey asked participants about their general experience with game development, their experience adding audio-visual effects to games, how they would usually go about adding effects into their games, and questions related to their experience working with Squeezer. In [95] the answers to two of the survey questions are emphasized, and the rest is quickly summarized for brevity. For the question **"Would you like to use this type of tool for prototyping in the future? If not, what would we have to change to make the tool useful for you?"** all four participants said yes, and three specifically mentioned using it for game jams. Additionally, one participant who also teaches game design mentioned *"[...] I would also definitely give this to my students when talking about game feel and juice. I think letting them play with these effects would be a nice, time-efficient way of getting to experiment with juice and exploring how it changes game feel."*. Another participant said *"[...] I could also see it being useful as a communication tool on teams, using the tool to quickly demonstrate various intents."*. These participants are game development experts, and they both describe uses for Squeezer (beyond game jams) as a visual and interactive aid in their work. The answers are interesting regarding the impact design tools that support the game feel design can have on the process itself *(RQ3)*. Communicating ideas to others effectively is a big part of both game development and teaching. Saving time when conveying points by sketching out intentions with a tool like Squeezer would undoubtedly alter the process itself for collaborative groups.

As a side note, only one of the four participants in the test used step-through mode; they found it a bit confusing to understand the context when the game paused itself, and they, unfortunately, overlooked the randomly generated "recommendations" at the bottom of the screen. While not directly related to the visual sequencing, these participants did not see any potential value in using step-through mode, and the one person that used it did not find it helpful. Additionally, step-through mode did not facilitate design iteration led to the decision to remove it in favor of a more straightforward editor-based setup. This result exemplifies how seemingly good ideas can appear more valuable and impact the design process less than expected. This impact is significant to note concerning how design tools can impact the game feel design process *(RQ3)*.

When asked about features they felt were missing during the test, one participant mentioned reordering the effects in the list. It was technically possible to remove and add effects from the list to accomplish the reordering at the time. However, to ease this process, a drag-and-drop system was added and the ability to copy and paste effects between different effect sequences. The reordering does not change the actual execution of the effect sequence unless an effect is moved to a different layer (attached to be executed when another effect completes). Another

participant mentioned the need to modify particle effects more accurately based on parameters from the interaction. During the first test, only one particle effect was available, namely the shatter effect, which creates several copies of the object (or a list of prefabs) and applies a force based on the interaction (in this case, a collision). To accommodate this concern, Squeezer has since gotten two additional "explosion" effects that allow more control over the particles. Additionally, the parameters passed from interactions to the effects have been changed, allowing designers to further control parameters based on the interaction parameters.

The participants also noted which features they found most useful. One participant noted that *"Screen shake, shatter, trail, and color changes were easy and powerful to apply. It felt like it would save me a significant amount of work if I were prototyping, and, e.g., at a game jam, this would be useful to throw in some nice effects quickly. [...]"*. Another is that the effects that spawned other objects could become very powerful with a few more parameters. One participant answered *"The sound effect, it just added life."* As the sound effects in Squeezer uses a version of SFXR [154] adapted to Unity, it only confirms that SFXR is an excellent tool for adding sound effects. However, summing up those statements, they cover many of the effects available with Squeezer. Combined with the answers about missing features, it suggests that Squeezer's effects seem to cover the design needs for audio-visual effects adequately.

As part of the survey, the participants needed to evaluate the descriptions and the individual elements they are made of. While some participants found it straightforward enough to understand the hierarchy and elements of a description, others found they were hard to read. Certain elements blended together, making it confusing to understand at first for some participants. The UI concerns and issues in both the first and the second user test have been iterated upon for clarity and better separation of the elements that make up a description: the interaction targets, the interaction triggers, the effect groups (where to apply the effects) and the effect sequences themselves.

When asked how they would usually add effects to games, all four participants mentioned writing code (or reusing old scripts). Some also use animation tools like the animator in Unity or tweening libraries. While writing or reusing small scripts for this task is how most experienced game developers design audio-visual effects in their games, the know-how is based on experience and largely hidden to novice game developers. With tools that present common effects and their parameters like Squeezer, not only does this information become visible to novice designers, but it becomes an easy way for experienced designers to quickly design something without needing to reacquaint themselves with old code. Additionally, the visual presentation of effect sequences allows designers to focus on and experiment with the design instead of making sure the code works or looking up how to implement something. This more straightforward workflow ties back to their answers related to whether they would like to use a tool like Squeezer. From the responses, it seems Squeezer would be useful in a prototyping context (like a game jam) when exploring the effect design for a game. However, it could also help teach novice designers about audio-visual effects or improve communication between different team members to convey specific intents and ideas.

Figure 5.1: The Moog modular synthesizer "system 55".

Revisiting the larger question in this chapter *"what is the impact of design tools on the game feel design process?" (RQ3)* The Squeezer user testing suggests that the impact of visual sequence design shifts the main focus away from "how to implement and orchestrate the effects" towards the design problem of "composing the right effect sequence". The shift allows designers to focus on the communicated feeling. As Lisa Brown says, "you're not juicing your game – you're actually picking a feeling that your game should communicate and juicing that feeling" [18]. Further, providing a tool built for the game feel design process, that process becomes more transparent to novice designers, making it useful for teaching.

5.3 Sequence Generation

The initial version of Squeezer gave designers a tool for juice design. A visual effect sequencing tool with many different effects to choose from and combine to produce the desired game feel. As noted in [95] Squeezer can be viewed as a juice effect synthesizer. Squeezer is similar to modular synthesizers (see Figure 5.1[8]), where modules can be connected in many different ways to generate sound. Each module in these synthesizers modifies the electric signal in some way, just like effects in Squeezer modifies the final audio-visual result in some way.

The main research question of this thesis asks *how can* ***design tools*** *support* ***game feel design****?* Following the line of investigation related to providing more support, this section

[8]See also https://www.moogmusic.com/news/return-moog-modular

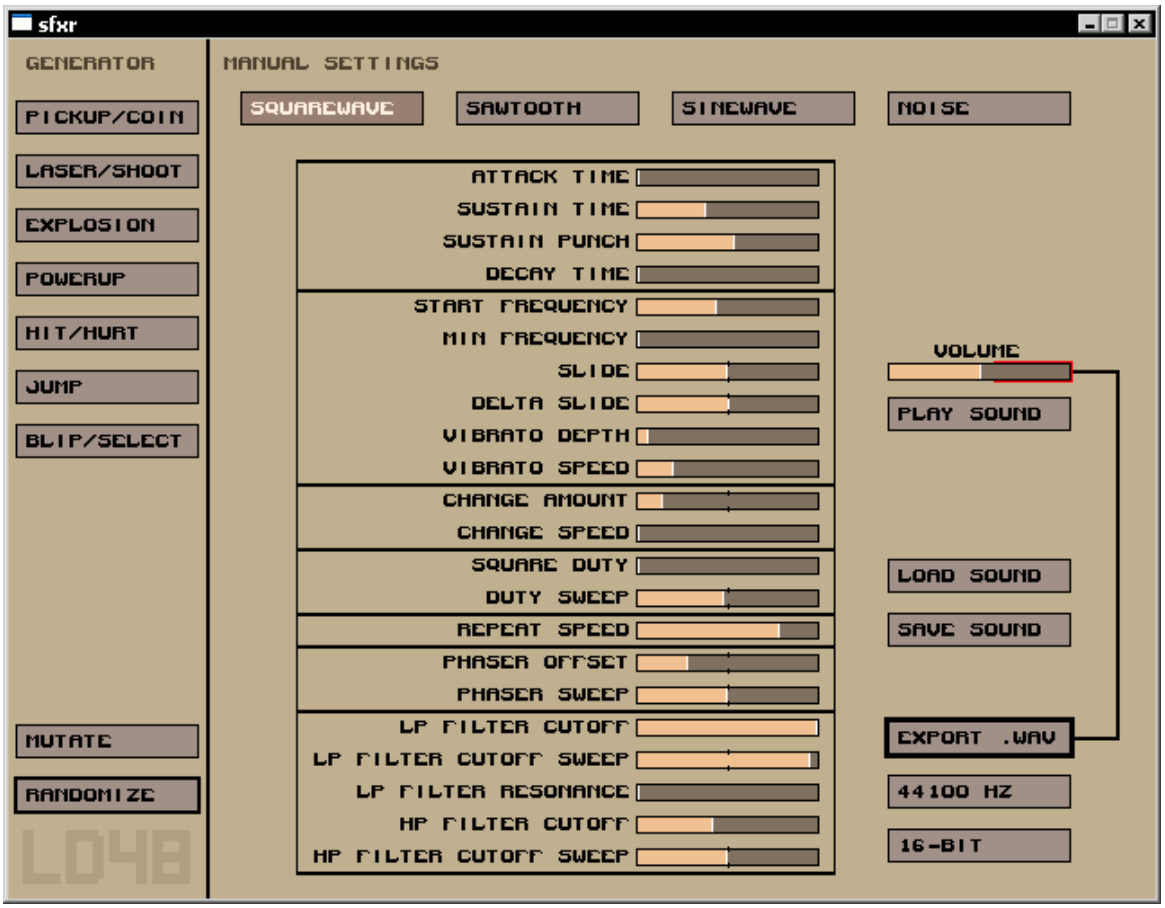


Figure 5.2: The SFXR user interface with category buttons on the left, adjustable parameters in the center and exporting options on the right.

explores the question *"how can effect sequences be generated through algorithmic means?"* as well as *"how does sequence generation impact the design process?"*

The sequence generation in Squeezer is inspired by the simple yet powerful approach in SFXR [154] and its purpose, as described by the author Tomas "DrPetter" Petterson:

Its original purpose was to provide a simple means of getting basic sound effects into a game for those people who were working hard to get their entries done within the 48 hours and didn't have time to spend looking for suitable ways of doing this. The idea was that they could just hit a few buttons in this application and get some largely randomized effects that were custom in the sense that the user could accept/reject each proposed sound.

The approach is based on a synthesizer with a series of buttons to randomize parameters resulting in a wide range of sound effects. Seen in Figure 5.2 there are eight "category" buttons (including "randomize"), that each randomly adjusts parameters within certain preset ranges. Each category generates a sound effect with distinct characteristics, such as "jump" generating what can best be described as the sound of a jump curve. Petterson achieved this by identifying parameters common to each category and providing acceptable ranges for those.

The purpose of adopting this approach in Squeezer is similar to the purpose described by Petterson, providing a simple means of getting basic effect sequences into a game. Interestingly Petterson also notes:

Anyone else in the same situation (need some basic sound effects, don't really care about top quality, have no idea where to get them) should find it pretty useful, ***if nothing else then just as placeholder sounds to kill the silence until final content has been produced.***

The sentiment of using placeholder sounds to kill the metaphorical silence is interesting in the larger context of adding juice effects to a game. Squeezer can be used (if nothing else) to create placeholder effects that *remove the emptiness until final content has been produced.*

Besides providing people with a way to generate placeholder sound effects for their games, SFXR also allows them to tinker with the generated parameters. This tinkering opportunity provides a playful platform for exploring and understanding how each parameter affects the generated sound. The categories provide a frame of reference for exploring what can otherwise be a daunting possibility space. Designing is a learning experience, and exploring the design space allows a designer to discover how elements interact. Offering support for guiding novice designers through design space exploration is one way of offering support for a design task.

Through an effect sequence generator, designers can explore the possibilities of specific areas in the game feel design space. In addition, using a sequence generator allows novice designers to learn which elements make up well-known types of effects such as "jump", "impact" or "explosion".

5.3.1 Guiding the Sequence Generation

In order to understand the impact of sequence generation, it is relevant first to answer the question *"how can effect sequences be generated through algorithmic means?"* As with all procedural content generation, there are many ways to achieve the goal. In the case of Squeezer, the descriptions including effect sequences already define the shape of the output. A hierarchy of effects must be generated, preferably a sensible one. Because Squeezer allows effects to be nested (delayed until the former effect finishes), randomly generating a tree of effects is as easy as selecting a random effect in the hierarchy (or nothing, i.e., the top layer) and adding another effect to it. The number of effects added can be adjusted, and the tree structure can be flatter or deeper depending on the probability of selecting an effect from the hierarchy or nothing.

While generating random sequences of effects could inspire adventurous designers during brainstorming. Generating sequences that conform to specific categories would let designers search for sequences that match their needs more easily. Drawing inspiration from SFXR [154], categories like "explosion", "impact" or "jump" can be dissected into the base elements that occur in a standard effect sequence of that kind. In SFXR, the synthesizer has over twenty parameters that control how the generated sound wave is created, and the synthesizer maintains a preset for each parameter for each category. Some parameter values are fixed, while others

are randomized within a specific range. This way, the "explosion" button can generate random sounds with a quick initial burst of low frequencies that slowly die off to maintain a rumbling sound. Similarly, Squeezer maintains a preset hierarchy for each category; an example of this hierarchy can be seen below:

```
Explosion example
- Sound Effect: Explosion
- Screen Shake: Duration=0.05-0.1
- Positional Flash: Sphere shape, Duration=0.05-0.2
    - Scale to Zero
- Particle Puff: Sphere shape, Count=10-50
    - Fade to black: Duration=0.5-2.0
        - Destroy
    - Scale to Random
Optionally for big explosion also add debris and fullscreen flash:
- Fullscreen Flash: White, Duration=0.01-0.05
- Particle Puff: Square shape, Count=5-10
    - Fade to black: Duration=0.1-0.5
        - Destroy
```

Generating the same explosion sequence every time might be helpful when a designer knows the sequence they want is similar to the "recipe". However, as a tool for exploring different options, that is not very helpful. Instead, Squeezer randomizes the generated effect sequences in two different ways.

First, every effect has randomly initialized parameters within some predefined ranges. Second, the generated effect sequence is mutated. The mutation algorithm steps through the effect sequence, and for each effect, it has a chance to remove the effect and add a new nested effect. Additionally, the mutation algorithm also slightly mutates the effect parameters (which can cause them to move outside the predefined "reasonable" ranges). Combining these two features creates effect sequences that align with the original recipes but can diverge in many different ways.

Aside from the presets and initial randomization in each effect. The sequence generator takes a category and intensity value. These values are used to build a base sequence with several effects nested to create the desired intensity and type (like an explosion with more or less "bells and whistles" effects). The sequence generator then runs the base sequence through the mutation system, creating additional variety. The generated effect sequences are still mostly interpretable as the intended category. However, as with SFXR, the output is not always in line with the designer's wishes. Luckily, there are several ways to deal with this issue:

1) The designer can generate a new sequence based on the same settings.
2) The designer can mutate the whole sequence to explore the local search space.
3) The designer can lock parts of the effect sequence, so the mutations cannot modify them.
4) The designer can manually tweak the values of individual effects in the sequence.

Each effect group has an expandable "generator" menu to accommodate these options. Figure 5.3 shows the generator interface with the expanded generator menu and four locked effects. Locked effects are unaffected by mutation and regeneration. Regeneration is used to generate a new sequence; the generator avoids changing the locked effects and will avoid generating additional effects of the same type. For example, a locked sound effect makes the generator avoid additional sound effects when regenerating. This locking allows the designer to keep core parts of a generated effect and regenerate the rest.

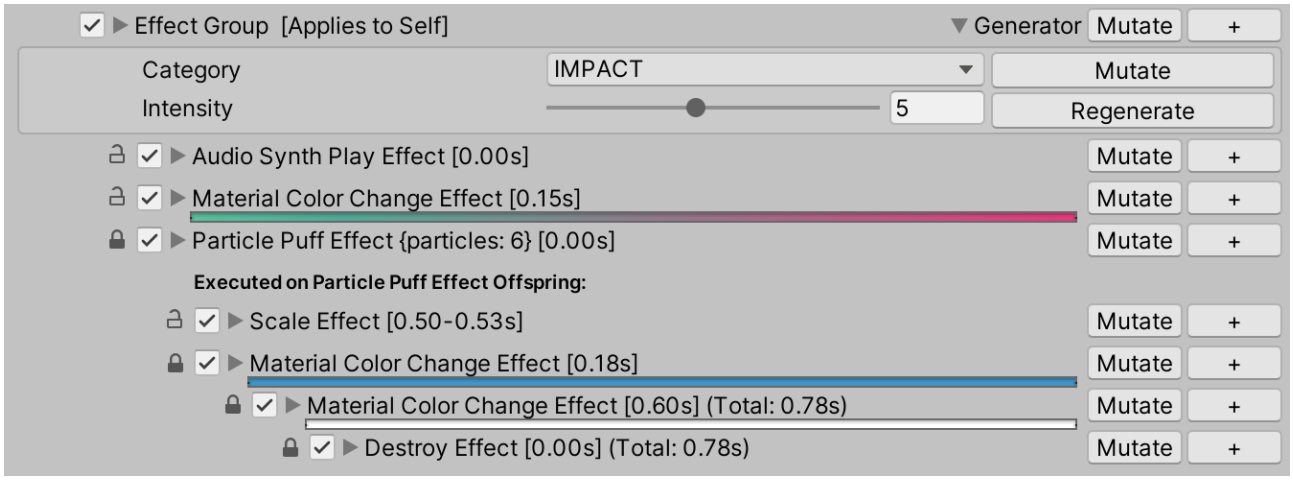


Figure 5.3: The effect tree generator interface featuring a drop-down menu for selecting a category and a slider for determining intensity. The mutate and regenerate buttons respect locked parts of the effect tree.

5.3.2 Reflecting on Sequence Generation

With the introduction of sequence generation, it becomes interesting to ask, "how sequence generation impacts the game feel design process?" *(RQ3)*

While visual sequence design allows the user to quickly create an effect sequence and evaluate it using previews or running the game, a designer still has to build the sequence before evaluation and iteration can begin. Sequence generation makes it easy to get into an evaluation and iteration loop quickly. It can be beneficial for exploring the design space and different combinations of effects. The generated sequences are based on recipes. A designer can learn how each effect influences the final result by generating and testing multiple variations or disabling different parts of the sequence. Even novice designers can quickly evaluate a generated sequence and decide if they want to regenerate, mutate, keep parts or tweak the parameters manually.

The sequence generation has only been evaluated indirectly through the interactive evolution interface (discussed in the next section) and automated game design. From personal experience, the author can say that sequence generation can jump-start the juice design process. It will also, at times, generate sequences with surprising "side-effects". These side-effects can cause unexpected changes to game mechanics that can inspire new ideas and changes in the core gameplay. A side-effect example the author has seen is the discs in the DISC ROOM [24] demo "scaling up" after hitting a wall. Larger discs are much harder to avoid, making the game more hectic. This side-effect could easily be turned into a new enemy type in DISC ROOM. In that light, the generated sequences can foster creativity, not just for juice design but also for the overarching game design.

5.4 Interactive Evolution

Building on sequence generation and mutation and to allow further exploration of the design space of generated effect sequences and matching it to gameplay, an interactive evolution [185, 179, 235, 60] interface was built for Squeezer [94]. We wanted to explore the possibilities of co-creativity [233] in Squeezer with a mixed-initiative [144] approach. In this setup, computers and humans collaborate to explore the design space to take advantage of their individual strengths. While the computer can quickly generate many artifacts, audio-visual artifacts (with temporal aspects) are hard to evaluate procedurally. Interactive evolution replaces computer evaluation in an evolutionary algorithm with human evaluation. The designer guides the design exploration by selecting the most suitable or pleasing artifacts. The computer then generates a new batch of artifacts based on the designer's choices, and the evaluation process starts over. This cycle continues until the generated artifact is good enough for the designer's needs.

For Squeezer, we hypothesized that allowing users to explore the design space through interactive evolution would be a good way to learn what the tool can do. We also envisioned that it would be a gentle introduction to various effects for people who are new to the idea of designing juice effects. With the ability to generate and explore effect sequences, novice juice

designers could build confidence before looking at the descriptions and tweaking parameters manually.

The effect sequence generator in Squeezer provides an excellent foundation for building an interactive evolution interface. The mutation system allows interactive local search of effect sequences through mutating and previewing them in the editor. However, while previewing effect sequences is fantastic for iterating on a design, the sequence generator UI only accommodates a single sequence at a time. Further, the effect sequences also need to be evaluated in the full context of a game and its interactions. The interactive evolution interface is designed to accommodate multiple sequences and full context evaluations while playing the game. This interface allows the designer to manually or automatically switch between different sets of effects to evaluate the game feel of each set. While it can be hard to play a game and design the juice effects simultaneously, playing a game and evaluating how well effect sequences fit the interactions is a much simpler task.

Interactive evolution interfaces usually display multiple artifacts side by side (e.g., Picbreeder [179]). This type of interface allows the user to get a quick overview of the underlying population. However, playing every artifact at once becomes overwhelming quickly for temporal artifacts such as sound or juice effects (which can include sound). Instead, such systems must allow users to select the artifacts they would like to evaluate and "play" them separately. In addition, juice effects are hard to evaluate in a vacuum. Ideally, they are shown in a game context. However, that means reloading the game scene with the correct context to create a "clean slate" for evaluating individual effect sequences. Squeezer contains a set of demo scenes (both 2D and 3D versions, an example is shown in Figure 5.4) for this purpose. Each scene is automated and displays a conceptual interaction for triggering effects (such as an object *jumping*, *shooting*, or *landing*). These scenes provide a clean stage with simple, repeatable events, where a designer can focus on getting feedback for a specific interaction correctly. Another option is to use real game scenes, which provide the full context, but that can be messy and might need the designer to provide player input.

The interactive evolution in Squeezer is relatively simple, generating a set of eight different effect sequences for the selected trigger event. The interface (seen in Figure 5.4) cycles through each of the effect sequences automatically, resetting the scene before showing the following individual. The time between switching individuals can be adjusted and even paused if the user wants to examine a specific effect sequence more closely. When ready, the user can select one of the eight effect sequences as the seed for the next generation of individuals. A new generation is created by copying the selected individual eight times and mutating each one some amount. In the future, it would be interesting to implement more advanced versions of interactive evolution with larger populations, multiple generations of change, and displaying a diverse set of individuals based on their novelty. The goal of the initial experiment with interactive evolution was to provide a proof of concept for a mixed-initiative approach to juice design. To assess whether interactive evolution would be the correct direction and how well the interface would work for this exploration.

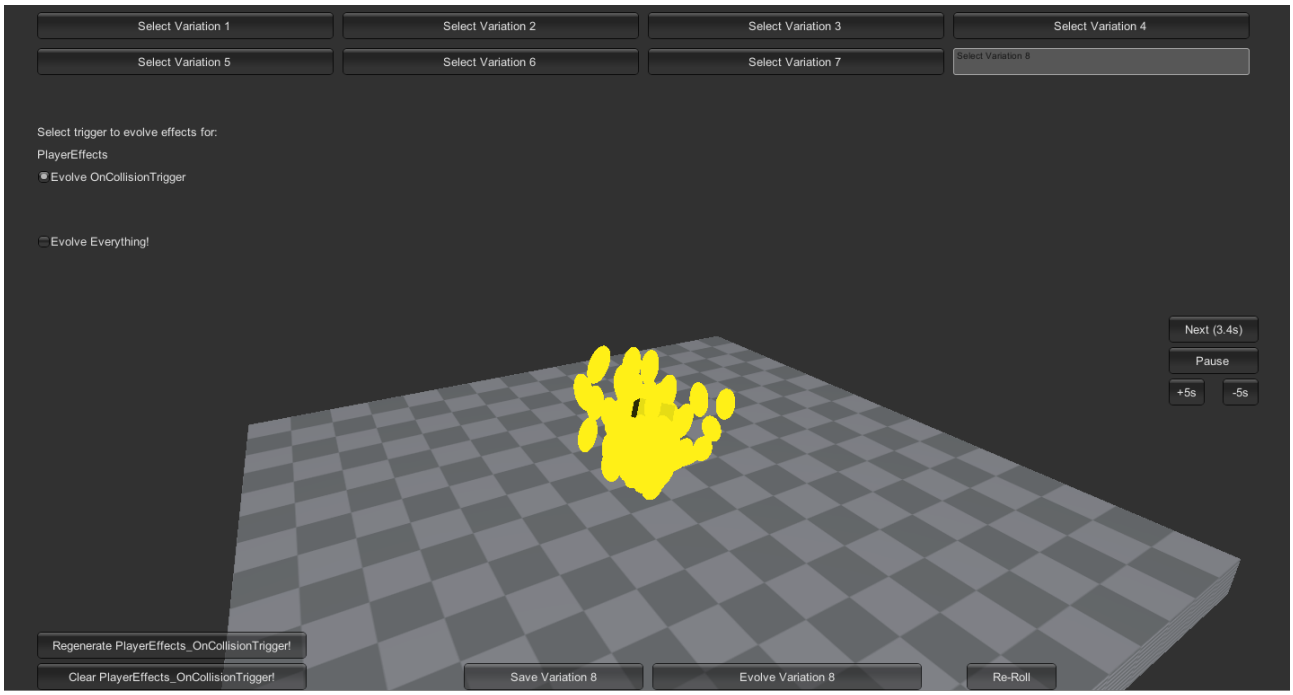


Figure 5.4: The run-time interactive evolution interface, seen here showcasing an explosion effect triggered upon object collision in a 3D scene.

5.4.1 User Test

We evaluated the new interface and the improvements to Squeezer through another qualitative user study in [94]. Once again, selecting users with a wide range of game design experiences. Our initial test found that novice users were excited by adding and removing effects to sequences [95]. We wanted to confirm our initial findings with this second round of user tests and test the effect exploration through the interactive evolution interface.

Five experts with no prior experience with the tool were invited to evaluate Squeezer. They had between two and fifteen years of experience making and designing games, and they had used several techniques to implement juice effects. The user test was conducted through a virtual meeting, with the users running the tool on their own computers while video conferencing with the author. Following the one-two hour sessions, the participants filled out a survey answering questions about their experience, usual approach to game feel design, and experience working with Squeezer. The user test was split into two parts, an open-ended exploration using the interactive evolution interface to design effects for a game prototype (DISC ROOM [24]). The goal was to get the users familiarized with the tool and its effects and determine if and when user fatigue would set in. User fatigue is a common issue in interactive evolution interfaces, as users become tired of evaluating artifacts. In some systems like Picbreeder [179], the artifact evolution is shared among all users, allowing multiple users to build off of each other's work. In Squeezer, the designers evaluate effect sequences while playing the game, potentially altering the onset of user fatigue. We expected that the experts would explore the evolution relatively quickly while getting familiar with the effects.

Following the first test phase, when our participants expressed the desire to accurately tweak the effects or guide the exploration. They were presented with the editor UI for manually tweaking and mutating the effect sequence and the option to lock parts of the sequence. One surprising takeaway from these tests is that the mutation locking system provided a way to shape the exploration in the interactive evolution. Because the locking system is based on a hidden data field on the effects themselves, the locks are transferred from selected artifacts to their "offspring" in the interactive evolution. This transfer lets designers shape or limit the exploration to certain parts of the effect sequence. In addition, the designers can lock effects that are nearly good enough and tweak values manually, then evolve the rest of the effect sequence. The granularity that Squeezer provides on the mixed-initiative spectrum (see the spectrum of agency diagram in Figure 2.3) allows designers of varying confidence levels to explore the design space with as much support as they wish. Reaching almost from full PCG to fully manual designing, Squeezer can accommodate a wide range of needs to help users find the proper effect sequence for a game prototype. Our participants found Squeezer's potential as a mixed-initiative tool for designing juice effects very promising. Some expressed interest in using it for learning, teaching, and prototyping effects. *One participant found it "chaotic, but also playful." It pushed them out of their comfort zone while helping them to "explore a possibility space too large to contain" in their head.* [94]. Playful exploration has been part of the goal for Squeezer since the beginning, and these preliminary user tests indicate that it has been somewhat successful. We also observed how less-experienced participants in our user tests tended to use the interactive evolution longer than the more-experienced designers. Additionally, while using the standard UI, less-experienced designers would explore the design space more often by using the generator and tweaking the descriptions. More-seasoned participants would rely on manually exploring the combinations and possibilities of various effects.

Our user test showed us that displaying effects one at a time was not the perfect interface for an interactive evolution. While the interface worked, it required much explanation, and it was somewhat confusing to work with. More experienced users would quickly note that the evolution was going too slow for their liking, and they would love to tweak elements themselves. The shorter evolution sessions indicate that the presented artifacts require greater diversity or better options for guiding the process than simply selecting the best individual.

A few participants noted that while Squeezer seemed well suited for game jams and small prototypes. They had concerns concerning moving from the prototype effects generated by Squeezer to the effects of a finalized game. As Squeezer currently targets the prototyping phase, the current effects lack both performance and how much they can be adapted to the aesthetic of a game. However, Squeezer is extensible enough that production quality effects could potentially be implemented. These production quality effects could easily rely on more heavily optimized sub-systems in Unity (or other game engines) to perform the heavy lifting while leaving only the sequence design to Squeezer. One tester who usually relies on writing

code requested an API to trigger an effect sequence from code[9].

I think I would prefer a more lightweight version that just let's me explore effects that I trigger manually (since I'll be writing C code for everything).
I'd love a Squeezer.Trigger("ExplosionA"), where ExplosionA is a preset that I can save/load-/evolve/inspect/...
I don't think I would want Squeezer to be in charge of the event triggering, but I understand it's use for more visual design

From the comment it is clear that they still think the option to generate, mutate, evolve, and edit the sequence manually through the UI, is valuable and important for other developers. To summarize the preliminary answers to how interactive evolution impacts the game feel design process *(RQ3)*:

- The test indicates that interactive evolution can surprise and even inspire designers to try new things.
- Interactive evolution seems to appeal more towards less experience designers, allowing them to spend more time exploring the effects and their impact.
- Without significant diversity in the presented effect sequences and meaningful ways to impact the direction interactive evolution quickly looses the interest of more experienced designers.
- To make further claims about the impact of interactive evolution the interface has to be improved.

5.5 Discussion

Looking back at the main research question *how can* ***design tools*** *support* ***game feel design****?* This chapter answers the question *"what is the impact of design tools on the game feel design process" (RQ3)* concerning Squeezer.

It is hard to claim that the impact of design tools in general on the game feel design process *(RQ3)* has been described in this chapter. However, it has provided initial insights into designing support and how it affects the game feel design process. There is still a lot to be done, primarily related to generating and evaluating game feel. While preliminary testing of the impact of mixed-initiative support has been done through evaluating an interactive evolution interface [94]. More research is needed to explore new algorithmic approaches and possibly discover other ways of providing assistance or support.

[9] Although this was not presented as an option during the second user test, Squeezer can serialize (save) and deserialize (load) effect sequences from files. These serialized effect sequences can then be loaded and executed through the API. Serializing effect sequences provides additional opportunities for sharing and collaboration between multiple projects and designers, as well as using a code-based approach.

Like game feel design is meant to invoke certain feelings and elicit certain responses and moods. There is also intent behind other types of software; in this case, game development is the overarching purpose. However, as discussed in the first section, the intent behind the prototyping tools explored in this thesis is playful exploration. The interface invokes elements of a design philosophy to inspire playful exploration and spark creativity in the users. Squeezer is meant to provide a playful exploration first approach to juice design, where traditional methods require more asynchronous actions (writing code and designing without immediate feedback). Supporting designers in the creative process with a tool that allows playful exploration of the design space by employing strategies identified to excite the playful human [85]. Many of these traits are shared by design tools like Doodle Studio 95! [161] or SFXR [154], and is what Victor calls for when urging developers to "Stop drawing dead fish" [227]. Victor calls for the design tools to provide direct manipulation and strive to present the designer with as much immediate feedback as possible.

Squeezer allows designers to explore how different effect sequences affect their games before thinking about how to implement systems that can execute them. Visual sequence design combined with an interactive description "language" provides the means to explore potential combinations of available effect types and visualize their output (through preview mode or running the game) without prior knowledge about designing juice effects. Immediate feedback is provided by previews of the designed effect sequence when rapid iteration is required.

Building sequences manually still takes time, and it requires either learning or knowing which general effect types go together to create the right effect sequence for the game event. Sequence generation supports novice designers while learning by creating reasonable variations of common structures for specific categories of events. At the same time, experienced designers can use sequence generation and mutation to get started or explore niche variations of effects quickly.

Interactive evolution of effects was tested in Squeezer in order to provide further support for novice designers and designers who are looking for inspiration. Limited user testing indicates that novice designers enjoy the heavy-handed guidance provided by interactive evolution more than experienced designers. While experienced designers find it neat, they more quickly wanted to manipulate the generated and mutated effect sequences directly. However, the participants did indicate that interactive evolution created options they did not expect. When the presented options that differ from the ideas the designer is exploring, they can sometimes reveal new and unexplored game ideas. Without a system like interactive evolution, or the sequence mutation system, designers might never find or try these alternative possibilities.

Tools for supporting a more open and playful approach allow better creative exploration of game feel design. They allow novice designers to engage with the subject more easily. Moreover, tools that support a design-first approach allow designers to stay in the creative mindset without concern for the underlying systems.

Besides the features explored in our user testing, Squeezer allows designers to save and load the effect sequences. Building a library of effect sequences can impact how a designer approaches

game feel design. Such a library could be used as suitable placeholders for common patterns they use or as a starting point they tweak to fit a specific game. A library of effects could replace the sequence generator for some designers. At the same time, they could still create some variation by mutating the loaded sequences manually after they are loaded. Designers could also share their designs with others using Squeezer or build a collective database of juice effects. A database would be great for novice designers to choose from and to learn the art of game feel design. A database of effects could also provide an interesting base for building game feel design support through machine learning and recommender systems.

Summarizing the answers found concerning the question *"what is the impact of design tools on the game feel design process" (RQ3)*:

- Indicated through testing of the visual sequencing in Squeezer's interactive description UI. Playful design interfaces beneficially impact the engagement and exploration of the possibility space.
- Designers appreciate the interface providing an easy way for designers to explore a set of juice effects and their impact on games. Allowing them to focus on the design exploration rather than on the implementation details of the underlying system.
- Though sequence generation has seen little direct testing, the generator can surprise and delight users. For example, in the interactive evolution, the generator provides the initial set of effect sequences, which surprised and intrigued the test participants more than once.
- Sequence generation can impact automated game design by providing the foundation for adding juice effects to the generated games.
- Interactive evolution can support less experienced designers in familiarizing themselves with the possibilities a tool provides.
- Diversifying the set of effect sequences shown in interactive evolution may provide more meaningful exploration, especially for experienced designers who otherwise quickly lose interest.
- Multiple interaction modes allow tools to support different designer workflows. By providing a flexible mixed-initiative interface, designers can settle on the exact amount of support they need in their work.

The full impact of design tools on the game feel design process is hard to discern. However, through the testing of Squeezer, it seems the impact is a positive one. With a better focus on design exploration and the potential to discover new and exciting combinations and game mechanics.

5.5.1 Automated Game Design

Having multiple different interaction modes allows different users, including AI agents, to use Squeezer. Interestingly, the API for saving, loading, and triggering effects through code that one of the test participants from [94] requested formed an excellent starting point for integrating with AGD systems.

Just like human users need to adapt to using new tools, AGD systems need to be adapted to use new tools. Apart from adding a juice design mode into the system, the whole visualizer had to be rebuilt to accommodate running effects between actual game turns. The integration between Squeezer and the AGD system relied on determining the most basic events used (e.g., *spawn piece*, *destroy piece* or *move piece*). The system had no way of performing lookups for color palettes, mood boards, or other design tools human designers would use. So the AGD could not adjust the colors or intensity of the effects to suit the game's properties; instead, it would match sequence generation categories to these event types. Besides this, games generated by Puck could easily use the *destroy* and *spawn* events to change a board piece into another piece, and the current implementation would not capture this context. Instead, such a "change piece" event would execute both the effects associated with the *destroy* and the *spawn* events, which could quickly become confusing. An AGD trying to add juice effects to a game needs to adjust effects based on their context. That context is based on color palettes, aesthetics, and the game's atmosphere. The more significant implication is that "meta" events such as "change piece" or even more complicated combinations (e.g., scoring a board state and adding points to the scoreboard) need to be used as context for the designed effects.

Supporting game feel design in an AGD system requires a way to analyze effects in relation to their game context. Additionally, comparing multiple effect sequences to each other and evaluating which one is most appropriate is equally important, and both remain open questions for future research.

One suggestion we propose is to perform A/B testing[10] online, using human player preferences, to select the correct effect sequences. For instance, either by asking human co-authors to evaluate the effects or by creating online polls for people to vote on the most suited effects.

The following chapter will discuss *"how the addition of game feel design tools alters the qualities of games" (RQ4)* by exploring how designing effects using Squeezer changes the resulting games. The main results for that discussion stem from testing games generated by Puck using Squeezer in the AGD process. But also include examples and experiences by the author and participants in the expert user tests of Squeezer.

[10] See https://en.wikipedia.org/wiki/A/B_testing

Chapter 6

Designing Game Feel with Squeezer

If you really want to understand something, the best way is to try and explain it to someone else.[1]

Related Papers
Johansen, Pichlmair, and Risi, *Squeezer - A tool for designing juicy effects*
Johansen, Pichlmair, and Risi, *Squeezer - A mixed-initiative tool for designing juice effects*
Johansen and Cook, *Challenges in generating juice effects for automatically designed games*

This chapter focuses on *"how the addition of game feel design tools alter the qualities of games?" (RQ4)* Reiterating first and foremost the goal and intent behind adding juice to games. As well as on surveys from Kao [105] and Hicks et al. [84], on the influence of game feel.

The results presented in this chapter are based on adding juice to games in an AGD process [93]. This chapter presents the numerical ratings of games with basic tweening animations, games with generated effects, and human-designed prototype effects. As well as a broad stroke analysis of the comments the participants provided with their ratings. These comments were not published as part of the original paper [93] due to page limitations. Additionally, the chapter reflects on the use of Squeezer by the author and test participants in the two user tests outlined in [95, 94].

Finally, the chapter discusses the potential impact of game feel design tools for future game development. How game feel design tools could alter automated game design and game AI research. Both in terms of the games produced and, more importantly, in terms of the interest in the field.

[1] From "Dirk Gently's Holistic Detective Agency" by Douglas Adams, 1987

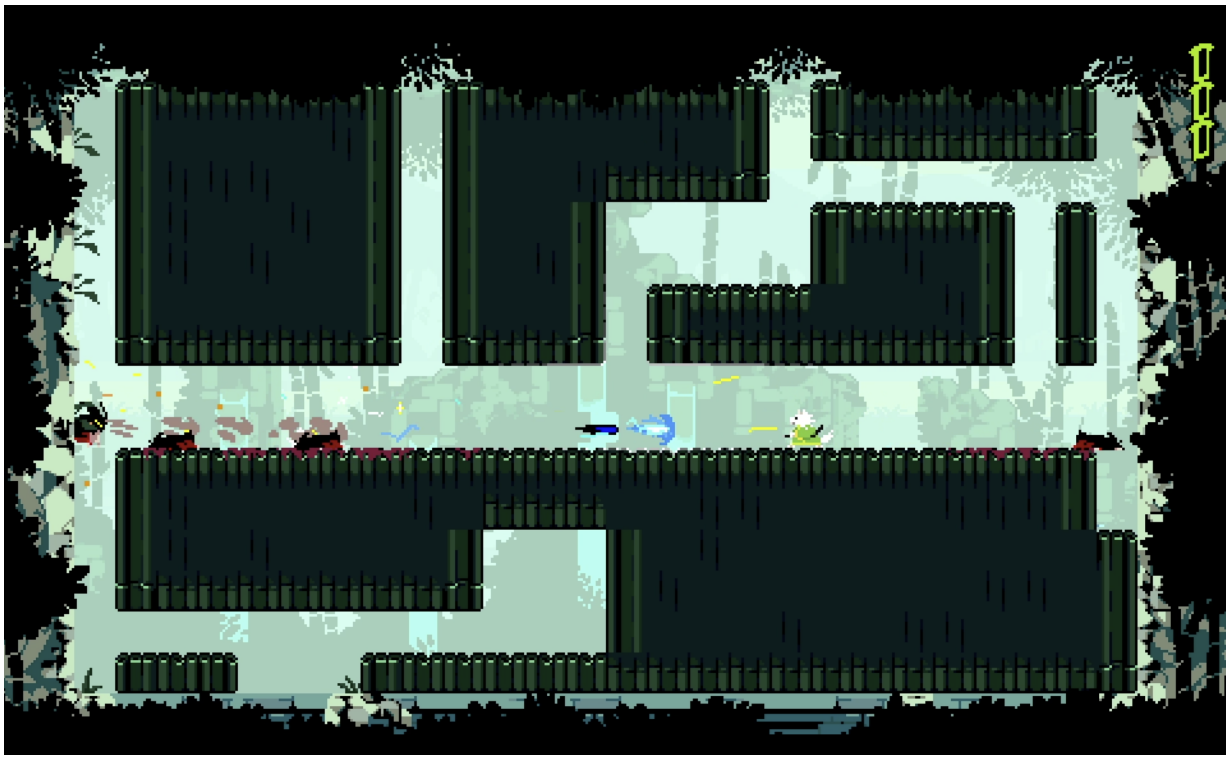
6.1 Game Feel Design Reflections

As discussed in Chapter 3, the goal of juice is to amplify game events, providing excessive amounts of feedback to the player in order to convey the significance, value, and effect of the interaction. An example could be hitting an enemy in Samurai Gunn [216] (see Figure 6.1). A hit momentarily freezes the game, highlights (horizontally or vertically) the rows/columns where the event happened, and causes a white flash to appear over the hit character. Additionally, red particles are splattered in the same direction as the impact after the game resumes. Freeze frames and blacking out the rest of the screen to highlight the hit guide the player's vision to notice the event. Additionally, splattering red particles over the nearby area conveys the game history and signify where players fought.

Juiciness, however, does not just convey the action and significance of the game; it also provides the player with *"[...] an immediate, pleasurable experience."* as Hunicke says [88]. Providing feedback for every little thing happening in the game, player provoked or otherwise, makes the game feel richer and more alive. Hicks et al. [84] says *"Juicy design refers to the idea that large amounts of audiovisual feedback contribute to a positive player experience."* In their article exploring *"how exactly feedback needs to be constructed to be perceived as juicy"*, by surveying 17 developers, creating an analysis framework, and applying it to two commercials (arguably juicy) games. The framework created by Hicks et al. attempts to make designing juiciness more actionable for designers and academics. They highlight some of the potential properties resulting from adding juice to a game. For example, do the mechanics of the game translate into expected feedback? Does the game directly respond to the physical input of a button? Are the reactions to action exaggerated to detail state change? The questions are meant to provide actionable design points that could potentially lead to "better game feel". Hicks et al. highlight several interesting points from their survey, including *"Juice should be used to direct the players attention, not divide it."* Indicating that juicy feedback can become overwhelming if improperly applied. This leads us directly to Kao's work [105], where Kao surveyed over three thousand participants to determine how the amount of juice influences the game experience. Tracking both playtime and actions and asking a small set of participants to rate how well effects fit the actions. Kao found that both no effects and extreme juice effects led to lower playtime and that *"Juiciness has a significant impact on player experience. Specifically, both the No Juiciness and the Extreme Juiciness conditions thwarted player experience as compared to Medium Juiciness and High Juiciness."*.

The fact that extreme amounts of feedback lead to less player engagement indicates that it functions as a friction [168] the player has to fight. Extreme juiciness then becomes part of the resistance [4] the game provides, instead of supporting [57] the player in their understanding of the game.

On the other hand, adding juiciness in moderate amounts leads to higher engagement, longer playtime, and players interacting more with the game's systems. As Juul describes it: *"Juiciness does not simply communicate information or make the game easier to use, but it also gives the player an immediate, pleasurable experience."* [102] Kao interestingly only notes



An intense battle between two characters, where the one on the right is about to be hit by the blue bullet.

The same battle a split second later, after the character on the right has been hit. Everything but the "row" with the action on has been blacked out.

Figure 6.1: Samurai Gunn

the detrimental effects of no or extreme juiciness. Another way of looking at the results would be to state that moderate amounts of juiciness remove one type of friction [168] from the game. Lower friction allows players to explore the game experience further and allows games to live up to their full potential more easily.

6.2 Reflecting on Squeezer

Reflecting more directly on how Squeezer alters the qualities of games *(RQ4)*? As discussed in the previous section, juiciness in moderate amounts allows games to live up to their full potential. As such, it could be inferred that game feel design tools like Squeezer allow games to live up to their full potential more easily, as they make designing juiciness easier. Tools like Squeezer makes it easier to explore many types of juicy feedback, letting designers more easily evaluate if the amount of juice is adequate. Having a tool that makes it easy to apply effects in any part of a game increases the designer's chance to explore the design space further. Without tools that make designing effects easy, they will sometimes be overlooked or ignored due to time constraints. As such, without tools for game feel design, the game feel might suffer involuntarily as a consequence.

All through the development and testing of Squeezer, the descriptions and the underlying sequence generator have surprised and intrigued users. Similarly, the interactive evolution has surprised both the author and our test users by suddenly adding effects that influence gameplay. While Squeezer was never meant to change the mechanics of a game, sometimes emergent features can be revealed through the exploration of effects. Accidentally designing new game mechanics when experimenting with juice effects is a wonderful side effect of using Squeezer. This type of design exploration helps designers illuminate parts of the design space during the prototyping phase. Knowing more of the design space allows designers to decide more effectively which parts of the design need to be cut and which need to be kept. Prototyping is all about testing different approaches, ideas, and solutions. Many juice effects are closely related to game-altering mechanics or features; with a juice design tool like Squeezer, it is more likely for these to be explored as "happy accidents". However, they might not be as easily discovered in traditional workflows, as the designer is less likely to stumble upon them during development. These properties allow Squeezer to be part of the more extensive design exploration, impacting the mechanics and the feedback and appearance of a game. Encouraging lateral thinking [53], not just by experimenting with various combinations of feedback effects, but also leading to exploration of new game design ideas.

Examples of areas where juice effects and mechanics closely overlap:

- Adjusting the size of objects is a common juice effect.

  Goal: indicate various physical reactions, build anticipation, follow-through or signify a weight change.

  Side-effect: if the size is not returned to the original quickly, it can cause objects to be interpreted as more or less threatening (if the collider or hitbox grows with the object, they might even alter collision detection). For instance, the discs in the DISC ROOM [24] demo growing to double size makes them a lot more challenging to dodge. This influences gameplay and, as such, can be explored by the designer as a potential enemy type.

- Blinking by making the object alternate between transparent and opaque.

  Goal: Showing an object was hit or is invulnerable.

  Side-effect: Making things such as enemies or the player character entirely invisible for short periods can make control more challenging. If the duration of invisibility is too long, it can become hard to control the player character or dodge potential enemies while they are invisible. Similar effects can happen with localized or fullscreen flashes of color that indicate impacts or damage. Fullscreen flashes could also simulate lights going out in a horror game.

- Slowing time down for a split second.

  Goal: Short bursts of slow-motion functions like freeze frames allowing the user to perceive what happened.

  Side-effect: Slowing time down for extended periods works more like bullet-time, making the game less challenging or allowing the player more time to deal with a challenge.

- Adding physics simulation (ragdoll) to units or characters that are unconscious or dead.

  Goal: A common juice effect for making objects more interesting after they stop interacting normally.

  Side-effect: Adding it to a player or non-player object while they are being controlled can lead to new unintended challenges. For example, in our early user test, a user added this to blocks in Breakout, so they would start falling when hit instead of being destroyed immediately. The falling blocks added a surprising additional challenge to the game, as they would still interact with the ball while falling. A designer might look at that and add a new block type into their game that has to be hit twice and begins falling after the first hit.

Flexible and versatile game design tools allow designer expressivity. The more "happy accidents" a tool like Squeezer brings into the design process, whether from co-creative support mechanics or because some elements are closely related and easy to explore, the more valuable it

becomes while prototyping. Not only does Squeezer allow designers to more easily explore and find the right type of juice effects for their game, but it also allows them to laterally explore different game mechanics and ideas closely related to these types of effects. The examples listed above are just a few examples of how Squeezer can influence game mechanics and design exploration.

6.3 Results from the AGD experiments

To explore how games created with an AGD system are altered by adding juice design into the process. We can look to the results from [93], where we added juice design into an AGD system and created a user study to verify how it performed.

The user test was designed to show if the perception was improved by adding effects and verify that generated effects needed to be customized properly to be competitive with human designs. We generated three sets of effects for each game (Antitrust and SameGame) for the test. Each participant was presented with three different versions of each game, one implemented with a simple, flat, linear tweening animation, one selected randomly from the three generated versions, and the last with a set of effects generated and then adjusted by a human expert. We hypothesized that the flat tweens would be rated lower than the generated effect sets (because they are more lively and exciting), and those, in turn, would rate lower than the versions we created manually (because they would be more contextually appropriate). Hypothesizing that even with the most basic implementation of effect sequence generation using Squeezer, the games would be rated better than games without effects (and, in turn, also better than games without animations), based on results from the work of Kao [105].

The AGD system, also known as Puck, generated the game we named *Antitrust*[2]. While a manual description of *SameGame*[3], was created in the file format understood by Puck. The manual game implementation provided a familiar game archetype usually rich with juicy feedback for the user study of the generated effects.

Participants would play three versions of each game and fill out a survey as they went along. A website served the user test online, and through links on social media, 113 participants chose to play and rate the games. In order to avoid play order bias, the website handled randomizing the order and versions each participant saw.

A slightly lower number of participants filling out the survey for Antitrust (only 106 participants) than SameGame (113 participants). We presumed this difference was related to the challenge and potential frustration of playing against the AI opponent. In Table. 6.1 we collected the average scores of each game and performed a significance test. The scores are grouped based on which generated effects the participant was shown. We argue that although these results do not establish significance, they can be interpreted to indicate that effects do

[2]Naming the games is another task Puck has yet to tackle.

[3]A classic casual game where you destroy groups of three or more orthogonally connected pieces of the same color.

make people enjoy them more. We also see that the AGD system has difficulty competing with a human designer without any alignment of the generated effects.

SameGame	Base		Generated		Expert
Gen1	4.11	$\sim$	4.06	$\sim$	4.49
Gen2	4.33	$<$	5.36	$\sim$	5.31
Gen3	4.17	$<$	5.23	$\sim$	5.63
Antitrust	Base		Generated		Expert
Gen1	4.69	$\sim$	5.31	$<$	6.54
Gen2	3.95	$\sim$	3.39	$<$	5.08
Gen3	4.42	$\sim$	4.24	$<$	5.24

Table 6.1: Average ratings each build received. Ratings are grouped by which generated build participants were shown, indicated by the row. Symbols between columns show significant orderings or $\sim$ for cases where significance could not be established..

Apart from the ratings in Table 6.1, the survey asked participants to fill out why they selected the specific rating they chose. Comments in Table 6.2 and Table 6.3 have been selected from the top-ranked comments for each game, based on a variation of *TextRank* [7][4], which returns the most representative sentences from set of sentences. The summarization tool did not have access to the ratings provided by the participants, and it is interesting to see the most representative comments be spread out across low and high ratings. For the commentary, the comments were grouped by game version, not accounting for presentation order or other games seen. While in Table 6.1 the ratings for the base and expert versions were grouped and only counted with the generated version they were shown beside.

For the singleplayer game (SameGame), we see from Table 6.1 that the base game is significantly worse than the versions with effects (both generated and designed) in two out of three cases. From the survey comments (representational selection seen in Table 6.2), we see that the slow tweening animations impacted the feeling of flow for the players. We also see the participants felt the lack of sound detracting from the experience. The overall consensus seemed that while this version was smooth and straightforward, it felt uninteresting to the participants. When looking over the comments from the games with generated and designed effects, it appears there is an expectation for match-three games to provide excessive amounts of positive feedback. This expectation is in line with what Juul & Begy note concerning game feel in casual games [104].

While significance could not be established between the generated effect sets and the designed set, it is interesting to reflect upon the comments for each of them. The sounds in the generated effect sets got many comments, both because they existed and because they were perceived as annoying or discordant. From the design perspective, these generated effect sets were never evaluated before the user test, and as such, they were incoherent with the game-

[4]Using the Gensim 3.6.0 summarization tool https://radimrehurek.com/gensim_3.8.3/summarization/summariser.html

play. While the designed sound effects got slightly better reviews, many participants noted a mismatch between the high definition art style and the 8-bit sound effects. Both the expert design and the generated effect sets got comments about the faster animation speeds and particle effects, making the interactions more satisfying. However, the particle effects were not aligned with gameplay due to the lack of design evaluation for the generated effect sets. According to participants, especially for one game (Generated 3), the particles grew slowly and lingered too long on the screen. For another game (Generated 1), the pieces would blink in and out while moving, irritating some participants. However, for all three generated games, some participants would note that the personality and aesthetic themes did not fit the gameplay. Though no significance could be established between generated and expert in any of the three cases, it was interesting to see that the average rating of the expert design versus the second generated game (Generated 2) favored the generated game. As the top comments also indicate, many participants found that particular set of effects playful, simple, and nice, though they still found the sounds to be a bit off. The relative success could indicate that the generator may have found a set of effects that worked better for the game than the expert design.

The expert effect design was built with specific constraints to make it comparable to the other games. There were six different backgrounds and five different sets of pieces to choose from. The combination of background and pieces for the generated games were selected via dice rolls as the AGD system did not yet have an art style design mode. The expert was only allowed a limited time to prototype effects using Squeezer and selected a combination of pieces and background that they felt would fit the best. The time limit was arbitrarily set to a single day for creating effect sets for both the singleplayer and multiplayer game. The low timelimit was selected to avoid too much iteration by the expert because the AGD system did not have any way to iterate on the generated effects. According to the comments, the sound effects and particles created by the expert were more cohesive and in tune with the gameplay. However, some participants still found the low-fidelity sound effects to be a bit too sharp and irritating.

For the multiplayer game (Antitrust), Table 6.1 indicates that both the base and the generated versions do significantly worse than the designed version. From the survey comments (representational selection seen in Table 6.3), we see that both a lack of effects and excessive effects in many cases disagree with the participants. Unlike the casual singleplayer games, where abundant amounts of positive feedback are commonplace, multiplayer board games (even digital ones) usually provide more subtle feedback. Different expectations for the game type could explain the ratings. Where the expert designer could more easily adjust the feedback, the AGD system still generated explosion-like effects for piece removal. From the comments on the expert design, it appears that the background and pieces provided players with a sense of familiarity, as many referenced the idea of a Go[5] board. This familiar setting may have provided a better context for the game than the base and the generated versions.

From the comments on the base version of the game, the slow tweening animations appear

[5]Go is a board game with roots dating back thousands of years, originating in Asia: https://en.wikipedia.org/wiki/Go_(game)

SameGame - Base	Rating
Background is ok. Pieces look a bit too flat: more shinny and pseudo-reflexes could possibly be better. Apart from the aesthetics, the timings of the animations feel weird, slow, and clumsy - if I were [evaluating] that, it would be a lower score still	5
The animations are painfully slow and impact the general game feel and the lack of sound gives no impact to any action.	3
Clear colors, text boxes look a bit generic, I miss having sound effects	6
Generic shapes; background of "field" didn't mean anything; pieces moving slowly felt uncomfortable; no sound	4
The animation is attractively smooth, but also quite slow; on the one hand, this looks nice and makes it very readable, but on the other it's very ponderous and simple, and breaks up activity without holding a lot of interest.	5
SameGame - Generated 1	
Basic shapes, annoying sounds	3
This feels unified but simple: bright, flat colours, big clean shapes and a rounded, accessible font all point in the same aesthetic direction. [...]	5
[...] I liked how it presents layers of feedback by making large plays that affect more of the pieces even more aurally distinct, even if it sounds kind of alien and odd.	5
I like the simple style but the colour of some of the pieces were similar to the background making it harder to play. [...]	7
The faster animations and particles indicating interactions make the game more satisfying, but the discordant sound effects are deeply irritating and makes enjoyment impossible.	1
SameGame - Generated 2	
Sound is a bit annoying, visuals look playful and simple, pieces and background are partially consistent.	7
[...] The sound effect and visual effects of the pieces collapsing and then moving make it very appealing. Color and shape being different for [every] piece group is also appealing.	10
The flat style and sound effects were cute, and the explosion sound effects and particles were nice. [...]	8
The visuals are more cohesive than the first game, but I like them less, and sounds are a bit strange.	6
Faster tweening, and sound effects, makes game more exciting.	6
SameGame - Generated 3	
Weird grey dots after things are deleted. The sound effects are a bit annoying.	5
The squealing sound was funny but slightly high-pitched in my opinion, especially if you hear it several times because a lot of pieces have to group up together	8
[Has] a nice little plop effect for the drop, I like how the separate pieces scoot over next to one another	5
The assets feel like they belong in different games. The effects feel slow and glitchy, like the explosion black dots, and the smoke cubes growing and staying too long. [...]	3
Sounds and animations are too much, but their speed feels good.	4
SameGame - Expert	
[...] the tiles didn't feel like they fit with the sci-fi background, and the 8-bit/sfxr style samples felt out of place with the crips HD artstyle.	3
The effects and movement were better (still not great) but the background didn't fit the gems at all.	6
Pieces look better than in the other game. Animation on pieces blowing and sounds are a plus. Pieces clumping speed no longer feel awkward with the sound effects. I miss some animated feedback on the standing pieces, or when hovering over them.	9
Background was too visually noisy and the sound effects were sharp and irritating	2
[More] cohesive "industrial" theme; effects when groups disappear; sound; but almost too busy & distracting	6

Table 6.2: **SameGame** - Selected representative comments for the question "Why did you select this rating? (Optional)" for each version of the singleplayer game. The right column is the rating (1-10) the user provided with their answer. Note that games were presented in random order. Some comments have been abbreviated and marked with [...], and spelling mistakes have been fixed and marked like [this].

to have worked reasonably well; however, many participants found the lack of sound and other effects a bit jarring. Multiple participants found it unengaging, dull, and lacking "a bit of punch". Interestingly, these comments seem to be the opposites of the reasons for adding juice effects to a game, which is to make the game feel more exciting, alive, and with amplified actions. Another observation, which the multiplayer survey shares with the singleplayer survey, is that people distinctly notice effects when missing. One participant in the singleplayer survey explained how they had to go back to check the other versions for effects they suddenly felt were missing. This feeling of missing effects speaks to the "invisible" nature of game feel design and juice effects. These effect types feel like they belong, even when they appear a bit jarring. When participants point out that it feels dull or lacks punch, they speak for a desire for excitement and amplification of events. From Table 6.1 we see that, on average, the singleplayer base game was rated lower than the multiplayer base game. There could be many explanations for the lower score. However, a reasonable assumption would be that multiplayer games, especially boardgame-like games, are not expected to provide the same abundance of positive feedback as the singleplayer match three-like game. Additionally, part of the game feel and friction when playing multiplayer games is provided by the challenge of interacting with the other player(s). Because the players all interact with the game world, they require less feedback from the game itself to acquire a sense of progress and friction.

The comments for the generated games highlight some interesting points. Participants noted that the background and pieces did not match satisfyingly for all the generated versions. Interestingly for one game (Generated 1), one participant notes that they cannot imagine a person looking at it without having ideas for how to improve it. As the AGD system created the generated games, no person had indeed looked at them. However, that version was, on average, rated higher than the other two. While many comments noted the mismatch between background and pieces, some also stated that they enjoyed the destruction effects when losing pieces. Specifically, they noted that the sound effect matched the awful feeling when losing a piece quite well. For another version (Generated 3), one participant noted that it looked like a prototype made by a programmer where the aesthetics had been considered. Another participant notes that it looks like an early prototype, referring to the look of the particle effects. For the last of the three generated versions (Generated 2), participants noted that it would generate large lingering clouds when placing pieces. They found this effect obscured the opponent's actions, and with the addition of a flashing effect when pieces were destroyed, many participants found this version annoying and distracting. Harsh effects like these being annoying is similar to the findings by Kao [105] when participants were presented with "extreme" effects; it tended to make them less engaged. With the multiplayer games, the ratings (see Table 6.1) were on average worse for two of the three generated versions compared to the base version. The lower scores could indicate that AGD systems need to account for the number of players when adding juice effects for games. However, the general dislike of the combinations of visuals (background and pieces) in the generated games could also have influenced the games' ratings.

For the designed version of the multiplayer game, participants commented that they enjoyed

the visual coherence and the visual effects. However, they also noted a mismatch between the peaceful visual setting and the sound effects. While the designer had toned these sound effects down by adjusting various parameters in the synthesizer and lowering the volume of the effects, they still had a crunchy quality to them that participants found unfitting for the game. Some participants even found that the game was a little too peaceful and said it might benefit from music or animations of unobtrusive elements. These are great suggestions and indicate that when building games, AGD systems could benefit from a design mode focused on entertainment value and the game's presentation. Considering the time player spends waiting between taking actions and providing actions or visuals that can entertain the player in the meantime. One such example is the interactive board elements, and ground tapping animation in Hearthstone [12]. These elements have no impact on the game state but provide the player with something to do while waiting for their turn or pondering their next action.

The survey intentionally only asked for a rating and a reason for the rating, intending to have participants describe their feelings towards each prototype. Each participant was also allowed to play each version multiple times, comparing back and forth between versions. Some participants would rate every game very low (1-3), while others would rate them high (6-10), alongside participants playing the same games and giving scores in the entire range. The survey was meant to spark interest and provide a starting point for thinking about and designing game feel in AGD systems. Many avenues need to be explored beyond adding generated juice effects to the games based on a selected set of events. Like evaluating where game interactions need amplification, altered physicality, or have to provide player support, and deciding how the game should be juiced, tuned, or streamlined [155] to achieve this. New evaluation methods have to be created to check that the solution achieves the intended game feel and how coherent they are with the overarching aesthetics of the game.

The expert designs were by no means perfect (personal opinion of the author and as indicated by the survey comments). While the Squeezer generator can generate category-based effect sequences, they could easily be improved to provide more exciting variations. Additionally, the number of categories could be increased to generate more specific effect sequences, providing both AGD systems and designers with choices more specific to their purpose.

Issues to be addressed in future user studies include comparing singleplayer or multiplayer games independently. As it stands, the survey and comments indicate that generated effects for singleplayer games score higher than in multiplayer games. Expert designs are higher rated for multiplayer experiences than singleplayer ones, which seems strange. It is also worth mentioning that the AI difficulty was selected to give the human players a good chance of winning in Anti-trust games. The difficulty adjustment was made to ensure participants would play all three versions of the game without giving up due to frustration. However, seven fewer participants completed the multiplayer survey than the singleplayer ones. However, it is unclear if that was caused by frustration, boredom, fatigue from playing all the versions, or just a general bias against multiplayer games. In any case, as the opponent(s) in multiplayer games provide a lot of the friction the player experiences, the difficulty of these opponents can significantly affect

Antitrust - Base	Rating
While the visuals communicate clearly and are not unpleasant, the lack of feedback and the slow animations makes the game feel dull and unengaging.	1
The lack of sound is a bit jarring, but the smooth animations work very well for the game.	7
[No] sound, the piece-adding animation was slow, but at least it didn't cause physical pain to play	4
I liked how it was super smooth, but after playing the first game the lack of sounds made me think it lacked a bit of punch	6
Utter aesthetic mismatch, no sound, slow basic size animations, no special visual effects.	2
Antitrust - Generated 1	
Graphics are great and cohesive, animations are a bit rough. No delay on [the AI] placing pieces looks bad.	7
Comic sans instructions, thematic disjoint between the space background image and the jewel play pieces. I just can't imagine that a person would look at elements of this and not have ideas for alterations.	4
Don't like the sun/earth background, it clashes with the colour of the pieces.	5
The gems are also not very appealing to me, probably because I don't think about them as typical pieces in a board game.	3
Flickering when placing was a bit unpleasant, but the "crunchy"/"hitty" sound and destructive visuals fit nicely with the feeling that losing pieces is bad.	8
Antitrust - Generated 2	
The effects when placing a piece, and when 4 pieces are cleared were too large and distracting.	4
The clouds are quite noisy and get in the way of seeing what's happening! The gems and space background are passable, they're just graphics. [...]	5
The sound effects were sort of unpleasant to hear repeatedly, the gem clearing animation obscured the next opponent move, and sometimes the previous gem placed would flash on placing the next one. Generally just a lack of polish.	4
The animation for disappearing is borderline nauseating, with an annoying flashing effect, and it is slow. It feels overall ugly and cheap.	1
[The] "cloud" when placing a piece stays around way too long	6
Antitrust - Generated 3	
I liked the crunchy sound effects! I expected graphics in a lower resolution though, it sounded like an 8bit game who maybe should have 8bit graphics	8
[The] placement particle effect staying on too long made it look quite janky, the gemstone tiles didn't really gel with the planetary [background] to me	5
It looks like a prototype made by a programmer, not something where the aesthetic matters, but I don't really care enough about visuals to have a strong opinion	4
Game pieces look cheap, no connection to background.	2
The little shapes flying off the pieces look old-fashioned, or like an early prototype, but in general i like the idea of the way they move.	7
Antitrust - Expert	
[...] As a result the lack of activity on the screen makes the game feel very empty, even though compared to the several of the singleplayer puzzle games I played in the 1st section, it still has feedback like sound effects, [...], effects on tiles being placed and cleared, I still somehow expected/would want more passive activity going on for a game of this pace, perhaps music, or an unintrusive visual element that was animated.	4
The graphics worked well, with the background appearing to be a Japanese shrine and the black & white pieces recalling a Go board, but the shrill electric-like sounds did not fit the otherwise peaceful aesthetic.	5
The visuals communicate clearly and have some thematic coherence [...], the faster animations, sound effects and slight visual effects make interactions more satisfying.	2
[I] like the black and white pieces; matches the Asian-looking image (I'm thinking of Go). I like the immediacy of placing tokens but what's up with the audio? Doesn't match.	5
Pieces and background are coherent, animation is nice and polished, sound effect doesn't fit with the theme and is not a pleasant sound.	5

Table 6.3: **Antitrust** - Selected representative comments for the question "Why did you select this rating? (Optional)" for each version of the multiplayer game. The right column is the rating (1-10) the user provided with their answer. Note that games were presented in random order. Some comments have been abbreviated and marked with [...], and spelling mistakes have been fixed and marked like [this].

the game feel. Future studies would also benefit from the games having the same or comparable base aesthetics; the background and pieces being different for each game led to many comments completely ignoring the visual effects. Lastly, the inclusion of both a base and an expert design and comparing three different versions of a game, unfortunately, obscured the results a bit. Participants could be asked why they preferred one version over another to get more precise indications in future studies.

In [93], we proposed, tested, and evaluated how even simple integration of juice effects in a game generation process can improve the generated games. Our tests indicate that effects should be considered when creating game generation systems. We argue that using more play data, the effect generation process could be shaped to more accurately determine when and how effects should happen and determine what they should look like (based on player colors, event type, and frequency of the event).

Adding juice design to an AGD system breaks new waters within game generation by introducing a new area of research, which we believe is essential. The importance is rooted in two points. One point is showing both how easy it is to add effects (using Squeezer) and how effects can improve the perception of games. The second point is improving the play experience of generated games to draw more players' and researchers' attention to the artifacts and the field. With more engaging and exciting play experiences, there is a better chance to catch the interest of players and researchers who might not be aware of the field of AGD. The test results did not confidently show that generating juice effects is better for games. However, the fact that no effect games rated significantly lower than the same game with human-designed effects using Squeezer still points to future research potential.

6.4 Discussion

Without game feel design tools, AGD systems that want to include game feel design can either only use pre-build snippets with nice game feel or possibly figure out a way to add game feel objectives as part of the core goals. However, in all honesty, that last one becomes a game feel design tool in and of itself, as it would need ways to adjust that part of the design based on trying to reach the objectives.

With tools like Squeezer, AGD Systems can begin to engage with effect design as a first step in exploring game feel design. Moreover, while there are many challenges to overcome in that space, adding effects to generated games seems not to be detrimental. According to some survey participants, the games become more exciting, engaging, and entertaining simply because the interactions are emphasized. Game design has come a long way since its earliest days, and players expect a certain level of feedback and attention to detail in games today that current AGD systems have not yet reached. Maybe they will never reach the same level. Instead, AGD systems need to be implemented with Human collaborators in mind unless the games target AI agents who cannot say no to playing when they find them uninteresting.

How are games affected by the addition of game feel design tools? The simple answer is that

tools like Squeezer allow the designer to explore/sketch out effects very early on in the process. A common thing that happens when people make (or begin making) prototypes or game jam games is that sound effects are either entirely left out or added as an afterthought. When that happens, the sounds can feel mismatched to the rest of the game. With game feel design tools like Squeezer, it becomes so easy to add effects early on that they become a part of the system. Instead of amplifying certain game events with juicy effects to make them more convincing, the designer can add effects that are "good enough" early on. By doing that, they get to experience these effects firsthand while designing levels, mechanics, challenges, and polishing other parts of the design. Then, instead of adding effects, juicing becomes the task of adjusting effects that are already there, polishing and designing them to provide the feeling they need to elicit to be a part of the gameplay. In essence, using tools like Squeezer juice design more easily becomes part of the design iterations. Although the initial effects created with Squeezer will probably be replaced at a later stage of development, they can increase the chance for a game to succeed in conveying the experience the designer intended.

The earlier game feel design becomes part of the design iterations, the more attention the designer will give it, and the more the player will feel that attention to detail shining through. As Hunicke suggests, it is essential to be "Loving Your Player With Juicy Feedback" [88]. Or as Nijman wraps up his presentation, "that's pretty much all I have to say about game feel: just fill your games with love and tiny details" [140]. Design tools allow designers to realize, shape, and evaluate ideas. Game feel design tools highlight an aspect of game design that might easily be overlooked. It is hard to say how games are changed by adding game feel design tools into the process. From the Squeezer user tests in [95, 94] it seems that experienced designers learn to shape and design game feel to elicit the experience they want the player to have. That behavior indicates that these designers might have internalized the game feel design process. However, even experienced designers know the importance of adding mockups, placeholders, and sketches while building a game to evaluate the whole experience and better visualize the design space.

Squeezer might only provide experts a stepping stone towards the game they envisioned. However, the game is not always a clear vision for many designers (some experts included) and can easily be susceptible to changes and new ideas throughout development. Squeezer allows designers to quickly adjust and experiment with new ideas during development by providing support for sketching and playful exploration of effects. While it is hard to prove how a game is impacted by using Squeezer, the added focus and ability to explore game feel has the potential to bring more creative visions to fruition.

In future work, it would be interesting to perform large-scale A/B testing on generated and human-designed effect sets for a single game to test the performance of the effect sequence generator in Squeezer. Such a test could be used as a baseline to test generator improvements, better categorization, or how well effect sequences provide the essential parts of a specific effect category. These baselines could also be used to compare other PCG systems for effect generation. In such a test, the game juiciness framework created by Hicks et al. [84] could

serve as the base for surveying the subjective juiciness of individual effect sets.

It is hard to claim that the question: *"how does the addition of game feel design tools alter the quality of games?" (RQ4)* has been answered in full this chapter. However, through the survey results from adding juice effects to games produced by an AGD system and a general discussion of adding juice effects to games, potential and preliminary implications have been noted. The AGD survey from [93] indicate that compared to a game with very basic effects, games with juice effects do not perform worse than their juiceless counterparts. The effect sets generated by Squeezer for the AGD system, overall did not perform as well as the sets designed by a human designer. Which indicates that there is a lot of potential future research in this area, to both understand the context of game events, as well as designing effects that adhere to and amplify the contextual information correctly. Still the results are similar to those obtained by Kao [105] by applying various intensities of juice effects to an RPG game. Kao discovered that both no effects and extreme amounts of effects led to lower play times (equated to less engagement and interest). The base set of tween effects in [93] is arguably still a set of animation effects, which may have increase the performance of the base game. Some of the generated effect sets could also be categorized as extreme juice effects, like the obscuring "clouds" participants would see in Antitrust (Generated 2), is an example of an effect that was too extreme for its purpose. The ratings did not significantly determine how well the generated effects performed compared to the base and expert games. The average rating values show the base games score significantly lower in general than the expert juice effects.

In addition, altering the perception of games by adding juice effects can help engage players in future user studies, even if the topic of the study is entirely different. As an example, trying to gauge how well a game can maintain a "flow" [50] state by adjusting the difficulty to suit the player, also known as Dynamic Difficulty Adjustment (DDA) [89]. The game will have to also to provide ample amounts of immediate feedback (see Chapter 3, Section 3.2) to keep the player from losing interest. Suppose juice effects are left out of such a DDA study. In that case, the results might wrongly indicate that the level of challenge was not correctly adjusted. When, in fact, the player lacked proper immediate feedback to understand the game world and feel a connection to the game. Not out of boredom, due to lack of challenge, but out of a lack of connection to the game world.

Throughout the development of Squeezer, the goal was to build a tool that could assist in the process of game feel design. While assistance has been explored through interactive effect sequence descriptions, effect sequence generation, and interactive evolution. A future research task remains to improve the design process, allowing designers and computers to cooperate as increasingly equal co-creative partners. Finding ways to leverage algorithmic evaluation processes or encourage lateral thinking, similarly to what Sentient Sketchbook [128] does for level design.

Besides, it is hard to know how moving effect design into focus will change the resulting games. What happens if a designer adopts a juice or game feel first approach when designing games? Would we get more toy-like games such as Townscaper [198]? Games where the interface

is almost invisible and the interactions are so in focus that objectives are either self-imposed or pushed far into the background. Would games feel as great as Super Mario 64 [141]? As Rogers [168] writes *"Friction was so important to Super Mario 64 that — so goes the legend — early in its development, Shigeru Miyamoto asked for a 'White Room': a blank, empty stage in which Mario would be terrifyingly alone. In this white void, Mario could run and jump endlessly without fear of death . . . or any kind of goal whatsoever. The purpose of the White Room was to allow the programmers to fine-tune the feeling of every little jump, exactly to the game designers' specifications."*

Chapter 7

Discussion

I may not have gone where I intended to go, but I think I have ended up where I needed to be.[1]

Showing an opportunity exists for building tools that assist designers in new ways during the critical prototyping phase of game development. This dissertation has presented work outlining a new sub-field of research within the field of computational creativity, contributing to the mixed-initiative and game generation research fields. Squeezer represents the embodiment of the presented research. Research that has been disseminated through related papers and talks at academic and industry events throughout this project.

While it is too soon to tell what the more significant effects of this Ph.D. project will be, colleagues, peers, test participants, and reviewers have noted that the work is breaking new ground. They have also expressed ideas for expanding learning opportunities and applying game feel in their teaching and game development tinkering. The survey paper [155], which functions as part of the background knowledge for this dissertation, provides a base for exploring the current state of the art in-game feel design. A paper that would have been exciting and extremely helpful to a certain young and inexperienced game developer beginning a career in games over a decade ago. Squeezer [95, 94] both a tool for prototyping juice effects in games and a learning environment for people trying to understand juice implementation. It is an open-source tool that allows other researchers and developers to build alternative interfaces, add different types of assistance, or integrate it within their systems.

In Section 7.1 the research questions and related contributions are reviewed. Followed by Section 7.2 discussing potential future research related to a wide variety of topics touched upon in this Ph.D. project.

[1] From "The Long Dark Tea-Time of the Soul" by Douglas Adams, 1988

7.1 Reviewing Contributions

The contributions of this project are based on the research questions, the experiments, and their results. In the effort to answer the main question *how can **design tools** support **game feel design**?* I decided to break it down into four narrower questions:

- RQ1 *"What is game feel design?"*
- RQ2 *"How can game feel design support get implemented in existing workflows?"*
- RQ3 *"What is the impact of design tools on the game feel design process?"*
- RQ4 *"How does the addition of game feel design tools alter the qualities of games?"*

7.1.1 Game Feel Design

To contextualize the first research question *"what is game feel design?" (RQ1)* The primary resource for the subject of game feel in the past decade has been Swink's book "Game Feel: A Designers Guide to Virtual Sensation" [214]. However, as the subtitle suggests, it is a guide to (a specific type of) virtual sensation, and as pointed out by Wilson [230], "Game Feel" is not the same as game feel. There is no single correct recipe for how a game should feel, but there are many elements that influence how a game feels (for examples, see Table 3.2). Interestingly, Swink has said that he had envisioned the book as a starting point for further exploration of the subject. Because as time passed, people have explored the subject of game feel in many directions, the concepts and vocabulary surrounding it have drifted similarly. Not least because terms like juiciness [75, 102, 98] describe concepts partial to game feel design, without necessarily relating to Swink's "Game Feel". In turn, the importance of juicing games was contested by the need to focus on streamlining (or oiling) [57], as the most important means to adjust game feel. Hicks et al. [84] attempted to coalesce the fluid concepts and ideas surrounding game feel by asking professional game developers to describe it. They created a framework for designing "good" game feel based on their questionnaire. While this approach certainly provided insights into the scattered definitions and understanding of the subject. Unfortunately, it did not help to streamline the vocabulary on the subject. In our survey [155] we instead attempt to create a coherent overview on the subject of game feel design in order to create an umbrella term for adjusting virtual physicality, juiciness, and player support in games. From Chapter 3 and [155] the main contributions can be listed as:

- Provide a survey of the history and state of the art in Game Feel Design, linking it with design and research in many other fields.
- Provide a list of design elements commonly used to adjust game feel and how each affects the game.
- Describe the three main design domains of game feel design and name the accompanying polishing task.

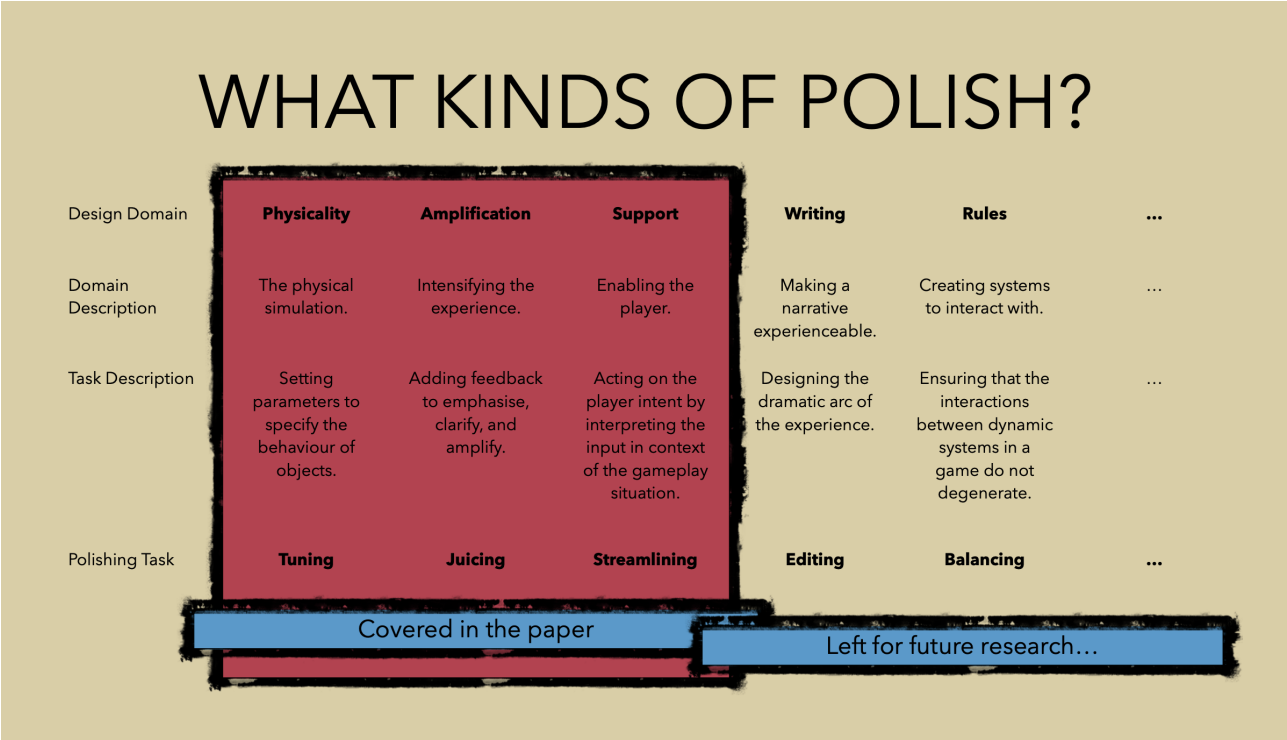


Figure 7.1: A slide from the presentation of "Designing Game Feel. A Survey." [155]. Presenting the domains described in the paper, along side examples of other design domains that influence game feel. Design domains which have been left for future research.

While the survey does not claim to provide all design elements linked to game feel, the list (see Table 3.2) is a good starting point for diving into game feel design and common ways to polish it. The identified design elements also have been linked to each of the three design domains of game feel design (see Table 3.1). There are more design domains related to how a game feels beside the moment-to-moment interactions, such as the overarching aesthetics, the narrative, and the game mechanics and challenges (see also Figure 7.1). Importantly each design element can also be linked to multiple polishing tasks in different design domains. For instance, effects like screen shake alter the perception of physicality, amplify the significance of the triggering event, and too much of it can distract rather than support the player.

7.1.2 Supporting Game Feel Design

Game feel design (see Chapter 3) heavily influences the experience of interacting with the game by shaping the perceived character and oomph of inputs and events. As discussed in Chapter 4 very few game creation tools exist that offer support for the design of moment-to-moment interaction. Often these tools lack ways to experiment with game feel design as a part of the game design exploration. There is a tendency among novice game designers to "forget" about sound as they are developing their first few projects, usually because the task seems too daunting at a glance. It is not until they are presented with tools like SFXR [154] or the novel

art of recording themselves making placeholder sounds that they start incorporating sound into their design process.

Similarly, many game engines and game creation frameworks lack inherent ways of "sketching" out the sounds and other juice effects while prototyping gameplay. Many newer engines, including fantasy consoles (e.g., Pico-8), include ways of supporting sketching elements like sprites, sound effects, and even music. However, support for game feel design has yet to earn a similar spot.

The lack of specific support for game feel design is what led to the question *"how can game feel design support get implemented in existing workflows?" (RQ2)* To answer that question, Chapter 4 describes experiments, hypotheses, and implementations of game feel design support in existing workflows. It covers implementation paths for game generation frameworks, automated game design, and general workflows like game development with the Unity game engine.

In particular, the chapter first shows three different potential solutions for implementing game feel support in VGDL. Examples of language modifications are presented based on previously unpublished experiments conducted during the development of UnityVGDL [96]. The modifications were meant to explore potential changes needed in VGDL, to be able to describe most of the effects presented by Jonasson & Purho [98] and Nijman [140]. Extending a language like VGDL to accommodate game feel design would require structural changes. Changes such as adding a set of juice effects (for event amplification), making the player classes more tune-able (for physicality adjustments), and making collision handling more adjustable (for supporting player maneuverability). This solution shows a direct way to implement game feel design support into a game generation framework like VGDL. Another more indirect potential solution relies on generating game prototypes for a different game creation system and allowing the designers to design game feel using another engine. The approach has previously been used in commercial game development, using Unity prototyped generated from Puzzlescript[2] implementations. The approach does not precisely add game feel design support to the description language itself. However, with a suitable pipeline for generating and editing the game descriptions and modifying them using an engine like Unity, it could be a compelling prototyping combination.

The core takeaway from working with VGDL descriptions is the possibility of creating a description-based system for juice effects. The language modification experiments clarified that most of the common effects for amplifying events could relatively efficiently be described and executed based on simple interaction definitions. Additionally, it is shown that using a nested tree structure, the descriptions also enabled easy sequencing of effects, making the sequences easier to understand conceptually at a glance.

Squeezer shows that it is possible to build on these ideas and implement an interactive sequence design system, including definitions for interactions and events that trigger the ex-

[2]An online puzzle prototyping framework based on a description language, available at: https://puzzlescript.net/

ecution of the designed sequences. More importantly, Squeezer provides the implementation details to add design support to the juiciest part of the process. Allowing visual exploration of effect sequences and previewing them in real-time makes it easier to get started with designing. Like Victor points out [227], designing with real-time previews allows designers to focus their energy on designing and evaluating, as opposed to spending most of it on imagining the resulting design. Chapter 4 describes various attempts and methods for shortening the iteration loops intended to allow designers to explore the design space more effectively.

As a third example, Chapter 4 shows how to implement support for game feel design in an automated game design (AGD) system through the use of Squeezer [93]. The potential approaches to implementing game feel design in an AGD system are often very similar to those available for game generation systems like VGDL. The presented implementation shows how to realize one such approach by using Squeezer as a procedural content generation system for juice effects. While the implementation approach is straightforward, it can add a new dimension to AGD research. Like humans have to adapt to new tools, Squeezer and the AGD system Puck show that AGD systems also have to adapt to new tools. Whether artificial or human, creative agents are part of creative workflows, and introducing new tools into these workflows requires both an appropriate tool and some adjustments from the agent. The AGD project presents some of the adjustments needed to both Squeezer and Puck for the implementation to work. Lastly, the chapter notes identified challenges and exciting problems requiring future research within AGD and general computational creativity. The biggest one is the issue of evaluating creative output, which has existed as long as the field of computational creativity itself, with Boden identifying it in the 90s [16]. A way to deal with this challenge would be collaborative game generation, where the AGD is designed to be part of a sort of "slow" design collaboration as described by Cook [42]. Generating games in a format that allows designs to be evaluated and altered externally by human designers. In this case, the AGD becomes a creative partner with a skill-set that allows it to generate, balance, and evolve games while human collaborators design the aesthetics.

While this set of environments does not cover every type of platform for game development, they do cover a reasonably extensive portion of game development workflows. Strategies for game generation (via description languages) and automated game design could extend to many other frameworks. Implementation details of systems and effects in Squeezer[3], can be used to implement similar systems for general workflows in other engines. Squeezer may also serve as a template for game generation, and automated game design implementations of game feel support. However, due to a lack of optimizations, the current Squeezer implementation is best suited for the prototyping stage of game development. Other tools or implementations would be needed for production quality and high performance. For example, implementing particle effects in shaders to run on the graphics processing unity (GPU) is a standard solution to performance issues. Though the juice effects implemented in Squeezer provide many ways of amplifying event signification (see Table 3.2), Squeezer lacks good ways to explore streamlining

[3]Remember Squeezer is available online as an open-source repository.

support and tuning physicality properly. Unfortunately, relying on triggering juice effects based on interactions and events focuses primarily on the amplification aspect of game feel. However, like sound and sprite editors have become commonplace in newer game engines, future game creation software may support the exploration of all three domains of game feel design.

7.1.3 A Playful Design Process

The intro of Chapter 5 describes the most common ways game designers currently go about designing game feel. These standard approaches are described based on personal experience, the answers from user surveys in [95, 94] and descriptions by other practitioners. The design process is often handled through custom code, tweening systems, and sometimes animation systems like the Unity timeline tool. Most of these approaches require expertise in knowing which adjustments to make and how to make the system do it.

Chapter 5 focuses on the impact Squeezer has on the game feel design process, providing a partial answer to the research question *"what is the impact of design tools on the game feel design process?"* (RQ3) The design intent behind Squeezer contextualizes the answers by clarifying the intended goal of creating a playful prototyping tool. Selecting the interaction requirements for playful exploration of design ideas, proposed by Hobye [85]. These interaction requirements are reinforced by design ideas proposed by Victor [227], as well as practical experience with using tools like SFXR [154] and Doodle Studio 95! [161]. Following the description of the intent is an investigation of the impact Squeezer has on the game feel design process. The investigation is based on user testing and anecdotal evidence from creating and conducting these tests. As well as the challenges and needs discovered while implementing Squeezer into an automated game design system.

The contributions derived from Squeezer's impact on the game feel design process can be summarized in the following way.

Identifying that visual sequencing increases exploration and lowers iteration time by removing direct coding requirements, allowing designers to focus on the design task rather than abstract and asynchronous coding tasks during the prototyping phase. Squeezer is not the first tool to do this, as approaches using tools like the Unity Timeline tool or MMFeedbacks [63, 64] are very similar in this regard (similar to Squeezer, MMFeedbacks also provides an ontology of effects triggered by game events).

Visual sequencing makes it easy to design, assess, and experiment with combining different ontology elements in new ways. Immediate effect previews allow the iteration loop to shorten. The lower time requirements make creative exploration more appealing, as changes are reflected immediately and can be reverted just as quickly if they are uninteresting. Exploratory creativity [16] based on combining unusual elements can have unexpected results, leading to lateral thinking [53] and transformational creativity [16]. Such as creative combinations fostering new game mechanics or entirely different gameplay ideas. Several Squeezer test participants exemplified this by attempting to or inadvertently altering the gameplay during their usage of Squeezer and deciding to keep the changes because they enjoyed the altered game more. Visual

sequencing and previews could be taken even further, allowing the designers to adjust various parameters directly in the game scene, using *Gizmos* and *Handles*[4]. Building and evaluating these features would be relevant approaches for future work on game feel design tools.

Squeezer also shows how effect sequences can be generated through algorithmic means and that generating sequences can avoid designers getting stuck on a "blank canvas" by generating a starting point. Having a starting point for exploring effects allows the designer to evaluate if an effect fits an interaction instead of creating one from scratch. By regenerating or mutating the effect sequence, the designer can form an opinion about the required sequence and adjust it accordingly. Squeezer's approach, similar to SFXR, for generating categories of juice effects (like impact, explosion, or jump) allows the designer to guide the sequence generation in the general direction they require. Though the list of categories Squeezer can generate sequences for is by no means definitive or extensive enough, it serves as a good starting point for effect sequence generation. It is left to future research to explore new categories and improve generation for current categories.

Our user tests with interactive evolution indicate that mixed-initiative interfaces can support novice designers in playful exploration. Where experienced designers quickly wanted more expressive freedom and the ability to adjust generated effects manually, novice designers tended to explore the interactive evolution for longer. The tests also show an opportunity for improving algorithmic support in the game feel design process. One example expert designers requested was guiding the exploration rather than only relying on the displayed variations in the evolution process. Providing opportunities for designers to define overarching goals for the exploration could be an exciting path for future interactive evolution research to explore. For example, by providing the evolution process with directions (e.g., more or less of something) or targets for crucial parameters like color, size, amount of movement, or particle counts.

Lastly, integrating Squeezer in an automated game design (AGD) system highlighted implementation and evaluation challenges when adding juice design into the process. The experiments [93] indicate that AGD systems need context-awareness both in terms of aesthetics and mechanics of a game, in order to provide the correct type and amount of juice. However, through simple matching of large-scale events (e.g., creation, removal, movement of game pieces) to Squeezer's generator categories. It seems possible to create a set of placeholder effects that indicate the feedback needed for given events. User testing even indicates that it is better than (or at least equal to) having no discernible juice effects.

Similarly, designers can use Squeezer's generated effect sequence to determine which types of effects fit a particular interaction. The AGD user testing indicates that players are much more expressive about games with effects than games that lack effects. It would appear that the addition of juice effects to a game opens the door to additional feedback on generated games, as opposed to pure evaluation of rules and levels.

[4] *Gizmos* and *Handles* are Unity terms for UI and interaction tools that appear when specific objects are selected. While the terms might be different, these types of direct manipulation tools exist in most 3D game engines.

It is hard to claim that this project has uncovered the full impact of design tools with game feel design support. However, it seems clear that there are many opportunities for further investigation of the subject, both in terms of visual representation and interface and mixed-initiative systems with generative and supportive properties. On top of that, game feel design is a brand new avenue for the field of automated game design to explore. With tools like Squeezer, built into their description languages, or something completely new, it is exciting to think about new goals for AGD systems, such as making interactions fun.

7.1.4 Game Feel Design with Squeezer

Chapter 6 attempts a direct and an indirect approach to answering the question *"how does the addition of game feel design tools alter the qualities of games?" (RQ4)* First, rooting the discussion of the impact of game feel design tools, in the elements described in Chapter 3, particularly in the design domain of amplification. Then providing anecdotal examples of observations from Squeezer user tests [95, 94], where users found novel gameplay mechanics by exploring the design space with Squeezer. Second, indirectly by expanding upon the analysis from [93] to answer the question "how does the addition of generated juice alter the qualities of games?"

One of the contributions of Chapter 6 is reaffirming the results from a user study testing different degrees of juice in an RPG game [105]. These results indicate that games without juice (and with extreme juice effects) are evaluated as worse than games with juice effects in them. Our results indicate that this might even extend as far as randomly generated effects being better than no effects. This result increases the future research potential of game feel design in AGD and other game generation systems.

Another contribution is identifying the benefit (or issue) that some common juice effects (when designed sub-optimally) can alter objects' perceptions and properties. The altered perception or properties can benefit lateral thinking and support design exploration, not just of juice effects but also entirely new game mechanics. This side-effect could change how we view effect design tools, and the role of tools that support game feel design play. Developers often exploit happy accidents and include unintended behavior as new game mechanics while developing games. A tool like Squeezer provides an opportunity to more directly explore the design space of happy accidents where juice effects and game mechanics collide and impact the game in new and compelling ways.

Lastly, adding Juice design to AGD and game generation systems can lift the perception of generated games. Five out of six game versions without effects were rated significantly lower than game versions with juice effects. However, the test also shows that juice effects may need to be evaluated in the context of aesthetics and the games' mechanics to increase ratings. This need is indicated by the fact that the generated effects were rated significantly higher than their no-effect counterparts in only two out of six game versions. While three out of six versions with generated effects were on average rated lower than their no-effect counterpart, they were not rated significantly lower. The numbers indicate that while it may not always

increase the perception of a game, the addition of generated juice effects to generated games is not significantly detrimental to the perception of those games.

It is, of course, hard to verify exactly how the addition of game feel design tools alter the quality of games. However, it is possible to get an indication by comparing games with no or extreme juice effects to games with some juice effects. Similarly, by evaluating the intentions behind adding juice to games, it becomes clear that amplifying the game experience with juice is an essential communication tool between designer and player. Like any other design discipline, the design informs how the user interprets and interacts with the designed artifact. It alludes to the importance of finding the design best suited for the designer's purpose. In games, that purpose is eliciting a specific experience for the player. As Lisa Brown puts it, "you're not juicing your game – you're actually picking a feeling that your game should communicate and juicing that feeling" [18]. However, while experienced designers may know or visualize precisely how to juice a specific feeling, most design requires a bit of experimentation and iteration. Tools can support this exploration by presenting attractive options, allowing combinations of common design elements, and speeding up iteration times. With shorter iteration times, designers can explore the design space more efficiently and find the design that best fits the intended experience and feeling.

7.1.5 Reviewing Research Questions

Reviewing the work in this dissertation would not be complete without looking at how well the project managed to answer the main research question. So once again we can ask *how can* ***design tools*** *support* ***game feel design****?*

In this project, the main research question was split into four sub-questions, each exploring an aspect of the main research question. Chapter 3 reviewed state of the art within game feel design, extensively answering the first research question *"what is game feel design?" (RQ1)* This explains the topic of **game feel design** from the main research question and the design intent and polishing tasks related to designing the moment-to-moment interactions in a game. Chapter 4 described three different workflows for game development to answer the second sub-question *"how can game feel design support get implemented in existing workflows?" (RQ2)* Addressing the implementation aspects of supporting game feel design by describing theoretical and practical examples of extending workflows. Identifying and presenting different approaches to supporting game feel design opens the possibility for broader adoption of game feel design tools in existing workflows. Chapter 5 answers the question *"what is the impact of design tools on the game feel design process?" (RQ3)* By evaluating user testing, first-hand interface evaluation, and comparing Squeezer to typical game feel design approaches, to understand the impact of using a design tool like Squeezer. The evaluation highlights the potential support design tools can provide to game feel design. Lastly, Chapter 6 reviews and analyses the impact of adding juice design to an AGD system and reflects on Squeezer usage to answer the research question *"how does the addition of game feel design tools alter the qualities of games?" (RQ4)*

The results of adding juice to games generated by an AGD are in line with the conclusions of similar studies with adding different amounts of juice to games. They indicate that adding some juice effects enhances the perceived game quality, over no effects or extreme effects. Tools like Squeezer allow designers to explore the space of possibilities concerning juice effects to a more considerable degree, making it more likely they find the best-suited effects to amplify events most desirably. Additionally, Squeezer allows complex and unlikely combinations to be tested quickly and easily. These combinations can inform designers about interesting game mechanics that follow from experimentation with juice effects.

This project has undoubtedly partially answered the main research question *how can design tools support game feel design?* First, through a thorough description of game feel design and its role in game design and relation to other subjects. Second, by highlighting potential ways to support game feel designs in existing workflows and through the description, implementation, and release of the juice design tool Squeezer. Third, by evaluating the impact of game feel design tools like Squeezer on the design process. Finally, by reflecting on how tools supporting game feel design alter the qualities of games and how juice effects specifically impact the perception of generated games.

While much ground has been covered in this project, the work has primarily outlined a new and exciting research area in the space between game design, mixed-initiative and computational creativity.

Supporting game feel design through tools remains an open research area, and in the pursuit of answering the questions in this project, many new ideas and questions have emerged. In the following section, these ideas have been outlined and suggested as future research projects, building on or related to the work presented in this dissertation.

7.2 Future Work

This Ph.D. project has presented work on building and evaluating tools that support the game feel design process. Along the way, questions and ideas emerged out of scope for this thesis. However, it is worth presenting these related ideas and potential directions for future research. This section will present several ideas both partially and closely related to the topic of this thesis.

7.2.1 Practice-Based Game Design Research

In this project, I utilized a practice-led approach using software as the "material" to explore the main research question *how can* ***design tools*** *support* ***game feel design****?* Combining examples of common practices, widely used tools, and experience to explore a design process and build tools from a designer's point of view. This approach could be defined as "practice-based game design research." I believe this approach is essential in bridging the gap between research and practice. Several researchers are exploring this way, but we lack clear definitions and method descriptions. I am fortunate that reviewers and peers recognized that game feel and juice were largely unexplored and important within mixed-initiative and computational creativity research. However, without the ability to point towards a clearly defined method and evaluation, other practice-led researchers might have a hard time explaining why their research matters.

At its core, this project investigated an area of game design that has been underserved in research—asking how the design process can be improved through technology while keeping the designer in mind. This approach differs from improving algorithms, strategies, and digital assistants for well-researched problems like level design, game logic, and rules. Though it is important for talented researchers to explore how software can assist with game design [126]; How to improve the use of procedural content generation in game design through tools [190]; How computational creativity can be improved through co-evolutionary strategies [41]; Or how to create general approaches to level generation [111]. I hope future experimental design talent will view game design less as a set of optimization problems for finding new and improved solutions to solve the most easily verifiable of these problems. Break out of these areas and explore the wide range of intriguing design subjects that are harder to evaluate instead. Discover why certain game design topics are less researched and how to support those topics. View the topic from a designer's perspective with a willingness to push boundaries through experimental design. Suppose the work presented in this project inspires and allows just one future experimental designer to research such a topic. In that case, it might just open a new research area for others to explore and optimize. In a similar vein, I hope that the topic of game feel design will be explored and optimized as extensively as level design and game rules have in the past.

However, practice-based design research is a niche within design research, in which game design research is also technically a niche. Practice-based game design research as an approach

would be even more of a niche, and as such, it would require strong voices within the topic to describe experimental methodologies that fit. In practice-based design research, the subject of experimentation is often manipulating a material. However, within game design research, the material is either the player's mental model of the rules or, in the case of video games, it is often the software governing those rules. Further, by extending these ideas into video game design, the "material" is the software that designers use to design the experience the player perceives in their mind. The definition and clarification of this research approach I leave for future research.

7.2.2 Squeezer

Squeezer arose from a desire to make game prototyping tools include more options for game feel design. It has already served to shine a light on the challenges of adding juice effects in automated game design systems [93]. Similarly, a system like Squeezer could be integrated into systems like Ludii or VGDL (GVGAI), or entirely new prototyping frameworks, that build on description language ideas and includes game feel design into those descriptions. An interesting and unexplored topic in computational intelligence research is evaluating how a game feels to play. While evaluation of games and game rules can be done using AI agents, most AI agents do not complain if the game they play feels terrible. *What would happen if AI agents had the option to decide not to play a game*[5]*, or rate it based on the aesthetics of game feel?*

Creating better interfaces and assistive options for Squeezer presents a new and exciting field of research. Not only is there still much work left in making Squeezer more user-friendly, but our work only scratched the surface of what is possible within interactive evolution. There are several other PCG options to explore based on machine learning [112, 206] and GANs [49, 83]. Three different design elements are outlined in our survey paper [155], and Squeezer mainly focused on amplification and the polishing task of juicing the game. Expanding the tool or creating new tools that tackle the polishing tasks of streamlining and tuning are interesting areas for new research to explore.

Fantasy Consoles & Game Generators

From the early onset of this project, supporting design exploration of game feel design has been a central idea. Design exploration that pushes the boundaries of games occurs most frequently during prototyping, particularly during game jamming. Game jamming is mostly a short-form version of getting from an idea to what is known as a "vertical slice". A vertical slice is a functional and relatively accurate representation of what a full game would look and feel like to play. The vertical slice will usually contain a limited subset of the game's potential content. In particular, jam games often avoid implementing extensive platform integration, and the play experience targets the ideal user (leaving out accessibility features, difficulty modes, and similar options). In Chapter 4, the idea of extending a game description language like VGDL with

[5]Similar but not identical to the iconic scene from the movie Wargames (1993).

support for game feel design was explored. The idea was to allow game feel design to exploration through a description language, similar to how puzzle design can be explored using a description language like Puzzlescript[6]. While this project did not lead to a game description language with game feel support, Squeezer does present a visual description language for juice effect design. Providing game feel design support on platforms for design exploration, and game jams is a way to get closer to this goal. Porting[7] Squeezer to platforms that could potentially take advantage of it and be used for game jamming and design explorations has much future potential. In relation to this project a lightweight version of Squeezer[8] was written in lua[9] as a proof-of-concept for the Pico-8 fantasy[10]. Pico-8 is often used as a platform for game prototypes and during game jams because of the development speed and the fact that the console has a built-in editor functionality for sprite, level, music, and sound effect. However, like many other game engines and frameworks, the system is missing tools for tweening animations and various other effect types. The pico-8 version of Squeezer allows coding, triggering, and executing effect sequences similar to the Unity version. The system's many simplifications allow it to drive effect sequences more flexibly, allowing it to drive game mechanics and animations as well. However, there is no user interface (UI) for editing effects and, currently, no easy way to save, load, and share these effects among projects. The Pico-8 platform presents a great opportunity for exploring the best implementations of such interfaces for supporting effect design in a constrained format. The Pico-8 interface is generally minimalist at only 128x128 pixels in 16 colors, with size limits to code and assets. These requirements force the UI implementation to be concise and well thought out and could very well feed into other versions of Squeezer and their UI design. The Lua code for Squeezer (missing UI, generator, and interactive evolution) is only around 1000 code lines. The short implementation makes it very manageable, showing researchers and practitioners alike that effect sequence tools can be implemented in their systems with reasonable efforts. An open-source implementation of the Pico-8 Squeezer has been released, allowing hobbyists and professionals to bring it into new projects and explore how they can use effect prototyping in their projects or explore new applications for this type of tool. However, this version of Squeezer could also serve as a reference implementation for game generation frameworks in future research.

Mixed-Initiative Tools

There is still much to explore by asking more questions in this area, such as "how can we expand the use of mixed-initiative tools to more areas in game development?" Looking into the many different roles, tasks, and responsibilities in game development, especially by gathering insights from practitioners in the field. I drew on my own experience in game development for these insights. However, others might find new areas by looking at ethnographic studies of game

[6] See examples on https://puzzlescript.net/

[7] Porting is the act of rewriting software to work in a new programming language or on a new platform.

[8] https://pyjamads.itch.io/squeezer-pico8/

[9] https://lua.org/

[10] https://www.lexaloffle.com/pico-8.php

companies or developers to build tools that assist developers in realizing, testing, and releasing the games they want. The field is vast and connected to many other disciplines; as exemplified by the tiny corner of game design we looked at in [155] (see also Figure 3.1). Thinking further along these lines raises questions like "How can we assist game designers more through the use of mixed-initiative tools?" and thinking about new technologies and workflows that could improve the way we interact with mixed-initiative tools. We envision futures where AI agents work simultaneously with human designers in the same online repository[11] or directly on the same computer.

Novelty Search and Tree Edit Distance

Improving the interactive evolution UI and algorithms built into Squeezer is another direction that could be interesting to explore. Evolutionary algorithms can still get stuck in local optima even with dramatic mutations. To counteract this an experimental implementation of novelty search [123, 124] is included in the Squeezer source code, with the purpose of providing better quality diversity [159]. While it was not part of the user testing, preliminary experimentation of combining a Tree Edit Distance[12] algorithm and novelty search seemed to provide a more diverse set of effect sequences for the user to choose from. We then experimented with using Tree Edit Distance as a novelty parameter, making the interactive evolution select only the most novel offspring of a generated population to display for the user. We reasoned that a diversified exploration of effect sequences would be more interesting for the user. Unfortunately, our initial implementation of novelty search was very slow, and we never had it tested by users. It remains an item for future research.

7.2.3 More Game Design Topics to Explore

Many elements in game development have yet to get as much attention as level generation and rule generation and could readily become topics for future research. Just by listing various assets categories, there are certainly elements that remain unaddressed or relatively unexplored. Looking at the games themselves assets like textures, models, animations, skeletal-rigs, sound effects, music, shaders, game mechanics, code, rules/goals, controls, visual effects, post-processing effects, game object types, camera controls, audio filters (and mixing), narrative text and storylines come to mind. However, this only covers the games themselves; on top of that, various assets for different storefronts are needed, such as trailers, screenshots, or images that explain the game and draw people in, description texts, app icons, different types of promotional material for conferences and other marketing material.

[11] Repository refers to a project in a version control system, such as Git, Subversion or Mercurial.
[12] http://tree-edit-distance.dbresearch.uni-salzburg.at/

7.2.4 Automated Game Design

Automated game design is an area where exploring different game expressions and how to implement those are very important to move the generated games from technical or curious prototypes into games that can challenge or push the status quo. Both implementation and evaluation aspects of adding juice design into an AGD process have multiple open questions. Implementation-wise, there are multiple questions about how and where juice design should happen in the process. Is juice design a part of the primary game design process, or should it be a separate process like in our paper [93]? What sort of data can we gather from AI agent playtesting, and how can we use that data to improve the game mechanics and the game feel of the generated games?

Furthermore, how do we best evaluate the generated effects, and how well they match the game mechanics? What algorithmic options are there for this type of evaluation, and how well do they work? Should evaluation be left to playtesters, and how can we make AI agents evaluate the coherence between game mechanics and effects and evaluate the overall game feel? Does this part of the process need to be automated, or can we make an AGD designer agent that collaborates with human designers in a development process, as Cook suggests in [42]?

7.2.5 Juice as Sense Replacement

A new branch of research that we identified while writing our game feel survey [155] is the idea that juice or design elements related to amplification can be viewed as sense replacements. To build the intended game feel, Brown notes, "you're not juicing your game - you're actually picking a feeling that your game should communicate and juicing that feeling" [18]. Likewise, juice can amplify the physicality of a game, thereby replacing senses or experiences through the available outputs. Most games are played on a flat-screen, using a keyboard, mouse, controller, or touch input. Although controllers and touch devices can provide haptic feedback, the senses engaged while playing games are usually limited to sight, hearing, and partially touch and maybe spatial awareness, leaving additional signals such as smell, taste, temperature, pain, balance, and more unaccounted for. Even virtual reality (VR) games only engage around two to three additional senses and can easily confuse these (hence the tendency to feel sick if a few things are off in VR games). We believe that juice can supply some of this lacking information to the brain. There is a broad consensus that brains have a remarkable ability to adapt to novel stimuli in neuroscience [197]. An example is experiments using electrodes on the tongues of participants with visual input, resulting in blind people being able to navigate their surroundings [74]. In VR, altering the tracked motions of a virtual hand can change the perceived weight of objects [174]. We believe that the reason the twelve principals of animation [217] work so well is that they provide context the brain might otherwise have gathered through senses other than sight. By amplifying certain visual, audible, or haptic aspects of games and other media, the brain can contribute additional information about the world it perceives and acts within. In the case of video games, if a designer wants something to feel more powerful,

they could add some bass boost to the sound effect [140], as Nijman anecdotally describes. However, two (or more) of the primary senses that get replaced in games are smell and taste. Smells will often be replaced visually by clouds or fumes coming off an element in a game. While taste is often replaced with textual descriptions, it can also be replaced by the gestures of other characters in games. In that sense, taste is often perceived through a character experiencing it, much like the principles of animation convey the illusion of life and the environment the characters experience.

7.2.6 Exploring How Juice Impacts AI Performance

Game AI scholars and competition developers' primary goal seems to be to create and solve the most complex logic problems that can be explored with simplified game simulations. Many frameworks for AI competitions severely lack juice effects and event signification. Event signification which is the primary goal of juice effects, is the part of game feel concerned with communicating the intended experience to the player. As juice effects are meant to amplify the game events to emphasize their importance, it stands to reason that AI algorithms able to perceive such effects could potentially benefit from them. Computer vision for AI algorithms is often limited by downsampling the screen, and general algorithms' audio perception is almost unheard of (pun intended). It would certainly be interesting to figure out the impact of juice effects on AI learning. Juice effects represent a kind of intrinsic reward for human players, and so it could potentially be used as such in a reinforcement learning [212] system. Furthermore, if such a system could learn to evaluate the intrinsic reward of juice effects, it could also be used in relation to automated game design for effect evaluation.

Chapter 8

References

[1] Oguz A. Acar, Murat Tarakci, and Daan van Knippenberg. "Creativity and Innovation Under Constraints: A Cross-Disciplinary Integrative Review". In: *Journal of Management* 45.1 (Jan. 2019), pp. 96–121. ISSN: 0149-2063. DOI: 10.1177/0149206318805832. URL: https://doi.org/10.1177/0149206318805832 (visited on 02/16/2022).

[2] Jeremy Alessi. *Games Demystified: Super Mario Galaxy*. https://www.gamasutra.com/view/feature/131997/games_demystified_super_mario_.php. Accessed: 2020-10-06T12:46:42Z. 2008. URL: https://www.gamasutra.com/view/feature/131997/games_demystified_super_mario_.php (visited on 10/06/2020).

[3] Heather Alexander. *The Quiet Importance Of Idle Animations*. https://kotaku.com/the-quiet-importance-of-idle-animations-1834564079. Accessed: 2020-08-24T09:09:29Z. 2019. URL: https://kotaku.com/the-quiet-importance-of-idle-animations-1834564079 (visited on 08/24/2020).

[4] Anna Anthropy and Naomi Clark. *A Game Design Vocabulary: Exploring the Foundational Principles behind Good Game Design*. 1st. Addison-Wesley Professional, 2014. ISBN: 0-321-88692-5.

[5] Nick Babich. *The UI/UX Design of Progress Indicator [Trends + Examples]*. https://usersnap.com/blog/progress-indicators/. Accessed: 2020-10-21T10:06:14Z. May 2019. URL: https://usersnap.com/blog/progress-indicators/ (visited on 10/21/2020).

[6] Cory Barlog. *Why Kratos' Axe Feels SO Powerful — Game Mechanics Explained*. https://www.youtube.com/watch?v=zpr-EE2In1M&ab_channel=Polygon. Accessed: 2020-10-06T13:00:20Z. May 2018. URL: https://www.youtube.com/watch?v=zpr-EE2In1M&ab_channel=Polygon (visited on 10/06/2020).

[7] Federico Barrios et al. "Variations of the Similarity Function of TextRank for Automated Summarization". In: *arXiv:1602.03606 [cs]* (Feb. 2016). arXiv: 1602.03606 [cs]. URL: http://arxiv.org/abs/1602.03606 (visited on 01/05/2022).

[8] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* 47 (June 2013), pp. 253–279. ISSN: 1076-9757. DOI: 10.1613/jair.3912. arXiv: 1207.4708. URL: http://arxiv.org/abs/1207.4708 (visited on 03/24/2019).

[9] Nicolae Berbece. *Game Feel: Why Your Death Animation Sucks*. https://www.gdcvault.com/play/1022759/Game-Feel-Why-Your-Death. Accessed: 2020-05-11T08:18:32Z. San Francisco, CA, 2015. URL: https://www.gdcvault.com/play/1022759/Game-Feel-Why-Your-Death (visited on 05/11/2020).

[10] Debosmita Bhaumik, Ahmed Khalifa, and Julian Togelius. "Lode Encoder: AI-constrained Co-Creativity". In: *2021 IEEE Conference on Games (CoG)*. Copenhagen, Denmark: IEEE, Aug. 2021, pp. 01–08. ISBN: 978-1-66543-886-5. DOI: 10.1109/CoG52621.2021.9619009. URL: https://ieeexplore.ieee.org/document/9619009/ (visited on 02/16/2022).

[11] ken birdwell. *The Cabal: Valve's Design Process For Creating Half-Life*. https://www.gamasutra.com/view/feature/131815/the_cabal_valves_design_process_.php. Accessed: 2020-08-24T09:05:58Z. 1999. URL: https://www.gamasutra.com/view/feature/131815/the_cabal_valves_design_process_.php (visited on 08/24/2020).

[12] Blizzard Entertainment. *Hearthstone*. 2014.

[13] Margaret A. Boden. "Chapter 9 - Creativity". In: *Artificial Intelligence*. Ed. by Margaret A. Boden. Handbook of Perception and Cognition. San Diego: Academic Press, Jan. 1996, pp. 267–291. ISBN: 978-0-12-161964-0. DOI: 10.1016/B978-012161964-0/50011-X. URL: https://www.sciencedirect.com/science/article/pii/B978012161964050011X (visited on 02/17/2022).

[14] Margaret A. Boden. "Creativity and Artificial Intelligence". In: *Artificial Intelligence*. Artificial Intelligence 40 Years Later 103.1 (Aug. 1998), pp. 347–356. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(98)00055-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370298000551 (visited on 02/17/2022).

[15] Margaret A. Boden, ed. *Dimensions of Creativity*. Cambridge, MA, USA: A Bradford Book, June 1994. ISBN: 978-0-262-02368-9.

[16] Margaret A. Boden. *The Creative Mind: Myths and Mechanisms*. Second. London: Routledge, Sept. 2003. ISBN: 978-0-203-50852-7. DOI: 10.4324/9780203508527.

[17] Eva Brandt et al. *XLAB*. Jan. 2011. ISBN: 978-87-92016-24-9.

[18] Lisa Brown. *The Nuance of Juice*. https://www.youtube.com/watch?v=qtgWBUIOjK4. Accessed: 2020-04-17T11:40:19Z. Vector, 2016. URL: https://www.youtube.com/watch?v=qtgWBUIOjK4 (visited on 04/17/2020).

[19] Mark Brown. *Secrets of Game Feel and Juice*. https://www.youtube.com/watch?v=216_5nu4aVQ. Accessed: 2020-04-17T11:52:14Z. 2015. URL: https://www.youtube.com/watch?v=216_5nu4aVQ (visited on 04/17/2020).

[20] Mark Brown. *Why Does Celeste Feel So Good to Play? — Game Maker's Toolkit*. https://www.youtube.com/watch?v=yorTG9at90g. Accessed: 2020-04-17T13:21:54Z. 2019. URL: https://www.youtube.com/watch?v=yorTG9at90g (visited on 04/17/2020).

[21] Cameron Browne. "A Class Grammar for General Games". In: *Computers and Games*. Ed. by Aske Plaat, Walter Kosters, and Jaap van den Herik. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 167–182. ISBN: 978-3-319-50935-8. DOI: 10.1007/978-3-319-50935-8_16.

[22] Cameron Browne and Frederic Maire. "Evolutionary Game Design". In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.1 (Mar. 2010), pp. 1–16. ISSN: 1943-0698. DOI: 10.1109/TCIAIG.2010.2041928.

[23] Paolo Burelli. "Virtual Cinematography in Games: Investigating the Impact on Player Experience". In: *International Conference On The Foundations of Digital Games*. Chania, Greece, May 2013. DOI: 10.13140/2.1.4643.1045.

[24] Kitty Calis et al. *Disc Room*. 2020.

[25] Christian Carlsson and Axel Pelling. "Designing Spectator Interfaces for Competitive Video Games". MA thesis. Gothenburg, Sweden: Chalmers University of Technology, 2015. URL: http://publications.lib.chalmers.se/records/fulltext/224247/224247.pdf.

[26] Bay-Wei Chang and David Ungar. "Animation: From Cartoons to the User Interface". In: *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology - UIST '93*. Atlanta, Georgia, United States: ACM Press, 1993, pp. 45–55. ISBN: 978-0-89791-628-8. DOI: 10.1145/168642.168647. URL: http://portal.acm.org/citation.cfm?doid=168642.168647 (visited on 05/11/2020).

[27] Megan Charity, Ahmed Khalifa, and Julian Togelius. "Baba Is Y'all: Collaborative Mixed-Initiative Level Design". In: *2020 IEEE Conference on Games (CoG)*. Osaka, Japan: IEEE, Aug. 2020, pp. 542–549. ISBN: 978-1-72814-533-4. DOI: 10.1109/CoG47356.2020.9231807. URL: https://ieeexplore.ieee.org/document/9231807/ (visited on 02/16/2022).

[28] Jenova Chen. "Flow in Games (and Everything Else)". In: *Communications of the ACM* 50.4 (Apr. 2007), p. 31. ISSN: 00010782. DOI: 10.1145/1232743.1232769. URL: http://portal.acm.org/citation.cfm?doid=1232743.1232769 (visited on 09/17/2019).

[29] Marc Christie, Patrick Olivier, and Jean-Marie Normand. "Camera Control in Computer Graphics". In: *Computer Graphics Forum* 27.8 (Dec. 2008), pp. 2197–2218. ISSN: 01677055, 14678659. DOI: 10.1111/j.1467-8659.2008.01181.x. URL: http://doi.wiley.com/10.1111/j.1467-8659.2008.01181.x (visited on 05/04/2020).

[30] David Ciccoricco. "Narrative, Cognition, and the Flow of Mirror's Edge:" in: *Games and Culture* (July 2012). DOI: 10.1177/1555412012454223. URL: https://journals.sagepub.com/doi/10.1177/1555412012454223 (visited on 08/24/2020).

[31] Ryan Clark. *The Clark Tank*. 2019.

[32] Jeff Clune et al. *EndlessForms.Com - Design Objects with Evolution and 3D Print Them!* http://endlessforms.com/. Accessed: 2020-11-17T17:22:46Z. 2012. URL: http://endlessforms.com/ (visited on 11/17/2020).

[33] Simon Colton. "Creativity Versus the Perception of Creativity in Computational System". In: (2008), p. 7.

[34] Simon Colton. "The Painting Fool: Stories from Building an Automated Painter". In: *Computers and Creativity*. Ed. by Jon McCormack and Mark d'Inverno. Berlin, Heidelberg: Springer, 2012, pp. 3–38. ISBN: 978-3-642-31727-9. DOI: 10.1007/978-3-642-31727-9_1. URL: https://doi.org/10.1007/978-3-642-31727-9_1 (visited on 02/26/2022).

[35] Simon Colton, Alison Pease, and John Charnley. "Computational Creativity Theory: The FACE and IDEA Descriptive Models". In: *Proceedings of the Second International Conference on Computational Creativity*. 2011, p. 6.

[36] Simon Colton and Geraint A Wiggins. "Computational Creativity: The Final Frontier?" In: *The Final Frontier* (2012), p. 6.

[37] Simon Colton et al. *The Painting Fool Sees! New Projects with the Automated Painter*. July 2015.

[38] Jared Cone. *It IS Rocket Science! The Physics of 'Rocket League' Detailed*. 2018. URL: https://www.gdcvault.com/play/1025341/It-IS-Rocket-Science-The (visited on 10/06/2020).

[39] M. Cook, S. Colton, and J. Gow. "The ANGELINA Videogame Design System—Part I". In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.2 (June 2017), pp. 192–203. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2016.2520256.

[40] M. Cook, S. Colton, and J. Gow. "The ANGELINA Videogame Design System—Part II". In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.3 (Sept. 2017), pp. 254–266. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2016.2520305.

[41] Michael Cook. "Co-Operative Coevolution for Computational Creativity: A Case Study In Videogame Design". PhD thesis. 2015.

[42] Michael Cook. "Software Engineering For Automated Game Design". In: *2020 IEEE Conference on Games (CoG)*. Aug. 2020, pp. 487–494. DOI: 10.1109/CoG47356.2020.9231750.

[43] Michael Cook, Simon Colton, and Jeremy Gow. "Automating Game Design In Three Dimensions". In: *AISB Symposium on AI and Games*. AISB, Apr. 2014, pp. 1–4. URL: http://research.gold.ac.uk/id/eprint/17354/ (visited on 06/24/2021).

[44] Michael Cook, Simon Colton, and Alison Pease. "Aesthetic Considerations for Automated Platformer Design". In: (2012), p. 6.

[45] Michael Cook, Jeremy Gow, and Simon Colton. "Danesh: Helping Bridge The Gap Between Procedural Generators And Their Output". In: (2016), p. 16.

[46] Michael Cook and Gillian Smith. "Formalizing Non-Formalism: Breaking the Rules of Automated Game Design". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015)*. FDG, 2015, p. 5.

[47] *Cooldowns Can Be Used to Balance Games*. https://game-design-snacks.fandom.com/wiki/Cooldowns_can_be_used_to_balance_games. Accessed: 2020-10-21T10:03:10Z. URL: https://game-design-snacks.fandom.com/wiki/Cooldowns_can_be_used_to_balance_games (visited on 10/21/2020).

[48] Joel Couture. *What Makes a Great Idle Animation? Devs Share Their Favorites*. /view/news/318163/What_makes_a_great_idle_animation_Devs_share_their_favorites.php. Accessed: 2020-08-24T09:13:58Z. 2018. URL: /view/news/318163/What_makes_a_great_idle_animation_Devs_share_their_favorites.php (visited on 08/24/2020).

[49] Antonia Creswell et al. “Generative Adversarial Networks: An Overview”. In: *IEEE Signal Processing Magazine* 35.1 (Jan. 2018), pp. 53–65. ISSN: 1558-0792. DOI: 10.1109/MSP.2017.2765202.

[50] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. New York: Harper and Row, 1990.

[51] Gustav Dahl and Martin Kraus. “Measuring How Game Feel Is Influenced by the Player Avatar’s Acceleration and Deceleration: Using a 2D Platformer to Describe Players’ Perception of Controls in Videogames”. In: *Proceedings of the 19th International Academic Mindtrek Conference on - AcademicMindTrek ’15*. Tampere, Finland: ACM Press, 2015, pp. 41–46. ISBN: 978-1-4503-3948-3. DOI: 10.1145/2818187.2818275. URL: http://dl.acm.org/citation.cfm?doid=2818187.2818275 (visited on 09/11/2019).

[52] Derek Daniels. *Why Some Games Feel Better than Others - Part 3*. Blog Post. Mar. 2007.

[53] Edward de Bono. “Lateral Thinking; Creativity Step by Step”. In: (1970).

[54] Sebastian Deterding. “The Lens of Intrinsic Skill Atoms: A Method for Gameful Design”. In: *Human–Computer Interaction* 30.3-4 (May 2015), pp. 294–335. ISSN: 0737-0024. DOI: 10.1080/07370024.2014.993471. URL: https://doi.org/10.1080/07370024.2014.993471 (visited on 04/17/2020).

[55] Sebastian Deterding et al. “Mixed-Initiative Creative Interfaces”. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. Denver Colorado USA: ACM, May 2017, pp. 628–635. ISBN: 978-1-4503-4656-6. DOI: 10.1145/3027063.3027072. URL: https://dl.acm.org/doi/10.1145/3027063.3027072 (visited on 10/16/2021).

[56] DICE. *Mirror’s Edge*. DICE. 2008.

[57] Lars Doucet. *Oil It or Spoil It!* https://www.fortressofdoors.com/oil-it-or-spoil-it/. Accessed: 2019-10-27T12:44:44Z. Aug. 2016. URL: https://www.fortressofdoors.com/oil-it-or-spoil-it/ (visited on 10/27/2019).

[58] Marie Ehrndal. “A Holistic Approach to Designing for a Specific Aesthetic Experience in Digital Games”. MA thesis. Malmö, Sweden: Malmö högskola, 2012. URL: http://muep.mau.se/handle/2043/13942 (visited on 04/17/2020).

[59] A. Eiben and Jim Smith. *Introduction To Evolutionary Computing*. Vol. 45. Jan. 2003. ISBN: 978-3-642-07285-7. DOI: 10.1007/978-3-662-05094-1.

[60] A. E. Eiben and J. E. Smith. “Interactive Evolutionary Algorithms”. In: *Introduction to Evolutionary Computing*. Ed. by A.E. Eiben and J.E. Smith. Natural Computing Series. Berlin, Heidelberg: Springer, 2015, pp. 215–222. ISBN: 978-3-662-44874-8. DOI: 10.1007/978-3-662-44874-8_14. URL: https://doi.org/10.1007/978-3-662-44874-8_14 (visited on 10/19/2021).

[61] Cosmic Engineers. *Cosmic Express*. 2017. URL: https://cosmicexpressgame.com/ (visited on 10/18/2021).

[62] Martin Fasterholdt, Martin Pichlmair, and Christoffer Holmgård. "You Say Jump, I Say How High? Operationalising the Game Feel of Jumping". In: *Proceedings of the First International Joint Conference of DiGRA and FDG*. Dundee, Scotland: Digital Games Research Association and Society for the Advancement of the Science of Digital Games, 2016. ISBN: ISSN 2342-9666. URL: http://www.digra.org/wp-content/uploads/digital-library/paper_248.pdf.

[63] Renaud Forestié. *Best Practices for Fast Game Design in Unity*. https://www.youtube.com/watch?v=NU29QKag8a0. Accessed: 2020-04-17. Unite LA 2018, 2018. URL: https://www.youtube.com/watch?v=NU29QKag8a0 (visited on 04/17/2020).

[64] Renaud Forestié. *How to Design with Feedback and Game Feel in Mind - Shake It 'til You Make It*. https://www.youtube.com/watch?v=yCKI9T3sSv0. Accessed: 2020-04-17. Unite Copenhagen 2019, 2019. URL: https://www.youtube.com/watch?v=yCKI9T3sSv0 (visited on 04/17/2020).

[65] Giorgio Franceschelli and Mirco Musolesi. "Creativity and Machine Learning: A Survey". In: *arXiv:2104.02726 [cs]* (Apr. 2021). arXiv: 2104.02726 [cs]. URL: http://arxiv.org/abs/2104.02726 (visited on 10/19/2021).

[66] Karmen Franinović, Karmen Franinovic, and Stefania Serafin. *Sonic Interaction Design*. MIT Press, 2013. ISBN: 978-0-262-01868-5.

[67] Matthew Fuller, ed. *Software Studies: A Lexicon*. Leonardo Books. Cambridge, Mass: MIT Press, 2008. ISBN: 978-0-262-06274-9.

[68] Tracy Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 3rd. A K Peters/CRC Press, Apr. 2014.

[69] Zach Gage. *Building Games That Can Be Understood at a Glance*. https://www.youtube.com/watch?v=YISKcRDcDJg&ab_channel=GDC. Accessed: 2020-10-06T14:28:07Z. 2018. URL: https://www.youtube.com/watch?v=YISKcRDcDJg&ab_channel=GDC (visited on 10/06/2020).

[70] *Game Accessibility Guidelines — A Straightforward Reference for Inclusive Game Design*. http://gameaccessibilityguidelines.com/. Accessed: 2020-10-21T09:57:56Z. URL: http://gameaccessibilityguidelines.com/ (visited on 10/21/2020).

[71] "Game Design Workshop_ A Playcentric Approa - Tracy Fullerton - 2nd". In: (Apr. 2015), pp. 1–491.

[72] Troy Gilbert. *Movement Mechanics*. https://troygilbert.com/deconstructing-zelda/movement-mechanics/. Accessed: 2020-04-17T13:52:31Z. 2012. URL: https://troygilbert.com/deconstructing-zelda/movement-mechanics/ (visited on 04/17/2020).

[73] Nathan Gitter. *Building Fluid Interfaces*. https://medium.com/@nathangitter/building-fluid-interfaces-ios-swift-9732bb934bf5. Accessed: 2020-10-06T14:39:14Z. Aug. 2018. URL: https://medium.com/@nathangitter/building-fluid-interfaces-ios-swift-9732bb934bf5 (visited on 10/06/2020).

[74] Patricia Grant et al. "The Functional Performance of the BrainPort V100 Device in Persons Who Are Profoundly Blind". In: *Journal of Visual Impairment & Blindness* 110.2 (Mar. 2016), pp. 77–88. ISSN: 0145-482X. DOI: 10.1177/0145482X1611000202. URL: https://doi.org/10.1177/0145482X1611000202 (visited on 09/23/2021).

[75] Kyle Gray et al. *How to Prototype a Game in Under 7 Days*. https://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php. Accessed: 2019-10-27. 2005. URL: https://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php (visited on 10/27/2019).

[76] Saul Greenberg et al. *Sketching User Experiences: The Workbook*. Elsevier, Nov. 2011. ISBN: 978-0-12-381961-1.

[77] Jaime Griesemer. *Design by Numbers: Cooldowns*. https://rewardingplay.com/2012/01/09/design-by-numbers-cooldowns/. Accessed: 2020-10-21T10:05:22Z. Jan. 2012. URL: https://rewardingplay.com/2012/01/09/design-by-numbers-cooldowns/ (visited on 10/21/2020).

[78] Matthew Guzdial et al. "Friend, Collaborator, Student, Manager: How Design of an AI-Driven Game Level Editor Affects Creators". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland Uk: ACM, May 2019, pp. 1–13. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300854. URL: https://dl.acm.org/doi/10.1145/3290605.3300854 (visited on 02/18/2022).

[79] Matthew James Guzdial. "Combinational Machine Learning Creativity". PhD thesis. Georgia Institute of Technology, July 2019. URL: https://smartech.gatech.edu/bitstream/handle/1853/61790/GUZDIAL-DISSERTATION-2019.pdf?sequence=1&isAllowed=y (visited on 12/09/2021).

[80] Mark Haigh-Hutchinson. "Fundamentals of Real-Time Camera Design". In: *GDC'05 talk* (2005), p. 20.

[81] Mark Haigh-Hutchinson. *Real Time Cameras: A Guide for Game Designers and Developers*. 1 edition. San Francisco, Calif. : Oxford: CRC Press, Apr. 2009. ISBN: 978-0-12-311634-5.

[82] Mark Haigh-Hutchinson. *Real-Time Cameras - Navigation and Occlusion*. https://www.gamasutra.com/view/feature/132456/realtime_cameras__navigation_and_.php. Accessed: 2020-06-29T22:03:24Z. 2009. URL: https://www.gamasutra.com/view/feature/132456/realtime_cameras__navigation_and_.php (visited on 06/29/2020).

[83] Andreas Hald et al. "Procedural Content Generation of Puzzle Games Using Conditional Generative Adversarial Networks". In: *International Conference on the Foundations of Digital Games*. Bugibba Malta: ACM, Sept. 2020, pp. 1–9. ISBN: 978-1-4503-8807-8. DOI: 10.1145/3402942.3409601. URL: https://dl.acm.org/doi/10.1145/3402942.3409601 (visited on 10/19/2021).

[84] Kieran Hicks et al. "Good Game Feel: An Empirically Grounded Framework for Juicy Design". In: *Proceedings of the 2018 DiGRA International Conference: The Game Is the Message*. DiGRA, July 2018, p. 17. URL: http://www.digra.org/wp-content/uploads/digital-library/DIGRA_2018_Paper_35.pdf.

[85] Mads Hobye. "Designing for Homo Explorens: Open Social Play in Performative Frames". PhD thesis. Malmö, Sweden: Malmö University, 2014. ISBN: 9789171045379.

[86] Amy K. Hoover et al. "Generating a Complete Multipart Musical Composition from a Single Monophonic Melody with Functional Scaffolding". In: *ICCC*. Vol. PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON COMPUTATIONAL CREATIVITY. Proceedings of the Third International Conference on Computational Creativity: ICCC, 2012, pp. 111–119.

[87] Johan Huizinga. *Homo Ludens: Proeve Fleener Bepaling van Het Spel-Element Der Cultuur*. 1938.

[88] Robin Hunicke. *Loving Your Player With Juicy Feedback*. dConstruct 2009, 2009. URL: http://2009.dconstruct.org/podcast/juicyfeedback (visited on 04/17/2020).

[89] Robin Hunicke. "The Case for Dynamic Difficulty Adjustment in Games". In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology - ACE '05*. Valencia, Spain: ACM Press, 2005, pp. 429–433. ISBN: 978-1-59593-110-8. DOI: 10.1145/1178477.1178573. URL: http://portal.acm.org/citation.cfm?doid=1178477.1178573 (visited on 01/10/2022).

[90] Sonic Hurricane. *Impact Freeze*. https://sonichurricane.com/?p=1043. Forum Post. Jan. 2010. URL: https://sonichurricane.com/?p=1043.

[91] Tommi Ilmonen and Janne Kontkanen. "The Second Order Particle System". In: *Journal of WSCG* 11.1 (2003). ISSN: 1213-6972.

[92] Thomas Jakobsen. "Advanced Character Physics". In: *In Proceedings of the Game Developers Conference 2001*. 2001, p. 19.

[93] Mads Johansen and Michael Cook. "Challenges in Generating Juice Effects For Automatically Designed Games". In: *Proceedings of The 17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Online: AAAI Press, Oct. 2021, p. 8. URL: https://sites.google.com/view/aiide2021/papers.

[94] Mads Johansen, Martin Pichlmair, and Sebastian Risi. "Squeezer - A Mixed-Initiative Tool for Designing Juice Effects". In: *Proceedings of the Foundations of Digital Games Conference (FDG 2021)*. Online, 2021, p. 11.

[95] Mads Johansen, Martin Pichlmair, and Sebastian Risi. "Squeezer - A Tool for Designing Juicy Effects". In: *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play*. CHI PLAY '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 282–286. ISBN: 978-1-4503-7587-0. DOI: 10.1145/3383668.3419862. URL: https://doi.org/10.1145/3383668.3419862 (visited on 11/03/2020).

[96] Mads Johansen, Martin Pichlmair, and Sebastian Risi. "Video Game Description Language Environment for Unity Machine Learning Agents". In: *2019 IEEE Conference on Games (CoG)*. Vol. 2019 IEEE Conference on Games (CoG). IEEE, Aug. 2019, pp. 1–8. DOI: 10.1109/CIG.2019.8848072.

[97] J. Johnson and G. Engelbeck. "Modes Survey Results". In: *SIGCHI Bull.* 20.4 (Apr. 1989), pp. 38–50. ISSN: 0736-6906. DOI: 10.1145/67243.67248. URL: https://doi.org/10.1145/67243.67248.

[98] Martin Jonasson and Petri Purho. *Juice It or Lose It*. https://www.youtube.com/watch?v=Fy0aCDmgnxg. Accessed: 2019-02-27. Nordic Game Conference 2012, 2012. URL: https://www.youtube.com/watch?v=Fy0aCDmgnxg (visited on 02/27/2019).

[99] Anna Jordanous. "Evaluating Computational Creativity: A Standardised Procedure for Evaluating Creative Systems and Its Application". PhD thesis. University of Sussex, 2012. URL: http://sro.sussex.ac.uk/44741/1/Jordanous,_Anna_Katerina.pdf (visited on 02/26/2022).

[100] Arthur Juliani et al. "Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning". In: *arXiv:1902.01378 [cs]* (Feb. 2019). arXiv: 1902.01378 [cs]. URL: http://arxiv.org/abs/1902.01378 (visited on 02/11/2019).

[101] Arthur Juliani et al. "Unity: A General Platform for Intelligent Agents". In: *arXiv:1809.02627 [cs, stat]* (Sept. 2018). arXiv: 1809.02627 [cs, stat]. URL: http://arxiv.org/abs/1809.02627 (visited on 01/07/2019).

[102] Jesper Juul. *A Casual Revolution — The MIT Press*. MIT Press, 2009. URL: https://mitpress.mit.edu/books/casual-revolution (visited on 04/17/2020).

[103] Jesper Juul. *Half-Real: Video Games between Real Rules and Fictional Worlds*. MIT Press, 2005. ISBN: 978-0-262-28413-4.

[104] Jesper Juul and Jason Scott Begy. "Good Feedback for Bad Players? A Preliminary Study of 'Juicy' Interface Feedback". In: *Proceedings of First Joint FDG/DiGRA Conference*. Vol. Proceedings of first joint FDG/DiGRA Conference. Dundee: DiGRA, 2016, p. 2. URL: https://www.jesperjuul.net/text/juiciness.pdf.

[105] Dominic Kao. "The Effects of Juiciness in an Action RPG". In: *Entertainment Computing* 34 (Feb. 2020), p. 100359. DOI: 10.1016/j.entcom.2020.100359.

[106] Dominic Kao and D. Fox Harrell. "Exploring the Impact of Avatar Color on Game Experience in Educational Games". In: *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 1896–1905. ISBN: 978-1-4503-4082-3. DOI: 10.1145/2851581.2892281. URL: https://doi.org/10.1145/2851581.2892281 (visited on 10/06/2020).

[107] Chan Karunamuni, Nathan de Vries, and Marcos Alonso. *Designing Fluid Interfaces - WWDC 2018 - Videos*. https://developer.apple.com/videos/play/wwdc2018/803/. Accessed: 2020-10-06T14:35:46Z. 2018. URL: https://developer.apple.com/videos/play/wwdc2018/803/ (visited on 10/06/2020).

[108] Fares Kayali and Peter Purgathofer. "Two Halves of Play-Simulation versus Abstraction and Transformation in Sports Videogames Design". In: *Eludamos. Journal for Computer Game Culture* 2.1 (2008), pp. 105–127.

[109] Itay Keren. *Gamasutra: Itay Keren's Blog - Scroll Back: The Theory and Practice of Cameras in Side-Scrollers*. https://gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php. Accessed: 2020-08-05T12:23:00Z. 2015. URL: https://gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php (visited on 08/05/2020).

[110] Itay Keren. *Scroll Back: The Theory and Practice of Cameras in Side-Scrollers*. 2015. URL: https://www.gdcvault.com/play/1022243/Scroll-Back-The-Theory-and (visited on 04/17/2020).

[111] Ahmed Khalifa. "General Level Generation". PhD thesis. 2020.

[112] Ahmed Khalifa and Magda Fayek. "Automatic Puzzle Level Generation: A General Approach Using a Description Language". In: (2015), p. 8.

[113] Ahmed Khalifa et al. "General Video Game Rule Generation". In: *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. New York, NY, USA: IEEE, Aug. 2017, pp. 170–177. ISBN: 978-1-5386-3233-8. DOI: 10.1109/CIG.2017.8080431. URL: http://ieeexplore.ieee.org/document/8080431/ (visited on 02/27/2019).

[114] David King. *Principals of UI Design in the World of Warcraft*. https://medium.com/@d.w.king12/principals-of-ui-design-in-the-world-of-warcraft-19e1a33feb61. Accessed: 2020-10-09T20:10:02Z. Dec. 2019. URL: https://medium.com/@d.w.king12/principals-of-ui-design-in-the-world-of-warcraft-19e1a33feb61 (visited on 10/09/2020).

[115] Shringi Kumari, Sebastian Deterding, and Jonathan Freeman. "The Role of Uncertainty in Moment-to-Moment Player Motivation: A Grounded Theory". In: *Proceedings of the Annual Symposium on Computer-Human Interaction in Play - CHI PLAY '19*. Barcelona, Spain: ACM Press, 2019, pp. 351–363. ISBN: 978-1-4503-6688-5. DOI: 10.1145/3311350.3347148. URL: http://dl.acm.org/citation.cfm?doid=3311350.3347148 (visited on 10/24/2019).

[116] Ben Kybartas, Clark Verbrugge, and Jonathan Lessard. "Expressive Range Analysis of a Possible Worlds Driven Emergent Narrative System". In: *Interactive Storytelling*. Ed. by Rebecca Rouse, Hartmut Koenitz, and Mads Haahr. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 473–477. ISBN: 978-3-030-04028-4. DOI: 10.1007/978-3-030-04028-4_54.

[117] Carolyn Lamb, Daniel G. Brown, and Charles L. A. Clarke. "Evaluating Computational Creativity: An Interdisciplinary Tutorial". In: *ACM Computing Surveys* 51.2 (Feb. 2018), 28:1–28:34. ISSN: 0360-0300. DOI: 10.1145/3167476. URL: https://doi.org/10.1145/3167476 (visited on 02/17/2022).

[118] Lasse Juel Larsen. "Collision Thrills: Unpacking the Aesthetics of Action in Computer Games". In: *Journal of Computer Games and Communication* 1.1 (Apr. 2016), pp. 41–52. ISSN: 21481881. DOI: 10.15340/2148188111997. URL: https://www.macroworldpub.com/makale_detay.php?makale_id=92&dergi_id=55#.VyMdGz9NjaY (visited on 04/17/2020).

[119] John Lasseter. "PRINCIPLES OF TRADITIONAL ANIMATION APPLIED TO 3D COMPUTER ANIMATION". In: *Computer Graphics, Volume 21, Number 4, July 1987* (1987), p. 10.

[120] Lutz Latta. *Building a Million-Particle System*. https://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php. Accessed: 2020-10-06T12:53:43Z. 2004. URL: https://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php (visited on 10/06/2020).

[121] Stephen Lavelle. *Flickgame*. 2015. URL: https://www.flickgame.org/ (visited on 10/01/2021).

[122] Adam Lefky and Artem Gindin. *Acceleration Due to Gravity: Super Mario Brothers*. 2007.

[123] Joel Lehman and Kenneth O. Stanley. "Abandoning Objectives: Evolution Through the Search for Novelty Alone". In: *Evolutionary Computation* 19.2 (June 2011), pp. 189–223. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/EVCO_a_00025. URL: https://direct.mit.edu/evco/article/19/2/189-223/1365 (visited on 09/15/2021).

[124] Joel Lehman and Kenneth O. Stanley. "Novelty Search and the Problem with Objectives". In: *Genetic Programming Theory and Practice IX*. Ed. by Rick Riolo, Ekaterina Vladislavleva, and Jason H. Moore. Genetic and Evolutionary Computation. New York, NY: Springer, 2011, pp. 37–56. ISBN: 978-1-4614-1770-5. DOI: 10.1007/978-1-4614-1770-5_3. URL: https://doi.org/10.1007/978-1-4614-1770-5_3 (visited on 10/19/2021).

[125] John Levine et al. "General Video Game Playing". In: *Artificial and Computational Intelligence in Games*. Ed. by Simon M. Lucas et al. Vol. 6. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 77–83. ISBN: 978-3-939897-62-0. DOI: 10.4230/DFU.Vol6.12191.77. URL: http://drops.dagstuhl.de/opus/volltexte/2013/4337.

[126] Antonios Liapis. "Searching for Sentient Design Tools for Game Development". PhD thesis. Copenhagen, Denmark: IT University of Copenhagen, 2014.

[127] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. "Computational Game Creativity". In: *ICCC*. 2014, p. 8.

[128] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. "Sentient Sketchbook: Computer-Aided Game Level Authoring". In: *Foundations of Digital Games*. 2013, p. 8.

[129] Youn-Kyung Lim, Erik Stolterman, and Josh Tenenberg. "The Anatomy of Prototypes: Prototypes as Filters, Prototypes as Manifestations of Design Ideas". In: *ACM Transactions on Computer-Human Interaction* 15.2 (July 2008), 7:1–7:27. ISSN: 1073-0516. DOI: 10.1145/1375761.1375762. URL: https://doi.org/10.1145/1375761.1375762 (visited on 10/19/2021).

[130] Nathan Lovato. *Squeezing More Juice out of Your Game Design!* https://bit.ly/2GsjSex. Accessed: 2019-10-27T12:44:31Z. Mar. 2015. URL: https://bit.ly/2GsjSex (visited on 10/27/2019).

[131] Todd Lubart. "How Can Computers Be Partners in the Creative Process: Classification and Commentary on the Special Issue". In: *International Journal of Human-Computer Studies* 63.4-5 (Oct. 2005), pp. 365–369. ISSN: 10715819. DOI: 10.1016/j.ijhcs.2005.04.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1071581905000418 (visited on 12/09/2021).

[132] Paul Luff and David Frohlich. "CHAPTER 13 - MIXED INITIATIVE INTERACTION". In: *Knowledge-Based Systems and Legal Applications*. Ed. by T. J. M. Bench-capon. Vol. 36. APIC. London: Academic Press, Jan. 1991, pp. 265–294. ISBN: 978-0-12-086441-6. DOI: 10.1016/B978-0-12-086441-6.50021-0. URL: https://www.sciencedirect.com/science/article/pii/B9780120864416500210 (visited on 10/18/2021).

[133] Tiago Machado et al. "AI-Assisted Game Debugging with Cicero". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. Rio de Janeiro: IEEE, July 2018, pp. 1–8. ISBN: 978-1-5090-6017-7. DOI: 10.1109/CEC.2018.8477829. URL: https://ieeexplore.ieee.org/document/8477829/ (visited on 04/11/2019).

[134] Tiago Machado et al. "Evaluation of a Recommender System for Assisting Novice Game Designers". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 15.1 (Oct. 2019), pp. 167–173. ISSN: 2334-0924. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/5240 (visited on 03/01/2022).

[135] Ryan Meyer. *ITSP Camera Explained*. https://www.youtube.com/watch?v=aAKwZt3aXQM&feature=emb_title&ab_channel=RyanMeyer. Accessed: 2020-10-06T12:58:22Z. May 2013. URL: https://www.youtube.com/watch?v=aAKwZt3aXQM&feature=emb_title&ab_channel=RyanMeyer (visited on 10/06/2020).

[136] Panagiotis Migkotzidis and Antonios Liapis. "SuSketch: Surrogate Models of Gameplay as a Design Assistant". In: *IEEE Transactions on Games* (2021), pp. 1–1. ISSN: 2475-1510. DOI: 10.1109/TG.2021.3068360.

[137] Rodolfo Mora-Zamora and Esteban Brenes-Villalobos. "Integrated Framework for Game Design". In: *Proceedings of the IX Latin American Conference on Human Computer Interaction*. CLIHC '19. New York, NY, USA: Association for Computing Machinery, Sept. 2019, pp. 1–6. ISBN: 978-1-4503-7679-2. DOI: 10.1145/3358961.3358984. URL: https://doi.org/10.1145/3358961.3358984 (visited on 10/06/2020).

[138] Lennart E. Nacke et al. "Player-Game Interaction Through Affective Sound". In: *Game Sound Technology and Player Interaction: Concepts and Developments*. IGI Global, Jan. 2011. ISBN: 978-1-61692-828-5. URL: https://www.igi-global.com/gateway/chapter/46796 (visited on 08/06/2020).

[139] Mark J. Nelson and Michael Mateas. "An Interactive Game-Design Assistant". In: *Proceedings of the 13th International Conference on Intelligent User Interfaces - IUI '08*. Gran Canaria, Spain: ACM Press, 2008, p. 90. ISBN: 978-1-59593-987-6. DOI: 10.1145/1378773.1378786. URL: http://portal.acm.org/citation.cfm?doid=1378773.1378786 (visited on 09/05/2018).

[140] Jan Willem Nijman. *The Art of Screenshake*. https://www.youtube.com/watch?v=AJdEqssNZ-U. Accessed: 2021-04-22. Dec. 2013. URL: https://www.youtube.com/watch?v=AJdEqssNZ-U (visited on 04/22/2021).

[141] Nintendo. *Super Mario 64*. 1996.

[142] Donald Norman. *The Design Of Everyday Things*. Basic Books, 1988.

[143] Aline Normoyle and Sophie Jörg. "Trade-Offs between Responsiveness and Naturalness for Player Characters". In: *Proceedings of the Seventh International Conference on Motion in Games - MIG '14*. Playa Vista, California: ACM Press, 2014, pp. 61–70. ISBN: 978-1-4503-2623-0. DOI: 10.1145/2668064.2668087. URL: http://dl.acm.org/citation.cfm?doid=2668064.2668087 (visited on 09/11/2019).

[144] David G Novick and Stephen Sutton. "What Is Mixed-Initiative Interaction?" In: (1997), p. 3.

[145] Christian Nutt. *The Magic of TowerFall : Depth, Simplicity, Community*. /view/news/241970/The_magic_of_TowerFall_Depth_simplicity_community.php. Accessed: 2020-05-15T10:28:22Z. 2015. URL: /view/news/241970/The_magic_of_TowerFall_Depth_simplicity_community.php (visited on 05/15/2020).

[146] Mauricio Orozco et al. "The Role of Haptics in Games". In: *Haptics Rendering and Applications*. London, United Kingdom: IntechOpen, Jan. 2012, pp. 217–234. ISBN: 978-953-307-897-7. URL: https://www.intechopen.com/books/haptics-rendering-and-applications/-the-role-of-haptics-in-gaming-experience- (visited on 06/29/2020).

[147] Marcus Pearce. "Boden and Beyond: The Creative Mind and Its Reception in the Academic Community." In: (2010), p. 21.

[148] Barney Pell. "METAGAME: A New Challenge for Games and Learning". In: *Programming in Artificial Intellegence: The Third Computer Olympiad. Ellis Horwood*. Ellis Horwood Limited, 1992, pp. 237–251.

[149] Robert Penner. *Robert Penner's Programming Macromedia Flash MX*. New York: McGraw-Hill/Osborne, 2002. ISBN: 978-0-07-222356-9.

[150] Diego Perez-Liebana et al. "General Video Game AI: A Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms". In: *arXiv:1802.10363 [cs]* (Feb. 2018). arXiv: 1802.10363 [cs]. URL: http://arxiv.org/abs/1802.10363 (visited on 11/27/2018).

[151] Diego Perez-Liebana et al. "General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms". In: *IEEE Transactions on Games* 11.3 (Sept. 2019), pp. 195–214. ISSN: 2475-1510. DOI: 10.1109/TG.2019.2901021.

[152] Diego Perez-Liebana et al. "The 2014 General Video Game Playing Competition". In: *IEEE Transactions on Computational Intelligence and AI in Games* 8.3 (Sept. 2016), pp. 229–243. ISSN: 1943-068X, 1943-0698. DOI: 10.1109/TCIAIG.2015.2402393. URL: http://ieeexplore.ieee.org/document/7038214/ (visited on 03/23/2019).

[153] Lee Perry. *The Single Most Useful Advice I Can Give for Making Any Game Better.. Feedback*. https://gamasutra.com/blogs/LeePerry/20130506/191739/The_single_most_useful_advice_I_can_give_for_making_any_game_better_feedback.php. Accessed: 2020-04-17T11:48:27Z. 2013. URL: https://gamasutra.com/blogs/LeePerry/20130506/191739/The_single_most_useful_advice_I_can_give_for_making_any_game_better_feedback.php (visited on 04/17/2020).

[154] Tomas 'DrPetter' Pettersson. *SFXR*. http://www.drpetter.se/project_sfxr.html. Accessed: 2020-07-11. 2007. URL: http://www.drpetter.se/project_sfxr.html (visited on 07/11/2020).

[155] Martin Pichlmair and Mads Johansen. "Designing Game Feel. A Survey." In: *IEEE Transactions on Games* IEEE Transactions on Games (Early Access).IEEE Transactions on Games (Early Access) (2021), pp. 1–20. ISSN: 2475-1510. DOI: 10.1109/TG.2021.3072241.

[156] Yoann Pignole. *Platformer Controls: How to Avoid Limpness and Rigidity Feelings*. https://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php. Accessed: 2019-10-27T12:43:44Z. 2014. URL: https://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php (visited on 10/27/2019).

[157] Kyle Pittman. *Math for Game Programmers: Building a Better Jump*. https://www.youtube.com/watch?v=hG9SzQxaCm8&ab_channel=GDC. Accessed: 2020-10-06T12:40:54Z. 2016. URL: https://www.youtube.com/watch?v=hG9SzQxaCm8&ab_channel=GDC (visited on 10/06/2020).

[158] Will Porter. *A Videogame History of Bullet-Time*. https://www.gamesradar.com/a-videogame-history-of-bullet-time/. Accessed: 2020-10-06T12:39:54Z. 2010. URL: https://www.gamesradar.com/a-videogame-history-of-bullet-time/ (visited on 10/06/2020).

[159] Justin K. Pugh, L. B. Soros, and Kenneth O. Stanley. "An Extended Study of Quality Diversity Algorithms". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. Denver Colorado USA: ACM, July 2016, pp. 19–20. ISBN: 978-1-4503-4323-7. DOI: 10.1145/2908961.2909000. URL: https://dl.acm.org/doi/10.1145/2908961.2909000 (visited on 10/19/2021).

[160] Kyle Pulver. *Platforming Ledge Forgiveness*. http://kpulv.com/123/Platforming_Ledge_Forgiveness/. Accessed: 2020-05-04T12:38:06Z. 2013. URL: http://kpulv.com/123/Platforming_Ledge_Forgiveness/ (visited on 05/04/2020).

[161] Fernando Ramallo. *Doodle Studio 95! By Fer Ramallo*. 2018. URL: https://fernandoramallo.itch.io/doodle-studio-95 (visited on 10/01/2021).

[162] Jef Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. ISBN: 0-201-37937-6.

[163] Johan Redström. *Making Design Theory*. Ed. by Ken Friedman and Erik Stolterman. Design Thinking, Design Theory. Cambridge, MA, USA: MIT Press, Sept. 2017. ISBN: 978-0-262-03665-8.

[164] William T Reeves. "Particle Systems A Technique for Modeling a Class of Fuzzy Objects". In: *ACM Transactions on Graphics* 2.2 (Apr. 1983), p. 17. DOI: 10.1145/357318.357320. URL: https://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/fuzzyParticles.pdf (visited on 05/11/2020).

[165] William T. Reeves. "Inbetweening for Computer Animation Utilizing Moving Point Constraints". In: *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '81*. Dallas, Texas, United States: ACM Press, 1981, pp. 263–269. ISBN: 978-0-89791-045-3. DOI: 10.1145/800224.806814. URL: http://portal.acm.org/citation.cfm?doid=800224.806814 (visited on 05/04/2020).

[166] Graeme Ritchie. "Some Empirical Criteria for Attributing Creativity to a Computer Program". In: *Minds and Machines* 17.1 (Mar. 2007), pp. 67–99. ISSN: 1572-8641. DOI: 10.1007/s11023-007-9066-2. URL: https://doi.org/10.1007/s11023-007-9066-2 (visited on 02/26/2022).

[167] Bill Rockenbeck. *The inFAMOUS: Second Son Particle System Architecture*. https://www.gdcvault.com/play/1020367/The-inFAMOUS-Second-Son-Particle. Accessed: 2020-10-06T12:23:34Z. 2014. URL: https://www.gdcvault.com/play/1020367/The-inFAMOUS-Second-Son-Particle (visited on 10/06/2020).

[168] Tim Rogers. *In Praise of Sticky Friction*. https://kotaku.com/in-praise-of-sticky-friction-5558166. Accessed: 2021-04-22. June 2010. URL: https://kotaku.com/in-praise-of-sticky-friction-5558166 (visited on 04/22/2021).

[169] Michel Sabbagh. *The Art of Designing Visceral and Engaging Bullet Time Gunplay*. https://michelsabbagh.wordpress.com/2015/10/07/the-art-of-designing-visceral-and-engaging-bullet-time-gunplay/. Accessed: 2020-04-17T14:21:12Z. Oct. 2015. URL: https://michelsabbagh.wordpress.com/2015/10/07/the-art-of-designing-visceral-and-engaging-bullet-time-gunplay/ (visited on 04/17/2020).

[170] Dan Saffer. *Microinteractions: Designing with Details*. Reilly Media, Inc., 2013.

[171] Vardan Saini and Matthew Guzdial. "A Demonstration of Mechanic Maker: An AI for Mechanics Co-Creation". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1 (Oct. 2020), pp. 325–327. ISSN: 2334-0924. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/7450 (visited on 02/18/2022).

[172] Adam Saltsman. *Canabalt*. 2009.

[173] Adam Saltsman. *Tuning Canabalt*. https://www.gamasutra.com/blogs/AdamSaltsman/20100929/88155/Tuning_Canabalt.php. Accessed: 2019-10-27T12:43:59Z. 2010. URL: https://www.gamasutra.com/blogs/AdamSaltsman/20100929/88155/Tuning_Canabalt.php (visited on 10/27/2019).

[174] Majed Samad et al. "Pseudo-Haptic Weight: Changing the Perceived Weight of Virtual Objects By Manipulating Control-Display Ratio". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow Scotland Uk: ACM, May 2019, pp. 1–13. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300550. URL: https://dl.acm.org/doi/10.1145/3290605.3300550 (visited on 09/23/2021).

[175] Tom Schaul. "A Video Game Description Language for Model-Based or Interactive Learning". In: *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. Niagara Falls, ON, Canada: IEEE, Aug. 2013, pp. 1–8. ISBN: 978-1-4673-5311-3 978-1-4673-5308-3. DOI: 10.1109/CIG.2013.6633610. URL: http://ieeexplore.ieee.org/document/6633610/ (visited on 04/25/2019).

[176] Tom Schaul. "An Extensible Description Language for Video Games". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014), pp. 325–331. ISSN: 1943-068X, 1943-0698. DOI: 10.1109/TCIAIG.2014.2352795. URL: http://ieeexplore.ieee.org/document/6884801/ (visited on 04/02/2019).

[177] Jesse Schell. *The Art of Game Design: A Book of Lenses*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN: 0-12-369496-5.

[178] Jacob Schrum et al. "Interactive Evolution and Exploration Within Latent Level-Design Space of Generative Adversarial Networks". In: *arXiv:2004.00151 [cs]* (Mar. 2020). arXiv: 2004.00151 [cs]. URL: http://arxiv.org/abs/2004.00151 (visited on 02/27/2022).

[179] Jimmy Secretan et al. "Picbreeder: Evolving Pictures Collaboratively Online". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Florence, Italy: Association for Computing Machinery, Apr. 2008, pp. 1759–1768. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357328. URL: https://doi.org/10.1145/1357054.1357328 (visited on 07/13/2020).

[180] Noor Shaker, Mohammad Shaker, and Julian Togelius. "Ropossum: An Authoring Tool for Designing, Optimizing and Solving Cut the Rope Levels". In: (2013), p. 2.

[181] Noor Shaker, Julian Togelius, and Mark Nelson. *Procedural Content Generation in Games*. USA: Springer, Oct. 2016. ISBN: 978-3-319-42714-0. URL: http://www.springer.com/gp/book/9783319427140 (visited on 06/08/2021).

[182] Tanya X. Short and Tarn Adams. *Procedural Storytelling in Game Design*. Boca Raton, FL: Taylor & Francis, 2019. ISBN: 978-1-138-59530-9.

[183] Miguel Sicart. *Play Matters*. MIT Press, 2014.

[184] Miguel Angel Sicart. "Loops and Metagames: Understanding Game Design Structures". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015), June 22-25, 2015, Pacific Grove, CA, USA*. 2015. ISBN: 978-0-9913982-4-9.

[185] Karl Sims. "Interactive Evolution of Dynamical Systems". In: *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. 1992.

[186] Andrey Sitnik and Ivan Solovev. *Easing Functions Cheat Sheet*. http://easings.net/. Accessed: 2020-05-04T12:25:58Z. Accessed: 2020-05-04 14:25:58. URL: http://easings.net/ (visited on 05/04/2020).

[187] Kristin Siu, Eric Butler, and Alexander Zook. *A Programming Model for Boss Encounters in 2D Action Games*. Technical Report WS-16-22. Experimental AI in Games: Papers from the AIIDE Workshop, 2016. URL: https://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/14058.

[188] Seth Sivak. *GAME 3400 Level Design - Moment Based Design*. Technology. 2012. URL: https://www.slideshare.net/sjsivak/game-3400-level-design-moment-based-design (visited on 06/29/2020).

[189] Smashpedia. *Invincibility Frame*. https://bit.ly/2JEPgrz. Accessed: 2020-10-06T12:50:44Z. URL: https://bit.ly/2JEPgrz (visited on 10/06/2020).

[190] Adam M Smith, Mark J Nelson, and Michael Mateas. “Computational Support for Play Testing Game Sketches”. In: (2009), p. 6.

[191] Adam M. Smith and Michael Mateas. “Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games”. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. Copenhagen, Denmark: IEEE, Aug. 2010, pp. 273–280. ISBN: 978-1-4244-6295-7. DOI: 10.1109/ITW.2010.5593343. URL: http://ieeexplore.ieee.org/document/5593343/ (visited on 10/18/2021).

[192] Adam Marshall Smith. “Mechanizing Exploratory Game Design”. PhD thesis. UC Santa Cruz, 2012. URL: https://escholarship.org/uc/item/4600g227 (visited on 12/07/2021).

[193] Gillian Smith, Jim Whitehead, and Michael Mateas. “Tanagra: A Mixed-Initiative Level Design Tool”. In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. FDG ’10. New York, NY, USA: Association for Computing Machinery, June 2010, pp. 209–216. ISBN: 978-1-60558-937-4. DOI: 10.1145/1822348.1822376. URL: https://doi.org/10.1145/1822348.1822376 (visited on 10/21/2020).

[194] Gillian Margaret Smith. “Expressive Design Tools: Procedural Content Generation for Game Designers”. PhD thesis. UC Santa Cruz, 2012. URL: https://escholarship.org/uc/item/0fn558gq (visited on 12/07/2021).

[195] Jonatan Söderström. *The Four-Hour Game Design by Cactus*. https://www.gdcvault.com/play/1243/(304)-The-Four-Hour-Game. Accessed: 2020-07-11. 2009. URL: https://www.gdcvault.com/play/1243/(304)-The-Four-Hour-Game (visited on 07/11/2020).

[196] Jiesang Song. *Improving the Combat 'Impact' Of Action Games*. Apr. 2005.

[197] Larry Squire et al. *Fundamental Neuroscience*. Academic Press, Dec. 2012. ISBN: 978-0-12-385871-9.

[198] Oskar Stålberg. *Townscaper*. Oskar Stålberg. 2020.

[199] Jasper Stephenson. *A UX Analysis of First-Person Shooter Damage Indicators*. https://medium.com/@jasper.stephenson/a-ux-analysis-of-first-person-shooter-damage-indicators-59ac9d41caf8. Accessed: 2020-10-06T12:49:09Z. Mar. 2018. URL: https://medium.com/@jasper.stephenson/a-ux-analysis-of-first-person-shooter-damage-indicators-59ac9d41caf8 (visited on 10/06/2020).

[200] Matthew Stephenson et al. “An Overview of the Ludii General Game System”. In: *2019 IEEE Conference on Games (CoG)*. Aug. 2019, pp. 1–2. DOI: 10.1109/CIG.2019.8847949.

[201] Paul Suddaby. *5 Important Ways to Add Polish to Your Game*. https://gamedevelopment.tutsplus.com/articles/5-important-ways-to-add-polish-to-your-game--gamedev-7642. Accessed: 2020-08-06T09:52:37Z. May 2013. URL: https://gamedevelopment.tutsplus.com/articles/5-important-ways-to-add-polish-to-your-game--gamedev-7642 (visited on 08/06/2020).

[202] Adam Summerville. “Expanding Expressive Range: Evaluation Methodologies for Procedural Content Generation”. In: (2018), p. 7.

[203] Adam Summerville. “Learning from Games for Generative Purposes”. PhD thesis. 2018.

[204] Adam Summerville et al. “Gemini: Bidirectional Generation and Analysis of Games via ASP”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 14.1 (Sept. 2018), pp. 123–129. ISSN: 2334-0924. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/13013 (visited on 07/29/2021).

[205] Adam Summerville et al. “Mechanics Automatically Recognized via Interactive Observation: Jumping”. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. New York, NY, USA: Association for Computing MachineryNew YorkNYUnited States, 2017, pp. 1–10. ISBN: 978-1-4503-5319-9.

[206] Adam Summerville et al. “Procedural Content Generation via Machine Learning (PCGML)”. In: *IEEE Transactions on Games* 10.3 (Sept. 2018), pp. 257–270. ISSN: 2475-1510. DOI: 10.1109/TG.2018.2846639.

[207] Ivan E. Sutherland. “Sketchpad a Man-Machine Graphical Communication System”. In: *SIMULATION* 2.5 (May 1964), R–3. ISSN: 0037-5497. DOI: 10.1177/003754976400200514. URL: https://doi.org/10.1177/003754976400200514 (visited on 05/26/2021).

[208] Nick Suttner et al. *Eggplant: The Secret Lives of Games (Formerly The Spelunky Showlike) 36: Game Feel as Procrastination with Jan Willem Nijman*. URL: https://thespelunkyshowlike.libsyn.com/36-game-feel-as-procrastination-with-jan-willem-nijman (visited on 08/06/2020).

[209] Nick Suttner et al. *Eggplant: The Secret Lives of Games (Formerly The Spelunky Showlike) 38: The Rhythms and Layers of Ryan Clark*. URL: https://thespelunkyshowlike.libsyn.com/38-the-rhythms-and-layers-of-ryan-clark (visited on 08/06/2020).

[210] Nick Suttner et al. *Eggplant: The Secret Lives of Games (Formerly The Spelunky Showlike) 39: The Tricks of the Toolkit with Mark Brown*. URL: https://thespelunkyshowlike.libsyn.com/39-gmtk (visited on 08/06/2020).

[211] Nick Suttner et al. *Eggplant: The Secret Lives of Games (Formerly The Spelunky Showlike) 42: The Secrets of Simplicity with Martin Jonasson*. URL: https://thespelunkyshowlike.libsyn.com/42-the-secrets-of-simplicity-with-martin-jonasson (visited on 08/06/2020).

[212] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning Series. Cambridge, MA: The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[213] Penelope Sweetser and Peta Wyeth. “GameFlow: A Model for Evaluating Player Enjoyment in Games”. In: *Computers in Entertainment* 3.3 (July 2005), p. 3. DOI: 10.1145/1077246.1077253. URL: https://doi.org/10.1145/1077246.1077253 (visited on 08/24/2020).

[214] Steve Swink. *Game Feel*. Morgan Kaufmann, 2009. ISBN: 0-12-374328-1.

[215] Steve Swink. *Game Feel: The Secret Ingredient*. https://www.gamasutra.com/view/feature/130734/game_feel_the_secret_ingredient.php?print=1. Accessed: 2020-04-17T11:52:32Z. 2007. URL: https://www.gamasutra.com/view/feature/130734/game_feel_the_secret_ingredient.php?print=1 (visited on 04/17/2020).

[216] Teknopants. *Samurai Gunn*. 2013.

[217] Frank Thomas and Ollie Johnston. *The Illusion of Life: Disney Animation*. New York: Abbeville Press, 1981. ISBN: 0-89659-233-2.

[218] Julian Togelius and Noor Shaker. "The Search-Based Approach". In: *Procedural Content Generation in Games*. Cham: Springer International Publishing, 2016, pp. 17–30. ISBN: 978-3-319-42714-0 978-3-319-42716-4. DOI: 10.1007/978-3-319-42716-4_2. URL: http://link.springer.com/10.1007/978-3-319-42716-4_2 (visited on 10/22/2018).

[219] Julian Togelius et al. "Search-Based Procedural Content Generation". In: *Applications of Evolutionary Computation*. Ed. by David Hutchison et al. Vol. 6024. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 141–150. ISBN: 978-3-642-12238-5 978-3-642-12239-2. DOI: 10.1007/978-3-642-12239-2_15. URL: http://link.springer.com/10.1007/978-3-642-12239-2_15 (visited on 10/19/2021).

[220] Ruben Rodriguez Torrado et al. "Deep Reinforcement Learning for General Video Game AI". In: *arXiv:1806.02448 [cs, stat]* (June 2018). arXiv: 1806.02448 [cs, stat]. URL: http://arxiv.org/abs/1806.02448 (visited on 02/12/2019).

[221] Mike Treanor et al. "Game-O-Matic: Generating Videogames That Represent Ideas". In: *Proceedings of the The Third Workshop on Procedural Content Generation in Games*. PCG'12. New York, NY, USA: Association for Computing Machinery, May 2012, pp. 1–8. ISBN: 978-1-4503-1447-3. DOI: 10.1145/2538528.2538537. URL: https://doi.org/10.1145/2538528.2538537 (visited on 09/17/2021).

[222] Matt Vainio. *The Visual Effects of inFAMOUS: Second Son*. https://www.youtube.com/watch?v=o2yFxPY2b1o&ab_channel=GDC. Accessed: 2020-10-06T12:25:29Z. 2014. URL: https://www.youtube.com/watch?v=o2yFxPY2b1o&ab_channel=GDC (visited on 10/06/2020).

[223] Laurene Vaughan. *Practice-Based Design Research*. Bloomsbury Publishing, Jan. 2017. ISBN: 978-1-4742-6782-3.

[224] Mark Venturelli. *Game Feel Tips I: The Ghost Jump*. https://gamasutra.com/blogs/MarkVenturelli/20140810/223001/Game_Feel_Tips_I_The_Ghost_Jump.php. Accessed: 2020-04-17T13:45:33Z. 2014. URL: https://gamasutra.com/blogs/MarkVenturelli/20140810/223001/Game_Feel_Tips_I_The_Ghost_Jump.php (visited on 04/17/2020).

[225] Bret Victor. *Drawing Dynamic Visualizations*. Stanford HCI seminar, Feb. 2013. URL: https://vimeo.com/66085662 (visited on 09/30/2021).

[226] Bret Victor. *Media for Thinking the Unthinkable*. MIT Media Lab, Apr. 2013. URL: http://worrydream.com/MediaForThinkingTheUnthinkable/ (visited on 09/30/2021).

[227] Bret Victor. *Stop Drawing Dead Fish*. SF SIGGRAPH, Nov. 2012. URL: https://vimeo.com/64895205 (visited on 09/30/2021).

[228] Bret Victor. *The Future of Programming*. DBX conference, July 2013. URL: http://worrydream.com/dbx/ (visited on 09/30/2021).

[229] Kurt Vonnegut. *The Shapes of Stories*. https://www.youtube.com/watch?v=oP3c1h8v2ZQ. Accessed: 2020-06-29T16:38:20Z. 1985. URL: https://www.youtube.com/watch?v=oP3c1h8v2ZQ (visited on 06/29/2020).

[230] Douglas Wilson. *A Tale of Two Jousts: Multimedia, Game Feel, and Imagination*. https://www.youtube.com/watch?v=hpdcek4hLA8. Accessed: 2020-04-17T10:37:46Z. 2016. URL: https://www.youtube.com/watch?v=hpdcek4hLA8 (visited on 04/17/2020).

[231] Alex Wiltshire. *How Hitboxes Work*. https://www.pcgamer.com/how-hitboxes-work/. Accessed: 2020-08-24T09:04:16Z. Aug. 2020. URL: https://www.pcgamer.com/how-hitboxes-work/ (visited on 08/24/2020).

[232] Robert Yang. *Queering Game Feel*. QGCon 2018 - Google Slides, 2018. URL: tinyurl.com/QueeringGameFeel (visited on 04/17/2020).

[233] Georgios N Yannakakis, Antonios Liapis, and Constantine Alexopoulos. "Mixed-Initiative Co-Creativity". In: *Foundations of Digital Games 2014*. Proceedings of the 9th International Conference on the Foundations of Digital Games: Society for the Advancement of the Science of Digital Games, 2014, p. 8. ISBN: 978-0-9913982-2-5.

[234] Georgios N. Yannakakis, Héctor P. Martínez, and Arnav Jhala. "Towards Affective Camera Control in Games". In: *User Modeling and User-Adapted Interaction* 20.4 (Oct. 2010), pp. 313–340. ISSN: 0924-1868, 1573-1391. DOI: 10.1007/s11257-010-9078-0. URL: http://link.springer.com/10.1007/s11257-010-9078-0 (visited on 06/29/2020).

[235] Du-Mim Yoon and Kyung-Joong Kim. "Interactive Evolution of 3D Models Based on Direct Manipulation for Video Games". In: *Procedia Computer Science* 24 (2013), pp. 137–142. ISSN: 18770509. DOI: 10.1016/j.procs.2013.10.036. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050913011782 (visited on 01/28/2020).

[236] Chris Zimmerman. *Reading the Player's Mind Through His Thumbs: Inferring Player Intent Through Controller Input*. https://www.gdcvault.com/play/1012339/Reading-the-Player-s-Mind. Accessed: 2020-10-06T12:20:45Z. 2010. URL: https://www.gdcvault.com/play/1012339/Reading-the-Player-s-Mind (visited on 10/06/2020).

Chapter 9

Publications

9.1 Video game description language environment for Unity machine learning agents

Video Game Description Language Environment for Unity Machine Learning Agents

Mads Johansen
IT University of Copenhagen
Copenhagen, Denmark
madj@itu.dk

Martin Pichlmair
IT University of Copenhagen
Copenhagen, Denmark
mpic@itu.dk

Sebastian Risi
IT University of Copenhagen
Copenhagen, Denmark
sebr@itu.dk

***Abstract*—This paper introduces *UnityVGDL*, a port of the Video Game Description Language (VGDL) to the widely used Unity game engine. Our framework is based on the General Video Game AI (GVGAI) competition framework and implements its core ontology, including a forward model. It integrates the Unity Machine Learning Agents (ML-Agents) toolkit with VGDL to train and run agents in VGDL-described games. We compare baseline learning results between GVGAI and UnityVGDL across four different games and conclude that the Unity port is comparable to the GVGAI framework. UnityVGDL is available at: https://github.com/pyjamads/UnityVGDL**

I. Introduction

In 2018 Juliani et al. [1] introduced a new toolkit for Machine Learning Agents (ML-Agents) within the Unity game engine. Unity is a widely used game engine with extensive functionality and flexibility to create games and simulation environments. The ML-Agents toolkit provides an easy way to integrate reinforcement learning agents, imitation learning agents and scripted agents into the environments created with Unity. The ML-Agents toolkit implements a custom python pipeline for training agents, as well as the OpenAI Gym interface [2].

Juliani et al. attribute the recent significant advances in deep reinforcement learning to the existence of rapid development environments such as the Arcade Learning Environment (ALE) [3], VizDoom [4] and Mujoco [5]. With a large set of games, ALE provided the base for a breakthrough in control-from-pixels called the Deep Q-Network by Mnih et al. [6] and games have been used extensively to test recent machine learning advances [7], [8]. However, while new environments such as the Obstacle Tower continue to be added to the ML-Agents toolkit [9], it only provides a small set of testing environments when compared to ALE.

One neglected field with significant potential to produce advances in AI research is that of General Video Game Playing (GVGP) proposed by Levine et al. [10]. GVGP aims to extend the challenge of playing many different games with the same algorithm, to playing many different games with the same agents. Levine et al. also envisaged "*the development of a Video Game Description Language (VGDL) as a way of concisely specifying video games*" [10]. The vision was a VGDL capable of describing 2D arcade style games like the Atari games found in ALE. The authors argue for hosting competitions in GVGP by challenging AI agents to play previously unseen games.

In 2014 Tom Schaul created an extensible version of VGDL [11] along with a framework for computational intelligence research [12]. Later Perez-Liebana et al. [13] built on VGDL to create the General Video Game AI (GVGAI) framework and ran the GVGAI Competition based on GVGP.

The GVGAI framework contains a large number of different games and game sets. These VGDL games are a mixture of arcade games like those in ALE and interesting computational intelligence challenges. The games vary in goals and types of interactions. Some are imitations of games that exist elsewhere, like Aliens[1], Frogs[2] or Lemmings[3]. Other games were designed as machine learning challenges, like Wait for breakfast (a simple game where you wait to be served breakfast) or the classic T-maze problem for testing reinforcement learning memorization [14]. The GVGAI competition has been running since 2014 and the list of games has been growing steadily ever since.

UnityVGDL expands the VGDL family from Python and Java to Unity and C#. UnityVGDL brings the GVGAI VGDL ontology to Unity and adds support for training agents with the ML-Agents toolkit. By combining those, ML-Agents can benefit from the corpus of games provided by UnityVGDL. Because UnityVGDL uses the GVGAI VGDL ontology, it allows GVGAI VGDL games to be interpreted and viewed in the Unity Editor and compiled as executables.

By integrating Unity and ML-Agents in UnityVGDL, we expose VGDL to a wider audience of potential machine learning researchers. At the same time we introduce VGDL to a wider audience of game creators. To validate the UnityVGDL framework we train reinforcement learning agents and demonstrate that their performance is comparable to GVGAI agents.

The UnityVGDL framework is available on GitHub[4] under the Apache open source license. The repository has instructions on how to set up ML-Agents, and include the same set of assets as GVGAI, available for AI research and competitions.

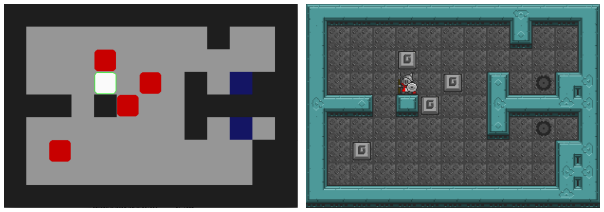


Fig. 1. Visualizations of VGDL Sokoban: left PyVGDL, right GVGAI

```
wwwwwwwwwwww
w          w  w
w    1        w
w    A 1 w 0ww
www w1  wwwww
w          w 0 w
w 1          ww
w             ww
wwwwwwwwwwww
```

Fig. 2. Simple Sokoban level description. 'A' denotes the Avatar (controlled by player), 'w' denotes walls, '1' denotes boxes and '0' denotes holes.

```
BasicGame
  SpriteSet
    hole   > Immovable color=DARKBLUE
    avatar > MovingAvatar
    box    > Passive
  LevelMapping
    0 > hole
    1 > box
  InteractionSet
    avatar wall > stepBack  #stop at wall
    box avatar  > bounceForward #push box
    box wall    > undoAll #wall stops box
    box box     > undoAll  #box stops box
    box hole    > killSprite #destroy box
  TerminationSet
    SpriteCounter stype=box limit=0 win=True
```

Fig. 3. Game description for a simple Sokoban game. Objects defined in the `SpriteSet` can be used to define interactions, terminations and level mapping. The VGDL Effects such as `stepBack` have comments '#stop at wall' to explain what they do when the two object types collide. The game is won by pushing boxes into holes, until no boxes are left.

II. Related Work

A. Video Game Description Language (VGDL)

PyVGDL is the formalized version of VGDL created by Tom Schaul, focused on describing a wide variety of 2D arcade games. Descriptions are split into two parts. The first is the game description (objects, rules, goals, level mapping). The second is the level description. PyVGDL needs one of each to interpret and run a working game. The game description consists of four sets:

- The `SpriteSet` contains definitions of all objects (including look and behavior) in the game.
- The `LevelMapping` describes the mapping between characters in the level description and objects in the `SpriteSet`.
- The `InteractionSet` defines all interaction effects that happen when two objects collide.
- The `TerminiationSet` describes when the level ends.

PyVGDL generates playable versions from VGDL game and level descriptions using the Pygame library. An example of a running game can be seen in the left visualization in Fig. 1. PyVGDL features 21 example games based on grid physics and six games based on continuous physics. They are meant to show the diversity of games describable in PyVGDL. Imitations of Mario, Zelda, Sokoban, Aliens, Frogger, and Pong are featured, along with the T-maze and traveling salesman problem. PyVGDL also has the option of rendering a first-person perspective of games. An extended Backus-Naur Form description of the full PyVGDL grammar is in [12]. The framework is available on Github[5]. The game shown in Fig. 1 is a simple Sokoban game defined in VGDL; level description and game description can be seen in Fig. 2 and Fig. 3, respectively.

B. General Video Game AI (GVGAI)

Perez-Liebana et al. [13] launched the General Video Game AI Competition in 2014, introducing a Java implementation of the VGDL ontology of PyVGDL called the GVGAI framework. The GVGAI framework also added Sprite assets, improving the visual appearance of the VGDL games as seen on the right in Fig. 1. The framework contains sprite assets licensed by Oryx Design Lab, that are free to use for research and competition[6]. The sprite assets help to communicate the essence of the game to users and make the games more visually appealing. Alongside the GVGAI competition several research projects[7] have been conducted, extending the VGDL capabilities [16], [17] as well as the list of implemented games. The number of implementations of agent types has been growing too, see [13], [18], [19]. The GVGAI framework comes with several different simple, heuristic and planning agents [18], a few based on macro actions [17] and learning agents [19]. It also features functionality to replay recorded agent actions.

The GVGAI framework contains VGDL descriptions for 121 single player games, 49 two-player games, and 11 continuous physics games. Unlike PyVGDL a first person rendering option is not available in the GVGAI framework.

[1]Space Invaders (Taito, 1978)
[2]Frogger (Konami, 1981)
[3]Lemmings (DMA Design, 1991)
[4]https://github.com/pyjamads/UnityVGDL
[5]https://github.com/schaul/py-vgdl

[6]Early versions of GVGAI used Open License assets by http://kenney.nl as can be seen in [15]
[7]http://gvgai.net/papers.php

The competition has changed a lot over the years [20]. The 2018 competition featured four tracks: Single Player, Two-Player, Level Generation and Rule Generation. The Single Player and Two-Player tracks are for playing the games. Level Generation aims to generate levels based on a VGDL game description. Rule Generation is a competition to generate game descriptions based on a VGDL level description. The overall structure of the game descriptions is similar in PyVGDL and GVGAI. Yet, there are minor differences between their respective ontology implementations — mostly regarding names and availability of specific Sprite and Effect types (i.e. WalkAvater vs. WalkerAvatar). A description of the VGDL Language and GVGAI ontology can be found online[8] in the GVGAI documentation.

In 2018 Torrado et al. [19] added the OpenAI Gym [2] interface to GVGAI. The OpenAI Gym interface was created to streamline the many different ways reinforcement learning systems interact with environments. The interface makes it much easier to compare and recreate the results of different algorithms across different environments. OpenAI also provides a set of baseline algorithms that can be used for comparison. Torrado et al. planned to compare their results with the results of Mihn et al. [6] using DQN on ALE [3]. However, the scores in the VGDL descriptions have not been modeled after the Atari games; as such the scores between VGDL and Atari games are not directly comparable. Instead, they decided to compare three of OpenAI's baseline reinforcement learning implementations with the state of the art planning agents on a selection of eight VGDL games from the GVGAI framework. The results presented by Torrado et al. will be discussed and compared in later sections.

C. Unity Machine Learning Agents (ML-Agents)

The ML-Agents toolkit allows developers to easily integrate machine learning agents in their games and provides AI researchers with an easily customizable platform to experiment with. The framework defines three different brain types: player brain, heuristic brain (i.e. scripted behavior), and learning brains. These brains control agents in the environment. In the ML-Agents toolkit Juliani et al. [1] chose to implement a baseline reinforcement learning (RL) [21] algorithm based on Proximal-Policy Optimization (PPO) [22].

Additionally for heuristic and RL agents, the ML-Agents toolkit provides an imitation learning agent. This agent gives developers the ability to teach their agents by example. It allows them to draft the kind of behaviors they would like to see in their game with relatively short training periods. Imitation learning can often work as a better reference than a random baseline agent when developing new algorithms.

III. VGDL in Unity

This paper introduces a Unity framework called UnityVGDL. The framework combines the GVGAI VGDL ontology with the ML-Agents toolkit inside Unity. The following section explains how the framework is structured.

A. Architecture

The Unity scenes are structured around the C# VGDL implementation with the outer most layer being the ML-Agents framework as depicted in Fig. 4. The `VGDLAcademy` manages which VGDL game environment to load. The `VGDLAgent` controls the player avatar. The agent uses the `VGDLRunner` to parse and run the VGDL game passed from the academy. The runner parses the game and level descriptions from the Resources folder, and instantiates the `VGDLGame` which in turn instantiates all VGDL sprites, effects, and terminations from the description. The VGDL game updates can be driven by the academy or by regular Unity updates. For learning purposes, letting the academy control updates guarantees the games are updated correctly even when running many instances in parallel. The `VGDLRunner` can also drive the visualization by calling the `VGDLRenderer`. The renderer renders the VGDL game world either to a Render Texture[9] or directly to the back buffer. When dealing with learning agents, the current VGDL ML-Agents implementation only supports visual observations. This means the game is rendered to a Render Texture and then passed as input to the `VGDLAgent`. The C# VGDL implementation in UnityVGDL is based on the GVGAI ontology and its JavaVGDL engine, with a few modifications.

The GVGAI ontology allows UnityVGDL to interpret and play games from the GVGAI backlog of games, however, a few things have been changed under the hood, mainly in regards to lookup tables.

`VGDLSprite`, `VGDLEffect` and `VGDLTermination` class lookup tables were replaced with C# Reflection, for increased extensibility, more closely resembling that in PyVGDL [11]. Further, JavaVGDL implements a type registration and lookup system for the user-defined objects in VGDL. This type registration system converts object names (the `stype`) to a unique integer. UnityVGDL stores the object names directly as keys in C# dictionaries. During development, this direct lookup avoided a layer of abstraction in the code. Since the abstraction also serves as an optimization in the JavaVGDL engine it could become useful again if the forward model needs optimization.

B. UnityVGDL Scenes

The UnityVGDL Testing scene hierarchy seen in Fig. 5 contains a `VGDLAcademy`, a single `VGDLRunner` and `VGDLAgent`. This scene is set up to run a single game with either human, heuristic, or learning brains controlling the VGDL avatar. The scene can be used directly in the editor to load and play a VGDL description with a player brain, to evaluate a trained learning brain play or to build and run a heuristic brain. Additionally, UnityVGDL has several different scenes with learning setups for one, two, four, or eight game instances in parallel. These scenes contain a single `VGDLAcademy`, the adequate number of VGDL instances (`VGDLRunner` and `VGDLAgent`) and a corresponding number of UI elements for

[8] https://github.com/GAIGResearch/GVGAI/wiki/VGDL-Language

[9] https://docs.unity3d.com/Manual/class-RenderTexture.html

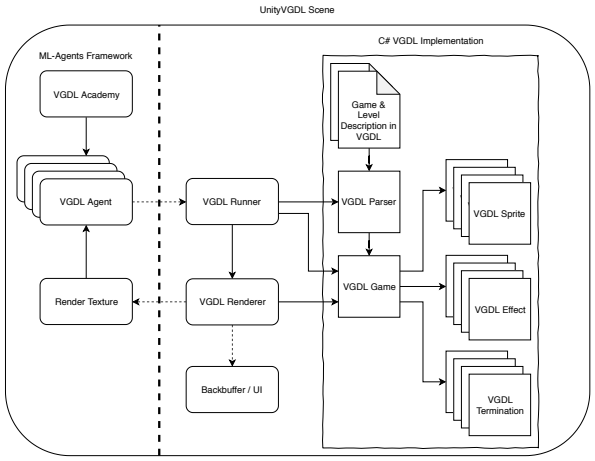


Fig. 4. UnityVGDL Scene Structure

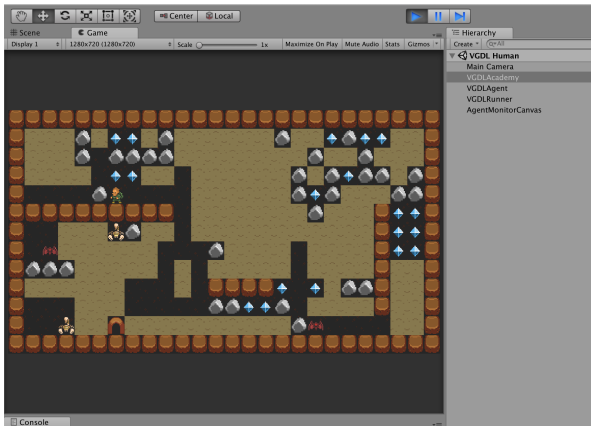


Fig. 5. UnityVGDL Testing scene running one game in the Unity editor

displaying multiple games side by side. The learning scenes can be compiled to executables for faster execution. The four instance version can be seen training on Wait for Breakfast in Fig. 6. Displaying multiple instances at once allows an observer to follow the ongoing training process. The training instances are each rendered in the resolution the agents need (84×84 pixels by default).

ML-Agents handles learning brains in two ways, either training a brain with the python backend or using a previously trained brain with the TensorFlow sharp plugin[10]. To train a learning brain to play a VGDL game, one of the learning scenes has to be edited to load the correct game. Then either a build is compiled or the editor can be used directly as the learning environment. The python ml-agents process is launched to start training. This process will automatically either connect to the editor or launch the executable, based on command line parameters. The default maximum learning steps are set to 50k. After reaching 50k steps the learning

[10]More information about ML-Agents training is available under "documentation" on https://github.com/Unity-Technologies/ml-agents/

Fig. 6. The generated executable from UnityVGDL of the game Wait for Breakfast, rendering four simultaneous game instances. Each game instance renders an 84×84 pixel view of the game world, that the agents get as their visual observation.

stops, and the model is saved. ML-Agents reports progress every 2000 steps to a TensorBoard[11] summery while training.

C. Controlling VGDL Avatars

UnityVGDL does not include the GVGAI competition framework. The ML-Agents framework can replace some of the wrapping functionality that GVGAI provides. The three different brain types in ML-Agents can be used in UnityVGDL, to define how avatars (i.e. player controlled sprites) in VGDL games are controlled. The player brain enables human players to control the avatar, the heuristic brain allows scripted behavior (i.e. random agents or planning agents) and lastly the learning brain for reinforcement or imitation learning. UnityVGDL implements a forward model to allow planning algorithms like Monte Carlo Tree Search as a `VGDLDecision` for the heuristic brain. The `VGDLDecision` class is an extension to the ML-Agents `Decision` class with the VGDL forward model. The current UnityVGDL ML-Agents integration only handles visual observations as input to the learning brains. With advances in deep reinforcement learning [7], learning games from pixels is very common. UnityVGDL does not implement general vector observation due to the many variations in VGDL games. VGDL games vary in size, number of objects and object types. Because the ML-Agents vector input space has to be fixed, capturing every possible VGDL variation was left to future work. Learning brains and `VGDLAgents` would have to be customized to individual games, to accommodate vector observations without a general approach.

D. New Environments for ML-Agents

Juliani et al. [1] suggested using their ML-Agents toolkit to make Unity a simulation platform for learning environments. UnityVGDL is one such environment, extending the available ML-Agents environments with the backlog of VGDL games created for the GVGAI framework. UnityVGDL also allows

[11]https://www.tensorflow.org/guide/summaries_and_tensorboard

researchers to define new games by using and extending the available VGDL ontology. New VGDL games simple have to be added to the resources folder in the project, after which they can be run or used for training with one of the scenes available in UnityVGDL. Extending the ontology can be achieved by inheriting from `VGDLSprite`, `VGDLEffect` or `VGDLTermination` or any of their sub-classes. Extensions will be available immediately because UnityVGDL uses C# reflection to look up the ontology.

IV. Experiments

To validate the UnityVGDL implementation, a selection of games should be compared between the UnityVGDL and JavaVGDL framework. The recent baseline learning results on GVGAI [19] presented a unique opportunity for comparing results on different games. This section outlines the games selected for comparison between UnityVGDL and GVGAI.

A. VGDL Games

Torrado et al. [19] selected eight games with the following consideration in mind:

> *"We tried to get an even distribution across the range going from games that are easy for planning agents, like Aliens, to very difficult, like Superman. The game difficulties are based on the analysis by Bontrager et al. [18]."*

Out of the eight games tested by Torrado et al. four games showed good learning behaviors: Aliens, Boulder Dash, Wait for Breakfast and Zelda. We describe these four games in more detail, which are also the games we choose for our learning test of UnityVGDL:

- **Aliens:** In Aliens the player uses a spaceship at the bottom of the screen to shoot at attacking aliens moving across the screen and dropping bombs. Based on the Atari game of the same name.
- **Boulder Dash**[12]: In Boulder Dash, the player digs tunnels to collect diamonds while avoiding enemies and falling rocks. Based on the Atari game of the same name.
- **Wait for Breakfast**: Game about waiting to be served breakfast, with the simple goal of waiting next to the correct table until the waiter comes by and serves the player. Mostly an interesting ML challenge.
- **Zelda**: Loose interpretation of Zelda or the Atari game Adventure with kill-able enemies and keys, that need to be picked up, before the level can be completed.

Torrado et al. compare their RL results to those of planning agents and of a random agent. By using the same games our results are directly comparable to theirs.

We added a max step limit per episode of 1,500 to ensure training episodes end. Aliens has a natural ending before that limit and Wait for Breakfast has a built-in time limit of 1,500 steps. However Boulder Dash and Zelda both need to be reset to avoid getting stuck. The experiments were all run using compiled executables on a high-end MacBook Pro 2018 laptop for one million steps. The game scene in each executable is set up with one or eight game instances running simultaneously[13], each with a separate agent and the same Learning-Brain attached. The brains use visual observations of 84×84×3 (width×height×RGB). Their action space is discrete with six options [NIL, UP, DOWN, LEFT, RIGHT, USE]. We provided no action masking, which means the agent had to learn to ignore [UP, DOWN] in Aliens and [USE] in Wait for Breakfast. Fig. 6 shows a learning scene with four parallel game instances and their low-resolution visual observation, built as an executable by Unity.

B. ML-Agents Training Parameters

We used the default ML-Agents hyperparameter settings as a comparison between Proximal-Policy Optimization (PPO) [22] and the GVGAI Gym results by Torrado et al. [19]. The parameters with default values and the recommended ranges[14] noted in [brackets] listed below:

- Gamma: 0.99 [0.8 - 0.995]
- Lambda: 0.95 [0.9 - 0.95]
- Batch Size: 1024 [512 - 5120]
- Buffer Size: 10240 [2048 - 409600]
- Number of Epochs: 3 [3 - 10]
- Learning Rate: 3.0e-4 [1e-5 - 1e-3]
- Time Horizon: 64 [32 - 2048]
- Max Steps: 1.0e6* [5e5 - 1e7]
- Beta: 5.0e-3 [1e-4 - 1e-2]
- Epsilon: 0.2 [0.1 - 0.3]
- Normalize: false [true/false]
- Number of Layers: 2 [1 - 3]
- Hidden Units: 128 [32 - 512]

One exception noted with an asterisk (*) in the list above is the Max Steps setting, which was increased to one million, making it comparable with the GVGAI Gym learning results. Each game was evaluated using both a one instance and an eight instance scene, noted as PPO (1) and PPO (8) in the results section.

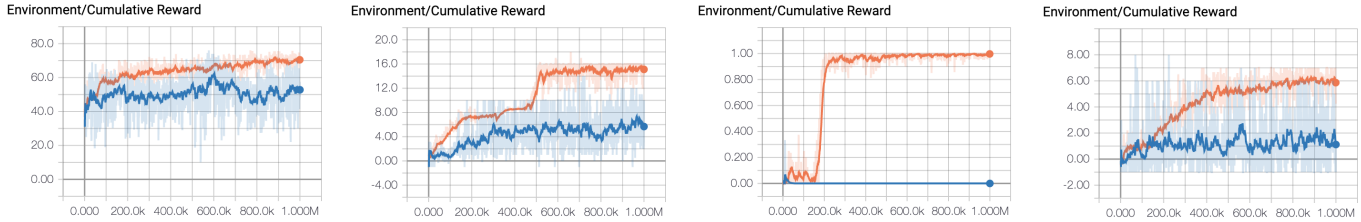
V. Results

ML-Agents allows agents to train either directly in the Unity editor, or using the faster option of a compiled executable. The speed of the training is dependent on the number of rules and size of the VGDL game. As an example training with eight game instances of Zelda, took around an hour per 100,000 steps on a high end 2018 MacBook Pro. No multi-threading was implemented for the game instances inside Unity. Fig. 7 shows the learning progress on each of the four games with one and with eight game instances running in parallel for one million steps. Running eight instances in parallel consistently performs better than a single instance for two reasons. First, more instances lead to more exploration

[12]The game description in UnityVGDL has been modified from the GVGAI implementation by adding a reward for exiting the level.

[13]Only six of them visualized in the build, the last two are rendered off-screen.

[14]https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md



(a) Aliens (b) Boulder Dash (c) Wait for Breakfast (d) Zelda

Fig. 7. Shows the training progression on the four games using ML-Agents in the UnityVGDL environment. Mean cumulative training reward across eight game instances PPO (8) orange, and one instance PPO (1) blue. The x-axis denotes steps. Note that rewards and the y-axis are different for each game. The lines are smoothed by 90% of the previous value, and the transparent outlines behind them are the actual means. The results clearly show learning progress across all four games, in line with the baselines from [19].

TABLE I
SCORE COMPARISON

Games	Random Agent	Best Planning Agent	GVGAI Gym Learning			ML-Agents Learning	
			DQN	PDDQN	A2C	PPO (1)	PPO (8)
Aliens	52	80.4	75	74	77	49.25 (40%)	71.35 (70%)
Boulder Dash	1.4	16.4	2.5	5	15.5	5.05 (0%)	15.95 (0%)
Wait for Breakfast	0	1	1	1	1	0 (0%)	1 (100%)
Zelda	-0.3*	7.6	4.2	4.2	6	1.4 (0%)	6.3 (0%)

Table I shows the results from [19] compared to final ML-Agents PPO scores averaged score over 20 runs with max step limit set to 1,500. Win-rates shown in parenthesise, i.e. percentage of runs in which the termination objective resulting in a win. Number of instances for PPO denoted in parenthesis.

during training. Secondly, in ML-agents steps are counted by the academy (the focal point for training agents in a game scene), which means the experience count is multiplied by the game instance count.

It is clear from the results that running a single instance with the default parameters is inferior to most other approaches. Preliminary experiments with a variation of tuned parameters have shown potential for better learning. We recommend tuning the hyperparameters or increasing the number of parallel training instances to achieve good results. Running learning on four games is not enough to determine how good the default PPO is compared to other algorithms, but these are baseline results, that give an indication of what is possible using ML-Agents with UnityVGDL.

A. Comparing with GVGAI Gym Results

The learning results of DQN, PDDQN, and A2C using GVGAI Gym should be directly comparable to the results of ML-Agents PPO if the framework is implemented correctly. Although the interpretation of VGDL is not guaranteed to be the same, the UnityVGDL interpretation should match JavaVGDL. The results are not meant to show that one algorithm or framework is superior to the other, but that the baseline results of UnityVGDL are comparable to those found by Torrado et al. The four games were selected exactly because agents can learn to play them well with the baseline algorithms. Finding vast discrepancies between the learning results would indicate an implementation issue or limitation in either framework.

Table I shows the results from Torrado et al. [19] for a random agent[15], the best planning agent score, GVGAI Gym Learning agents and the UnityVGDL ML-Agents score. The final ML-Agents scores were captured for each trained brain by averaging the score of 20 runs of each game with a max step limit of 1,500. Win rates, i.e. the percentage of the 20 runs in which the termination condition resulting in a win, are also shown. It is interesting to note here that no agent managed to achieve a win in neither Zelda nor Boulder Dash. The win condition in these games depend on specific actions, which makes them harder to reach. Like having the key in Zelda, or having gathered exactly ten diamonds in Boulder Dash, prior to reaching the door in either. The baseline GVGAI Gym and ML-agents results are very similar across all games (Table I). Just like the single instance DQN and PDDQN perform worse than A2C, single instance PPO performs worse than eight instance PPO.

VI. DISCUSSION

This paper introduced a new framework that combines the GVGAI VGDL ontology with ML-Agents in Unity. The comparison between learning results in UnityVGDL and GVGAI indicates that there are no major discrepancies between the two. As expected the learning results on the selected games were good, but many harder challenges exist in VGDL games. The learning agents have yet to train on more than a single level of a single game for the baseline results.

The hyperparameter values are meant to be adjusted when using ML-Agents, but the default values were kept except the number of max steps, which was adjusted for comparison with

[15]The score of -5.2 reported by Torrado et al. [19] for a random agent on Zelda must be an error, as Zelda never yields a score below -1.0. A new evaluation was made by averaging the score over 25 runs in GVGAI. The updated score has been marked in the table with an asterisk (*).

GVGAI Gym [19]. The default values provided in the ML-Agents framework have been created to provide the best results when learning from vector observations with continuous action spaces. The UnityVGDL implementation differs in both respects, as it only uses visual observations and has a discrete action space.

Hyperparameter tuning could provide even better learning conditions. Similarly, the imposed limit of 1,500 steps per agent episode was chosen because Wait for Breakfast has a built-in limit of 1,500 steps. During training, the agents would often perform some initial actions and then go wait in a corner for the episode to end. A lower step limit would most likely result in faster real-time training with similar results. Avatars in the GVGAI ontology can limit available actions, e.g. in Aliens the FlakAvatar actions are LEFT, RIGHT and USE. Action masking can be beneficial to limit the state-action space, when learning or planning. However, action masking was not implemented and therefore the neural networks had to learn that some actions did not have any effect on the game. It is unclear whether Torrado et al. removed unavailable actions from the action space of their learning agents. We plan to add action masking to UnityVGDL, which should require minimal changes but could improve learning speed in games with fewer available actions.

A. Adaptations

To maintain compatibility to JavaVGDL, UnityVGDL has inherited parts of the GVGAI framework. Large parts of the GVGAI framework are related to the actual competitions and not part of the VGDL core, so they have not been ported. Some support functionality has been added and made slightly more flexible, like parsing colors and names of fields and classes. UnityVGDL does not use a type lookup system. Instead, classes and fields are looked up using .NET Reflection. Reflection makes it easy to add new effects, sprite types and termination conditions by simply inheriting from another class in the ontology. Another significant change is the usage of `stypes` directly instead of using a registry index, which initially eased implementation. This change could become a performance bottleneck that has to be addressed in the future. UnityVGDL implements a forward model, but it is currently unused, except for relaying game state information to the ML-Agents (for step reward calculation). The VGDLPlayer player interface from JavaVGDL has been kept, albeit it creates a less than elegant implementation of the ML-Agents Agent class; it will most likely be changed in the future.

B. Limitations

Not all sprite types and effects have yet been ported from JavaVGDL to UnityVGDL. The porting process is relatively painless as Java and C# are very similar and most functionality in `VGDLGame` has kept their names in the porting process. In grid-based games path planning has yet to be implemented. Two-player games are only partially implemented, and will not currently work. Continuous physics games have been consciously left as future work. The VGDL implementation should also allow vector observations for VGDL agents as an alternative to visual observations. GVGAI supports using an observation grid containing sprite ids for each square on the grid. A similar approach could be used for vector observations in ML-Agents.

VII. Future Work

A. Missing Features

The relatively small task of porting the remaining class functionality in the ontology remains. All ontology classes are instantiated with the correct data and registered by the parser in `VGDLGame`. Functionality for some rarely used types still needs to be ported from JavaVGDL. For now, the main focus has been on getting the core functionality working and testing it on a small but diverse set of games. For sprites, the general functionality for sprite animations and auto-tiling (automatically choosing sprite from a list based on level map) has not been implemented, and collisions are less optimized than in JavaVGDL. As mentioned earlier, vector observations have also been left for future work.

B. Utilizing the Unity Engine

While a port of the grid-based VGDL engine of JavaVGDL is sufficient for grid physics games, continuous physics games would profit from being translated into native Unity scene hierarchies of game objects that interact with each other based on the VGDL description. This will require the interpretation and custom implementation of generalized VGDL `MonoBehaviours`[16] that can be attached to the game objects based on the VGDL description. Creating a custom implementation of the physics types from VGDL would create a new interesting challenge for transfer learning research.

Generating a custom Unity scene from a VGDL description would also enable Unity game developers to use VGDL to prototype game ideas. Extending the use of VGDL beyond the scope of computational intelligence research would benefit both game developers and researchers. Developers would benefit from a highly abstract description language for creating game prototypes and researchers could gain access to more complex games. A community website like Puzzlescript.net and an extensible ontology could become a rich resource for data and knowledge exchange.

C. New Opportunities through ML-Agents

There are several different avenues to explore using UnityVGDL's ML-Agents setup. The ML-Agents toolkit contains an easy-to-use implementation of a curiosity module, a memory module, and imitation learning. Curiosity has shown good results on games with sparse rewards [23]. VGDL games such as Zelda and Boulder Dash, for which the learning agents did not achieve the win-condition might benefit from curiosity. The memory module is a recurrent neural network (LSTM [14]), which allows agents to use knowledge from previous states when evaluating new states. Agents might

[16] https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

use this capability to figure out when they collected enough diamonds in Boulder Dash to head to the exit. Lastly, imitation learning can be used to mimic behavior, and could in the future be used as a starting point for reinforcement learning agents. The combination of these capabilities makes UnityVGDL and ML-Agents an appealing playground for research.

D. Procedural Content Generation with UnityVGDL

In GVGAI VGDL has been used extensively for procedural content generation (PCG) competitions [20]. Both level generation from known game rules and rule generation from known levels have been a part of the competition for several years. Outside of competitions, agents trained in procedurally generated environments [24], instead of only a particular one, have shown better generalization abilities in various different domains and also have a lower chance of overfitting [25]–[27]. Using UnityVGDL for procedural content generation is thus an excited future possibility.

VIII. Conclusion

The UnityVGDL framework opens new possibilities for competitions in General Video Game Playing [10], research, and general game development. UnityVGDL turns VGDL into a prototyping tool for game developers and a research tool. We would like to encourage the community to help complete and expand the UnityVGDL framework.

Acknowledgements

We would like to thank Niels Justesen, Miguel Gonzlez and Djordje Grbic for fruitful discussions and insightful comments on the framework presented in this paper.

References

[1] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A General Platform for Intelligent Agents. *arXiv:1809.02627 [cs, stat]*, September 2018.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[4] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. *arXiv:1605.02097 [cs]*, May 2016.

[5] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Vilamoura-Algarve, Portugal, October 2012. IEEE.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[7] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 2019.

[8] Sebastian Risi and Julian Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2015.

[9] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning. *arXiv:1902.01378 [cs]*, February 2019.

[10] John Levine, Clare Congdon, Marc Ebner, Graham Kendall, Simon Lucas, Risto Miikkulainen, Tom Schaul, and Tommy Thompson. General Video Game Playing. *Dagstuhl-Followups*, 6:77, November 2013.

[11] Tom Schaul. An Extensible Description Language for Video Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):325–331, December 2014.

[12] Tom Schaul. A video game description language for model-based or interactive learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pages 1–8, Niagara Falls, ON, Canada, August 2013. IEEE.

[13] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couetoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, September 2016.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[15] Diego Perez, Spyridon Samothrakis, and Simon Lucas. Knowledge-based fast evolutionary MCTS for general video game playing. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8, Dortmund, Germany, August 2014. IEEE.

[16] Raluca D. Gaina, Adrien Couetoux, Dennis J. N. J. Soemers, Mark H. M. Winands, Tom Vodopivec, Florian Kirchgesner, Jialin Liu, Simon M. Lucas, and Diego Perez-Liebana. The 2016 Two-Player GVGAI Competition. *IEEE Transactions on Games*, 10(2):209–220, June 2018.

[17] D. Perez-Liebana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas. Introducing Real World Physics and Macro-Actions to General Video Game AI. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 248–255, August 2017.

[18] Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. Matching Games and Algorithms for General Video Game Playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, September 2016.

[19] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. Deep Reinforcement Learning for General Video Game AI. *arXiv:1806.02448 [cs, stat]*, June 2018.

[20] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. General Video Game AI: A Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms. *arXiv:1802.10363 [cs]*, February 2018.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, MA, second edition edition, 2018.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, July 2017.

[23] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. *arXiv:1705.05363 [cs, stat]*, May 2017.

[24] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.

[25] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *NeurIPS 2018 Workshop on Deep Reinforcement Learning*, 2018.

[26] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

[27] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

9.2 Squeezer - A tool for designing juicy effects

Squeezer - A Tool for Designing Juicy Effects

Mads Johansen
IT University of Copenhagen
Copenhagen, Denmark
madj@itu.dk

Martin Pichlmair
IT University of Copenhagen
Copenhagen, Denmark
mpic@itu.dk

Sebastian Risi
IT University of Copenhagen
Copenhagen, Denmark
sebr@itu.dk

ABSTRACT

This paper introduces *Squeezer*, a tool for designing juicy effects in the Unity game engine. Drawing upon inspiration from sound effect synthesizers and description languages, Squeezer can "synthesize" common types of juice effects, by combining simple effects into effect sequences. These effect sequences can be edited and executed even while playing the game in the editor, lowering iteration times and easing the exploration of effects. We make a preliminary usability test to verify the functionality and scope of the tool. Squeezer is available at: https://github.com/pyjamads/Squeezer

CCS CONCEPTS

• **Applied computing** → **Computer games**; • **Human-centered computing** → **Graphical user interfaces**; Usability testing.

KEYWORDS

Game Development; Game Design; Juice Effects; Interaction Feedback; Toolkit; Prototyping; Generator

1 INTRODUCTION

In this work, we present a tool for assisting game designers in applying common juice effects to game prototypes. Designing prototypes is a very common practice in game design. Their purpose is to explore a design space or to communicate a game mechanic. Squeezer seeks to enrich prototypes with Game Feel [25] by adding juice [13]. We aim to create a way for designers to quickly find or generate effect sequences that are “good enough” for their prototyping needs. This kind of tool that is not currently available.

Squeezer can generate various juice effects to quickly and efficiently find effect sequences that can serve as "good enough" in the prototyping stage of game development. The goal of prototyping is always to verify or communicate concepts and ideas; in prototyping, the faster the design is revealed to fail or succeed, the better. As the preliminary user study suggests, Squeezer can help to quickly determine which kinds of juice effects detract from and which enhance certain features in a game prototype.

To guarantee practical application and good test cases, we choose to develop the tool in the widely used Unity[1] game engine. Unity lets users quickly develop and integrate custom tools to extend their editor. Those tools can even be sold commercially via the Unity Asset Store[2]. Users are used to adding extra libraries to their projects to extend the functionality of the game engine and its editor. By developing Squeezer for Unity, we increase the real-world application probability and the number of available expert users.

The three main parts of Squeezer are (1) trigger setup, (2) effect sequencing, and (3) effect execution. Effects are triggered by a simple event system that ties into the prototype’s code. Effects are sequenced by structuring them into a tree and using relative time offsets (delays). The execution of effects is managed by a simple Tweening [3] system that schedules effects and continuously updates ongoing effects after they are triggered. The word tween comes from "in betweening" [21], which comes from cartoon animation, where a senior would draw keyframes of animation sequences, and juniors would then fill in the timelines between those keyframes. A tweening system interpolates over a duration between a beginning and end value (also known as keyframes). The interpolation can be linear or eased in and/or out using easing curves[3].

Importantly, Squeezer is more than a tweening system: the descriptions allow runtime manipulation and fast iteration on ideas. Additionally, with the export and import of full or partial descriptions, users can easily create a library of effect sequences and apply them widely in or between projects. Squeezer also includes complex effects such as the SFXR audio synth effect and the spawner effects which create objects and build initial effects sequences for their offspring, allowing users to alter them easily.

This paper has two contributions. The first one is the introduction of Squeezer, a new tool for exploring juice effects during game prototyping. It combines structure and ontology ideas from the Video Game Description Language (VGDL) [17] with Unity editor integration for quick iterations. The second contribution is the idea of a juice effect "synthesizer", combining modular sequencing and presets to generate complex effect sequences, based on categories similar to those used in SFXR [20].

2 BACKGROUND

Around a decade ago, the concept of game feel and juiciness gained traction after being discussed mostly for prototyping [9] in the indie game community. Juice is a game design term for abundant feedback that amplifies interactions related to input and other in-game events [10, 15]. It is superfluous from a strictly mechanical perspective but makes interacting with the system more pleasurable. Juice helps sell the illusion that the game world has real properties, just like exaggeration in cartoons create the illusion of life [26]. Hunicke [11] says *"Juiciness can be applied to abstract forms and elements and it is a way of embodying arbitrarily defined objects and giving them some aliveness, some qua, some thing, some tenderness."*

The term ’Game Feel’ was coined by Swink, who first wrote an article [24] and later a book on ’Game Feel’ [25]. One part of designing of game feel Swink calls ’polish’, which is also seen in other academic contexts [8, 16] described as *the impression of*

[1] https://unity.com/
[2] https://assetstore.unity.com/
[3] see examples on http://easings.net/

physicality created by layering of reactive motion, proactive motion, sounds, and effects, and the synergy between those layers. Which has a remarkable resemblance to Hunicke's description of 'juiciness'.

Both game feel and juiciness have been widely discussed in the game development community, leading to developers like Jonasson & Purho [13] as well as Nijman [18] to talk about juice and game feel from their respective points of view. Swink's book and the talks by Jonasson & Purho and Nijman are still the primary sources for introducing the concepts of game feel and juiciness. Although all three had excellent demos available when they were released, these resources have unfortunately since deteriorated or disappeared. However, more recent projects such as MMFeedbacks [7] and Game Maker's Toolkit [1, 2] continue the work on this topic.

With MMFeedbacks, Forestié has created an expert tool for adding juiciness to Unity games. However, MMFeedbacks handles triggering, delaying, and sequencing effects, but subsidizes designing some effects to various subsystems in Unity. For instance, the user still needs to understand the Unity particle system and create the effect they want with that system. This design choice makes sense for MMFeedbacks as an expert tool because the Unity subsystems (Particle system, Cinemachine, Timeline, Animator, and more) are powerful for their specific purposes. However, an inexperienced user or a designer who is sketching out part of a game, might not have the skill or the time to use MMFeedbacks meaningfully in this context.

On the other hand, the sound effect tool SFXR [20] has long been used heavily for prototyping and game jamming purposes [9], to lift the appeal of a prototype with "good enough" placeholder sound effects. Pettersson originally developed SFXR in 2007 for game jam participants, to

> *...provide a simple means of getting basic sound effects into a game for those people who were working hard to get their entries done within the 48 hours...*

SFXR is a procedural content generator (PCG) and a synthesizer. It is operated simply by selecting a category of sound effect, and pressing the category button repeatedly until the user hears a desirable sound effect. Apart from the main sound generation, the user can also manually tune each of the more than twenty different parameters, or mutate all parameters a small amount by the click of another button. The categories read as follows [Coin/Pickup, Laser/Shoot, Explosion, Power-up, Hit/Hurt, Jump, Blip/Select]. The categories act as presets, explicitly setting some parameters, limiting others to preset ranges, and randomizing the rest. The sounds generated by SFXR can then be inserted as placeholders until a game or prototype is mature enough to get a sound designer involved or a sound pack implemented. For a game designer, applying placeholders can often reveal which effects and mechanics they can lean into or should steer clear off. Thus to create a sound effect that works with their particular game, designers might have to generate 10-20 different sound effects to find a suitable one. Similarly, finding suitable effects for the remaining aspects of feedback for a game prototype requires a lot of trial and error. The tool proposed here makes trial and error faster and simpler by allowing the designer to test effect sequences triggered by events in the game, similarly to MMFeedbacks, but with a stronger sense of proceduralism for generating common types of effect sequences.

3 IMPLEMENTATION

A simple breakout clone provided a point of reference and informed the development of Squeezer. Fig. 1 shows the basic game and setup window in the top left. After setting up the triggers the top right shows the game with a bit of color and an initial generated effect sequence for block destruction. The bottom left shows the foldout menu for editing parameters of a color tween effect. And along the bottom towards the right you can see various effects executed over a single play session. In the juiced versions Squeezer is adding, effects such as ball trail, impact squashing, sound effects, block shattering, explosions, color changes, simple starting animations, and time dilation effects. Fig. 1 shows a potential workflow for Squeezer, but designers usually work iteratively, repeating step one to three for each class of objects.

One consideration made early on, was to implement a simple tweening system, for scheduling and executing effects. The reason for doing this was not to rely too heavily on Unity features, to allow the open-source community to adapt the code to other engines more easily. The class structure used within Squeezer, is almost entirely pure C# classes without Unity dependencies except for Random, and Unity specific effect logic.

3.1 Analysis

The initial problem of triggering effects based on events or interactions has been solved many times in the past. One solution is a hierarchical description approach, including an ontology, such as PyVGDL, JavaVGDL, and UnityVGDL [12, 19, 22]. These VGDL frameworks execute entire games based on this structure and effectively hide complexity in the descriptions with the provided ontology. However, effect sequences for juice effects additionally require scheduling, both sequentially and simultaneously, which adds the need for more complex nesting of effects.

Apart from the structure of descriptions, analyzing which events commonly trigger juice effects is needed. As well as what the kinds of effects most frequently used are, how they get applied and where. By reviewing industry talks [13, 18], we identified an initial set of triggers, those are: **OnStart**, triggered when creating an object/when the game starts. **OnCollision**, triggered when a collision occurs. **OnMove**, triggered while moving or changing movement state. **OnRotate**, triggered when the object rotates in some way. **OnDestroy**, triggered when destroying an object. **OnDisable**, triggered when an object becomes disabled (a common way of "destroying" objects, without invoking garbage collection in Unity). **OnCustomEvent**, triggered when the system receives a custom event (e.g., Shoot, Jump or when some effect terminates like FadeInComplete).

We also identified a few different groups of effects. Sound effects, color effects, particle and trail effects, transform effects (translate, rotate, scale), time dilation effects, flashing effects (full-screen or localized), wiggle (a combination of several transform effects) and shake (quick random translations) effects. However, common for all effects is that they can be delayed, can be applied to various targets, can be sequenced (scheduled in relation to other effects), and can be independent of in-game time. Apart from those common properties, feedback effects tend to be durational, most common are tween effects. Tweening [21] moves a value, between a start and

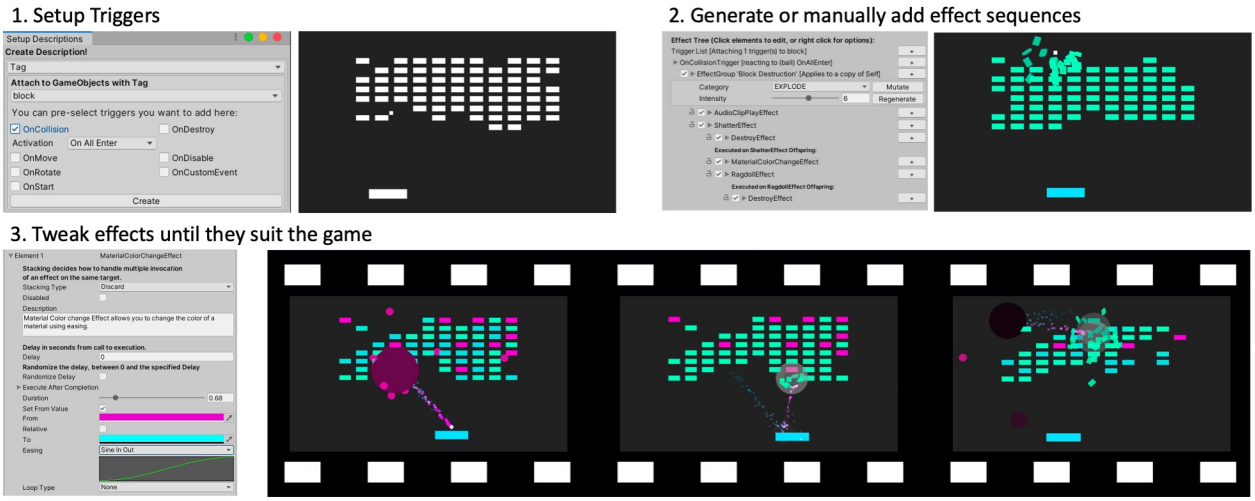


Figure 1: Step (1) take a game prototype and setup the Squeezer event triggers using the setup window. Step (2) add some color and generate or manually add initial effect sequences. Step (3) explore or design parameters of individual effects until they suit the game mechanics.

end value, using an easing function. The simplest easing function is linear interpolation, but to simulate acceleration and deceleration when starting and stopping easing curves such as a sine wave or an exponential function can be used instead. Another class of effects is the ones that spawn additional objects in the game instead of manipulating objects that already exist. Trails and other particle effects that generate objects will contain an additional list of effects administered to their offspring. The four types of effects are *One shot (can be delayed), Durational, Tween, and Spawner effects*. Lastly, we look at the relationship between the trigger and the affected objects. The most common target is the object that detected the event itself. However, other targets include: objects of a certain type or with a certain tag, the other object in a collision, specific objects (e.g., the camera) and editor values (e.g., time scale).

3.2 Implementation details

Based on the analysis above and working with a hierarchical approach, broken down from root to leaf, Squeezer's functionality is as follows: **Description**, in charge of attaching triggers to actual game objects in the game, contains a list of Triggers. **Trigger**, managing which events cause effects to occur, selected from the seven identified trigger types, contains a list of effect groups. **Effect Group**, determines which game objects the effects will be executed and contains a list of effects. **Effect**, includes functionality to execute the effect itself, and contains a list of effects to apply on completion (spawner effects also contain a list of effects to run on any generated objects).

Squeezer addresses two user interaction aspects apart from the descriptions. One is a setup window, easing the process of setting up the descriptions initially. The second is shorter iteration cycles, by allowing persistent editing while playing. To facilitate editing while playing, Squeezer has a Step-Through Mode feature inspired by the Klik'n'Play[4] feature of the same name. This mode pauses the game automatically when collisions or other selected events occur, allowing users to add or modify effects as the game plays out. Step-Through Mode also includes a random effect sequence generator called "Assist Me", which generates a random set of up to five effects. The randomness was intended as a proof of concept, for more advanced generation features later on. However, the feature generated a surprising amount of exciting combinations during development.

A few built-in effects, like *TrailEffect* and *ShatterEffect*, combine their spawning logic with other effects such as color-changing and destruction. However, you can create very complex effects, with the building blocks in Squeezer. Imagine a vehicle exploding. First, we could add a positional flash and a sound effect and then "shatter" the object. This debris could then fly off, and after a while, they explode, making a small positional flash and sound. The initial explosion could also be extended, by adding several flashes, scale them in/out to simulate smoke and shaking the camera.

3.3 Secondary analysis

After the initial implementation of a random effect sequence generator, the need arose to consider implementing categories of generated effects. Using SFXR's seven different categories as a starting point, we found the following initial set of categories: Pick-up, Destroy/Explode, Jump, Shoot, Hit/Hurt, Interact/Use, Projectile move and Player move. The only additions being Player and Projectile move for continuous triggers, which have no counterpart in SFXR.

4 USER TEST

We conducted a preliminary user test divided into three parts; a briefing 15 minutes, the user test 30-60 minutes, and lastly, the participants answered a set of 15 questions about their experience.

[4]Klik'n'Play by Clickteam https://knpforschools.webs.com/

The briefing included how to set up and interact with Squeezer and introduce the breakout clone we provided as an example game. The participants were first shown a version that showcased most of the available effects, and then for the actual test, they were provided a completely juice free version of the game. The participants were told to spend 30-60 minutes adding any effects they saw fit, exploring the possibilities of Squeezer. During this part of the test, their usage was recorded anonymously, and any bugs and user experience issues encountered were logged by the authors. The participants had the option of asking for clarification on anything and which effects to use to achieve specific ideas.

5 RESULTS

A video and gif showcase of Squeezer and the breakout example game is available in the repository[5].

5.1 Preliminary Qualitative User tests

We tested Squeezer with four users ranging between two and ten years of experience using Unity. Out of the four participants, only one had recently been in charge of implementing game effects. When prompted on how they would usually mock-up juice effects in games, they all replied they would make small scripts or use the Unity Animator for simple things, and use a Tweening library for more complicated effects. Each participant found novel effect combinations that resulted in a very different look and feel for the same basic breakout clone. All four users claimed they would love to use this kind of tool for testing out ideas or during game jams. One participant, who also teaches game design to students said:

> *I would also definitely give this to my students when talking about game feel and juice. I think letting them play with these effects would be a nice, time-efficient way of getting to experiment with juice and exploring how it changes game feel.*

And another said:

> *I would be interested in using this in small experiments and at game jams. I could also see it being useful as a communication tool on teams, using the tool to quickly demonstrate various intents.*

5.2 Participant usage

One participant decided to manipulate gameplay elements. They added a resource management layer to the breakout game by modifying the size of the paddle, making it smaller when it moved, and larger again as the ball hit blocks. These modifications of the gameplay made the player ration their paddle movements. Three out of the four users decided to go for many of the elements touched upon by Jonasson & Purho [13], such as adding screen shake, sound effects, block destruction by shattering and scaling various objects up/down due to interactions. One participant tried to make a color-changing effect repeat indefinitely. However, due to a limitation in the Tweening effects, the blocks stopped changing their color after five seconds. Another participant got around this limitation by tying the color-changing to the OnMove event of the paddle, which essentially restarted the effect whenever it would end.

[5]https://github.com/pyjamads/Squeezer/tree/master/Showcase

We asked the participants, 'Which features and effects did you find most useful?' one participant noted:

> *Screen shake, shatter, trail, and color changes were easy and powerful to apply. It felt like it would save me a significant amount of work if I were prototyping and e.g., at a game jam, this would be useful to throw in some nice effects quickly.*

while another said:

> *The sound effect, it just added life.*

6 FUTURE WORK

The current Squeezer prototype is just a first step towards a more powerful tool for designing juicy effects. We plan to continue exploring how to best present the effect sequences and other user experience elements with the aid of more user testing:

- Adding more powerful generator options that can add effect sequences based on selected categories is an important next step. The current "assist me" feature is the first step, but unfortunately, our user study participants did not get to use it due to the current interface.
- For more straightforward sequencing, we will explore a timeline visualization, and a way to preview the effect.
- Squeezer can collect anonymous usage data. We will explore improvements to automatically create categories and effect sequences based on usage data from our user tests.
- We will be looking towards interactive evolution and expressive range analysis, exploring approaches similar to Picbreeder [23] and Danesh [6] but applied to effect sequences.
- Another aspect we would like to explore is automated game design [4, 5], and how to use code to provide additional context to a system generating effects for a prototype.

7 CONCLUSION

While still a work in progress, Squeezer is a promising juice "synthesizer" that can create a wide range of different effect sequences by combining more than twenty different effects. Our initial user test confirmed that there is an interest and a place for a tool that simplifies creating juice effects, both as a tool for learning, game jamming, and ideation [14].

REFERENCES

[1] Mark Brown. 2015. Secrets of Game Feel and Juice.
[2] Mark Brown. 2019. Why Does Celeste Feel So Good to Play? | Game Maker's Toolkit.
[3] N. Burtnyk and M. Wein. 1971. Computer-Generated Key-Frame Animation. *Journal of the SMPTE* 80, 3 (March 1971), 149–153. https://doi.org/10.5594/J07698
[4] Michael Cook. 2017. A Vision For Continuous Automated Game Design. *arXiv:1707.09661 [cs]* (July 2017). arXiv:1707.09661 [cs]
[5] Michael Cook. 2020. Software Engineering For Automated Game Design. *arXiv:2004.01770 [cs]* (April 2020). arXiv:2004.01770 [cs]
[6] Michael Cook, Jeremy Gow, and Simon Colton. 2016. Danesh: Helping Bridge The Gap Between Procedural Generators And Their Output. (2016), 16.
[7] Renaud Forestié. 2019. How to Design with Feedback and Game Feel in Mind - Shake It 'til You Make It.
[8] Tracy Fullerton. 2014. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games* (3rd ed.). A K Peters/CRC Press.
[9] Kyle Gray, Kyle Gabler, Shalin Shodhan, and Matt Kunic. 2005. How to Prototype a Game in Under 7 Days.

[10] Kieran Hicks, Patrick Dickinson, Juicy Holopainen, and Kathrin Gerling. 2018. Good Game Feel: An Empirically Grounded Framework for Juicy Design. (2018), 17.

[11] Robin Hunicke. 2009. Loving Your Player With Juicy Feedback.

[12] Mads Johansen, Martin Pichlmair, and Sebastian Risi. 2019. Video Game Description Language Environment for Unity Machine Learning Agents. In *2019 IEEE Conference on Games (CoG)*. 1–8. https://doi.org/10.1109/CIG.2019.8848072

[13] Martin Jonasson and Petri Purho. 2012. Juice It or Lose It. (2012).

[14] Ben Jonson. 2005. Design Ideation: The Conceptual Sketch in the Digital Age. *Design Studies* 26, 6 (Nov. 2005), 613–624. https://doi.org/10.1016/j.destud.2005.03.001

[15] Jesper Juul and Jason Scott Begy. 2016. Good Feedback for Bad Players? A Preliminary Study of 'Juicy' Interface Feedback. In *Proceedings of First Joint FDG/DiGRA Conference*. Dundee, 2.

[16] Lasse Juel Larsen. 2016. Collision Thrills: Unpacking the Aesthetics of Action in Computer Games. *Journal of Computer Games and Communication* 1, 1 (April 2016), 41–52. https://doi.org/10.15340/2148188111997

[17] John Levine, Clare Bates Congdon, Marc Ebner, Simon M Lucas, Risto Miikkulainen, Tom Schaul, and Tommy Thompson. 2013. General Video Game Playing. (2013), 7.

[18] Jan Willem Nijman. 2013. The Art of Screenshake.

[19] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couetoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. 2016. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8, 3 (Sept. 2016), 229–243. https://doi.org/10.1109/TCIAIG.2015.2402393

[20] Tomas 'DrPetter' Pettersson. 2007. SFXR. http://www.drpetter.se/project_sfxr.html.

[21] William T. Reeves. 1981. Inbetweening for Computer Animation Utilizing Moving Point Constraints. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '81*. ACM Press, Dallas, Texas, United States, 263–269. https://doi.org/10.1145/800224.806814

[22] Tom Schaul. 2013. A Video Game Description Language for Model-Based or Interactive Learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, Niagara Falls, ON, Canada, 1–8. https://doi.org/10.1109/CIG.2013.6633610

[23] Jimmy Secretan, Nicholas Beato, David B. D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. 2008. Picbreeder: Evolving Pictures Collaboratively Online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. Association for Computing Machinery, Florence, Italy, 1759–1768. https://doi.org/10.1145/1357054.1357328

[24] Steve Swink. 2007. Game Feel: The Secret Ingredient.

[25] Steve Swink. 2009. *Game Feel*. Morgan Kaufmann.

[26] Frank Thomas and Ollie Johnston. 1981. *The Illusion of Life: Disney Animation*. Abbeville Press, New York.

9.3 Squeezer - A mixed-initiative tool for designing juice effects

Squeezer - A Mixed-Initiative Tool for Designing Juice Effects

Mads Johansen
madj@itu.dk
Center for Computer Games Research
IT University of Copenhagen
Denmark

Martin Pichlmair
mpic@itu.dk
Center for Computer Games Research
IT University of Copenhagen
Denmark

Sebastian Risi
sebr@itu.dk
Center for Computer Games Research
IT University of Copenhagen
Denmark

ABSTRACT

This paper presents a Mixed-Initiative version of *Squeezer*, a tool for designing juice effects in the Unity game engine. Drawing upon sound synthesizers and game description languages, Squeezer can synthesize common types of juice effects by combining simple building blocks into sequences. Additionally, Squeezer offers effect generation based on predefined recipes as well as an interface for interactively evolving effect sequences. We conducted a user study with five experts to verify the functionality and interest among game designers. By applying generative and evolutionary strategies to juice effect design, Squeezer allows game designers and researchers using games in their work to explore adding juice effects to their games and frameworks.

Squeezer is available at: https://github.com/pyjamads/Squeezer

CCS CONCEPTS

• **Applied computing** → **Computer games**; • **Human-centered computing** → **Graphical user interfaces**; Usability testing.

KEYWORDS

Game Development, Game Design, Juice Effects, Interaction Feedback, Toolkit, Prototyping, Generator, Interactive Evolution, Mixed-Initiative

1 INTRODUCTION

Game designers regularly create prototypes as part of their practice. Their purpose is to explore a design space, communicate game mechanics, and collect player feedback on game mechanics. While prototypes often require little to no juice – a game design term for visual and audible feedback and effects, see [24] – to verify if a rule or mechanic works, there are many cases where they depend on placeholder effects to provide enough feedback to make a game mechanic understandable by the player. In all cases, prototypes should be fast to make, which can be supported by tools.

Squeezer [18] assist game designers in prototyping by enhancing game feel [32] through simplifying the process of adding juice [19]. Squeezer can generate and link various juice effects to sequences. This way, designers can quickly and efficiently create rich feedback during the prototyping stage of game development.

In this paper, we expand upon the preliminary version of Squeezer by implementing category-based effect tree generation, mutation of effect trees, and adding an interface for the interactive evolution of effects. Interactive evolutionary computation [33] or interactive evolution builds on an AI-assisted creation loop. An interactive evolution system presents different artifacts to the user, who selects the artifacts they prefer among the selection. The system then generates a new set of artifacts through crossover and mutation. This next generation of artifacts is subsequently presented to the user, and so on. In practice, the user plays the part of the fitness function in the evolutionary process.

In our experiment, we explore if adding interactive evolution to Squeezer is beneficial for assisting designers in the exploration and design of effects for their game prototypes. Interactive evolution has been used to generate sonic and visual artifacts before, in tools such as Picbreeder [28], Endless Forms [7], MaestroGenesis [15], DrawCompileEvolve [36], ZzSsprite [9], or more recently in combination with deep generative methods [2]. It has not been applied to game feel design. In this paper, we present the results of testing two assumptions about the introduction of interactive evolution to Squeezer:

- Adding interactive evolution makes it easier for new users to get acquainted with Squeezer. It provides an alternative introduction to the possibilities of Squeezer[1] by letting users explore effect sequences interactively, based on examples, as opposed to manually testing individual effects one by one or familiarisation through documentation or code.
- Interactive evolution assists users in exploring the design space and in discovering new ideas for effects.

In summary, we show how generative and evolutionary strategies can assist designers and even surprise them by revealing unexplored areas of the design space when applied to juice effect design.

2 BACKGROUND

2.1 Game feel and Juice

Juice is the game design term for abundant feedback that amplifies interactions related to input and in-game events. The concept has gained a lot of traction over the last decade. Originally mostly discussed in the indie game community [19, 21], and in relation to prototyping [12], juice is nowadays being examined by a host of researchers [14, 20, 24]. Juice is superfluous from a strictly mechanical perspective but makes interacting with the system more pleasurable. Juice helps to sell the illusion that the game world has realistic properties, just as exaggeration in cartoons creates the illusion of life [34]. Hunicke states that "*Juiciness can be applied to abstract forms and elements and it is a way of embodying arbitrarily defined objects and giving them some aliveness, some qua, some thing, some tenderness.*" [16]

Juice is but one facet of 'Game Feel', a term that encompasses different kinds of polish of interaction design in games, a concept we reviewed thoroughly in [24]. The term itself was coined by Swink, who first wrote an article [31], and later a book [32] about this

[1]Further supported by a small suite of demo scenes, with effects already implemented, that new users can poke and prod at.

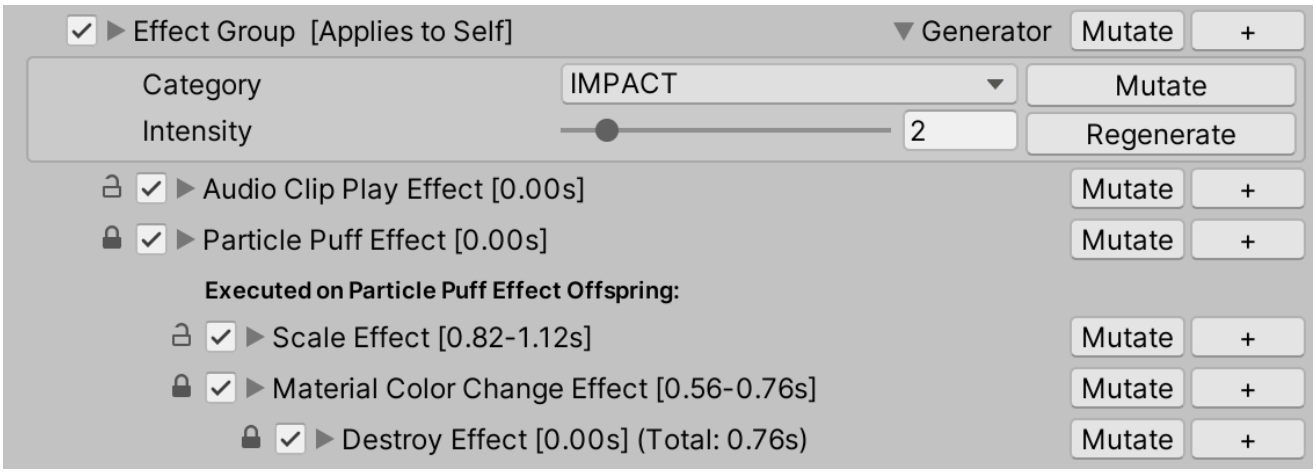


Figure 1: The effect tree generator interface featuring a drop-down menu for selecting a category and a slider for determining the intensity. The mutate and regenerate buttons respect locked parts of the effect tree.

topic. Fullerton describes polishing as "*the impression of physicality created by layering of reactive motion, proactive motion, sounds, and effects, and the synergy between those layers*" [11], an observation remarkably in line with Hunicke's description of 'juiciness'.

Both game feel, and juiciness have been widely discussed in the game development community [24, 30]. Developers like Jonasson & Purho [19] as well as Nijman [21] examine juice and game feel from their respective points of view, offering insights into how crucial this aspect of game design is to practitioners. Swink's book, and the talks by Jonasson & Purho and Nijman, are the primary sources for introducing the concepts of game feel and juiciness. Although all three had excellent demos available when they were released, these resources have unfortunately since deteriorated or disappeared.

New projects, such as MMFeedbacks[2] [10] and Game Maker's Toolkit [3, 4], continue to offer practical and analytical tools for designing game feel. MMFeedbacks is an expert tool for adding juiciness to Unity games. MMFeedbacks handles triggering, delaying, and sequencing effects yet subsidizes designing some of them to various existing subsystems in Unity. For instance, the user still needs to understand the Unity particle system and create the desired effect with that system before integrating it into MMFeedbacks. This design choice makes sense for MMFeedbacks as an expert tool because the Unity subsystems (e.g., Particle system, Cinemachine, Timeline, and Animator) are well suited for their specific purposes. However, an inexperienced user or a designer sketching out part of a game might not have the skills or the time to use MMFeedbacks meaningfully.

2.2 Procedural Content Generation

Procedural Content Generation (PCG) is a collection of methods to automate content generation for games [29]. Instead of manually designing game elements, PCG systems implement design rules for their automatic creation. PCG can be found in a wide variety of areas of asset production. Hendrikx et al. [13] map the field in their survey paper and provide an overview of common applications. Modern game engines like Unity support some procedural content generation systems out of the box (for example, terrain generation), but most PCG systems have to be either acquired or custom-built for each game. It is important to note that there are run-time and design time (offline) applications of procedural content generation. Often resource-intensive algorithms are run at design time. If curation is a part of the design process, the system also needs to be run at design time. An example of a tool working like this is SFXR, a sound effect tool [23].

SFXR has been used for a long time to lift the appeal of prototypes and game jam games [12] via "good enough" placeholder sound effects. Pettersson originally developed SFXR in 2007 for game jam participants to "*provide a simple means of getting basic sound effects into a game for those people who were working hard to get their entries done within the 48 hours.*" [23]

SFXR is a procedural content generator also known as a synthesizer. It is operated simply by clicking a button with a category name and pressing it repeatedly until the user hears a desirable sound effect. Apart from the sound generation, the user can also manually tune each of the more than twenty different parameters or mutate all parameters. The categories read as follows: [Coin/Pickup, Laser/Shoot, Explosion, Power-up, Hit/Hurt, Jump, Blip/Select, Random]. The categories act as presets, explicitly setting some parameters, limiting other parameters to specified ranges, and randomizing the rest. The sounds generated by SFXR can then be inserted as placeholders in a game or prototype until it is mature enough to get a sound designer involved or a sound pack integrated. For a game designer, applying placeholders can often reveal which mechanics they can build on and which they should remove. Designers often have to generate and try out 10-20 different versions to arrive at a sound effect that works in a particular situation. An interface that supports rapid iteration can thus speed up design significantly.

Squeezer supports rapid iteration by combining generative properties like those exhibited by SFXR, the simplicity of effect design

[2]https://feedbacks.moremountains.com/

such as in Particle FX Designer[3], and the universality of a feedback system like MMFeedbacks. Additionally, mutation and interactive evolution allow Squeezer to function as an effect sequence exploration tool.

3 IMPLEMENTATION

Prototyping is the phase of game development where iteration happens the fastest. The game modeled in the prototype is usually small – a single game mechanic, rule, or interaction. Prototyping is part of the exploration phase of design and is used to gain insights into the design space. Squeezer supports this exploration with the assistance of interactive evolution. This mechanism can be used to narrow down the possible design solutions or to arrive at new ideas. Squeezer has the power to surprise the user and lead them to design inspiration.

To encourage practical application and get access to high-quality test cases, Squeezer works with the widely used Unity[4] game engine. Unity lets users quickly develop and integrate custom tools to extend their editor. These tools are also sold commercially via the Unity Asset Store[5]. Users are accustomed to adding extra libraries to their projects to extend the functionality of the game engine and its editor. By developing Squeezer for Unity, we increase the probability of real-world applications and the number of potential users. However, the core functionality of Squeezer does not require Unity, and the underlying open source project can be adapted to other game engines.

3.1 Core Functionality

The three core elements of Squeezer are (1) triggering effects, (2) effect sequencing, and (3) effect execution. Effects are triggered by a simple event system that ties into the prototype's code or game events. The section about integration describes how to achieve that.

Effects are sequenced by structuring them into a tree and by using relative time offsets (delays) as seen in Fig. 3. Each node in the tree will execute all child nodes upon completion. The tree is displayed in its collapsed form, similar to the description language VGDL [27], to increase readability and simplicity. Effect nodes can be expanded to view and adjust their properties. They can also be mutated randomly to explore alternate variations of an effect quickly.

The executor manages the execution of effects and is in charge of scheduling and continuously updating active effects after they are triggered. Effects track their progress internally, letting the executor know when they finish and can be removed from active effects. When an effect ends, it will clean up after itself and queue all subsequent effects with the executor.

The executor is implemented like a simple tweening [5, 25] system, designed to handle one-shot as well as durational effects. Tweening systems are commonly used to drive simple animations and effects within game projects. Most tweening system implementations leave the interpolation to the individual tween classes, like Squeezers executor, which calls the active tweens on every update.

[3]https://codemanu.itch.io/particle-fx-designer
[4]https://unity.com/
[5]https://assetstore.unity.com/

Yet Squeezer is more than a tweening system: the effect sequence descriptions allow run-time manipulation and fast iteration on ideas. The interactive evolution interface allows exploration of the range of possible effects and combinations with very little knowledge of how the system works. The more advanced Inspector UI (see Fig. 3 and 1) can be used manually or in conjunction with interactive evolution, providing additional control and manual manipulation of values. Additionally, with the export and import of full or partial descriptions, users can easily create a library of effect sequences and apply them widely in a project or between projects. Squeezer also includes complex effects such as the SFXR[6] [23] audio synth.

3.2 Integration

In addition to the core elements, Squeezer provides several layers of user interfaces and code interfaces (Application Programming Interfaces or APIs) that allow designers and game programmers to integrate the tool exactly where and how they want. A setup window to help setting up the framework (step one in Fig. 2), where the user can select game objects or tags and the event triggers they would like effects to respond to. The inspector interface, with its simplified tree view providing a simple visual interface to generate, mutate, add and remove elements from effect sequences (step two in Fig. 2). As well as the expanded effect editor (step three in Fig. 2) that can be used to modify effect parameters. For those who prefer an API, Squeezer allows triggering and saving effect sequences generated by the sequence generator with the shorthand *Squeezer.Trigger(...)*. This shorthand generates an effect sequence based on a selected category and triggers it with the provided target, position, optionally a direction, and handing back a reference to the sequence for reuse later on. Programmers can also build and trigger effects or sequences manually; examples of this can be found in the sequence generator and several of the spawner effects (see Section 3.5).

3.3 Workflow

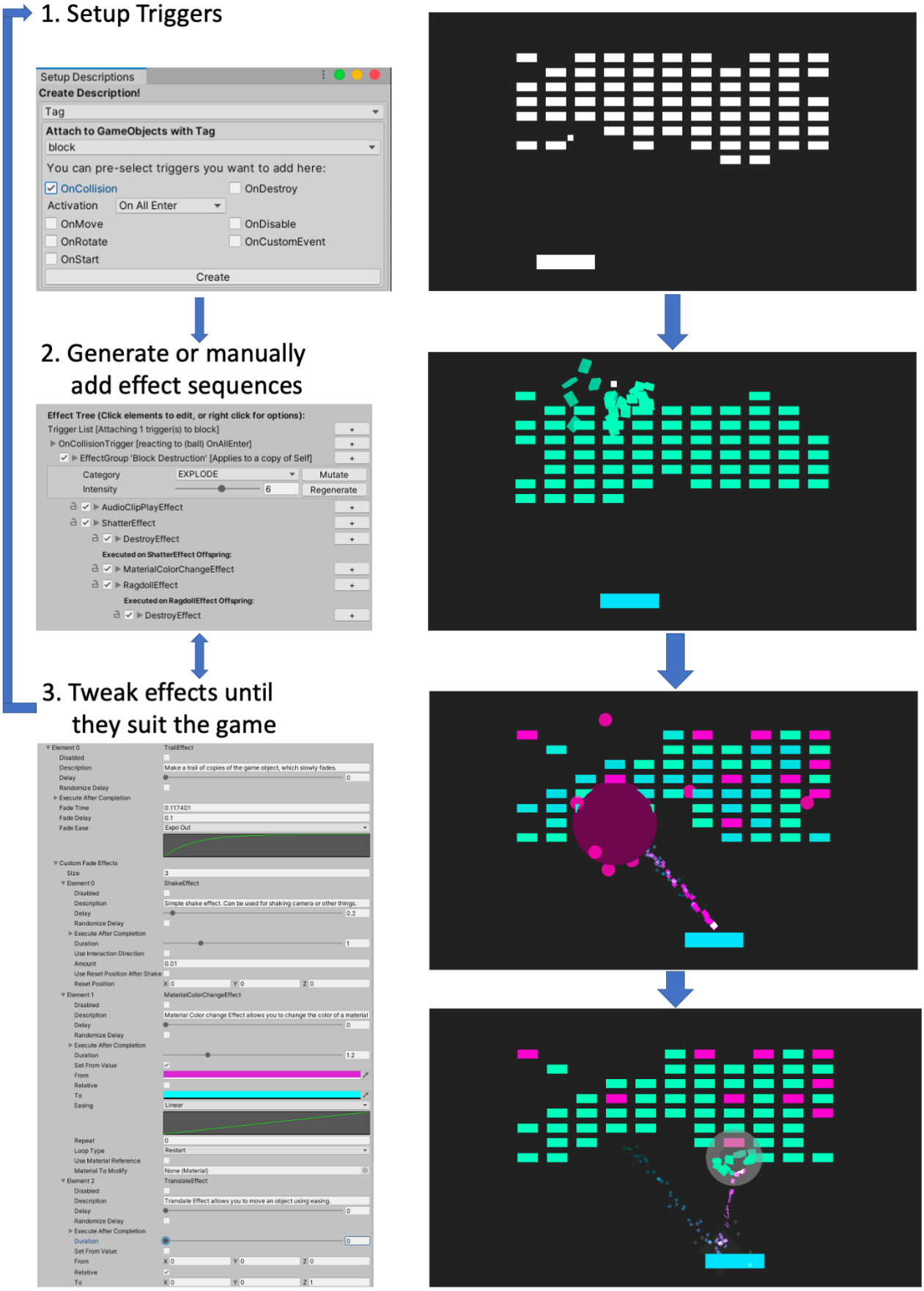
Fig. 2 shows a potential workflow for Squeezer, along with juice iterations on a breakout clone. Step one shows the setup window and basic game without effects. After setting up the triggers, step two shows the game with a bit of color and an initial generated effect sequence for block destruction. Step three shows the foldout menu for editing parameters of the trail effect. On the right side of step three, you can see a few iterations of the effects. Designers work iteratively, going back and forth between steps two and three, in addition to repeating steps one to three for every class of objects and event triggers they need.

3.4 Demo Scenes

We created four demo scenes in 2D and 3D. The demo scenes trigger effects on simple game objects when specific events occur. They are named Timer, Move, Jump and Shoot according to the event they demonstrate. All scenes either feature 2D objects and an orthographic camera or 3D objects and a rotating perspective camera. In Fig. 5) you can see one of the 3D demo scenes in the middle of executing an explosion effect.

[6]The Unity port of SFXR called usfxr: https://github.com/zeh/usfxr

-70pt-


1. Setup Triggers
Setup Descriptions
Create Description!
Tag
Attach to GameObjects with Tag
block
You can pre-select triggers you want to add here:
OnCollision
OnDestroy
Activation
On All Enter
OnMove
OnDisable
OnRotate
OnCustomEvent
OnStart
Create
2. Generate or manually add effect sequences
Effect Tree (Click elements to edit, or right click for options):
Trigger List [Attaching 1 trigger(s) to block]
OnCollisionTrigger [reacting to (ball) OnAllEnter]
EffectGroup 'Block Destruction' [Applies to a copy of Self]
Category
EXPLODE
Mutate
Intensity
6
Regenerate
AudioClipPlayEffect
ShatterEffect
DestroyEffect
Executed on ShatterEffect Offspring:
MaterialColorChangeEffect
RagdollEffect
Executed on RagdollEffect Offspring:
DestroyEffect
3. Tweak effects until they suit the game
Element 0
TrailEffect
Disabled
Description
Make a trail of copies of the game object, which slowly fades.
Delay
Randomize Delay
Execute After Completion
Fade Time
0.117401
Fade Delay
0.1
Fade Ease
Expo Out
Custom Fade Effects
Size
3
Element 0
ShakeEffect
Disabled
Description
Simple shake effect. Can be used for shaking camera or other things.
Delay
0.2
Randomize Delay
Execute After Completion
Duration
1
Use Interaction Direction
Amount
0.01
Use Reset Position After Shake
Reset Position
Element 1
MaterialColorChangeEffect
Disabled
Description
Material Color change Effect allows you to change the color of a material
Delay
Randomize Delay
Execute After Completion
Duration
1.2
Set From Value
From
Relative
To
Easing
Linear
Repeat
Loop Type
Restart
Use Material Reference
Material To Modify
None (Material)
Element 2
TranslateEffect
Disabled
Description
Translate Effect allows you to move an object using easing.
Delay
Randomize Delay
Execute After Completion
Duration
Set From Value
From
Relative
To

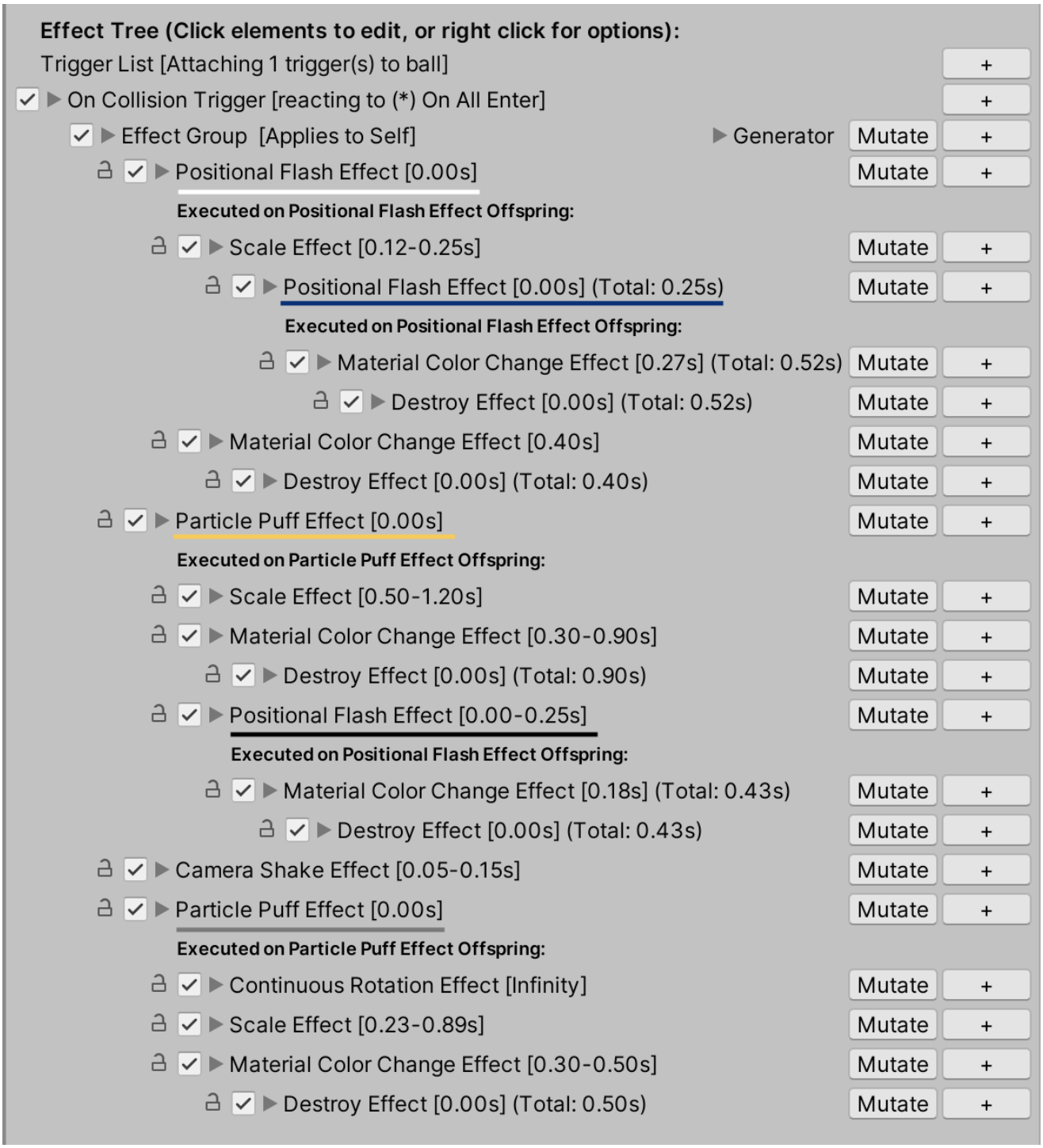


Figure 3: The description for an Explosion effect sequence. Underlines added externally to associate descriptions and visuals with the animation seen in Fig. 4.

Figure 4: The "synthesized" effect shown at 5fps by executing the description in Fig. 3. The big white square is the breakout ball (i.e. not part of the effect). There are three visible parts of the effect sequence in the description (the "Camera Shake" is not visible here). The first is the "Positional Flash" (white underline in Fig. 3) which shows up as a white circle in the first three frames, scaling up and fading out. After the scale up finishes a blue flash (blue underline in Fig. 3) appears and fades out. The second part of the sequence is the "Particle Puff" (yellow underline in Fig. 3) controlling the slowly expanding yellow circles, each will spawn a quick black flash (black underline in Fig. 3), giving the slight illusion of smoky dust as they expand and fade away. The third part of the sequence is the "Particle Puff" (gray underline in Fig. 3) simulating debris with the small fast moving gray boxes.

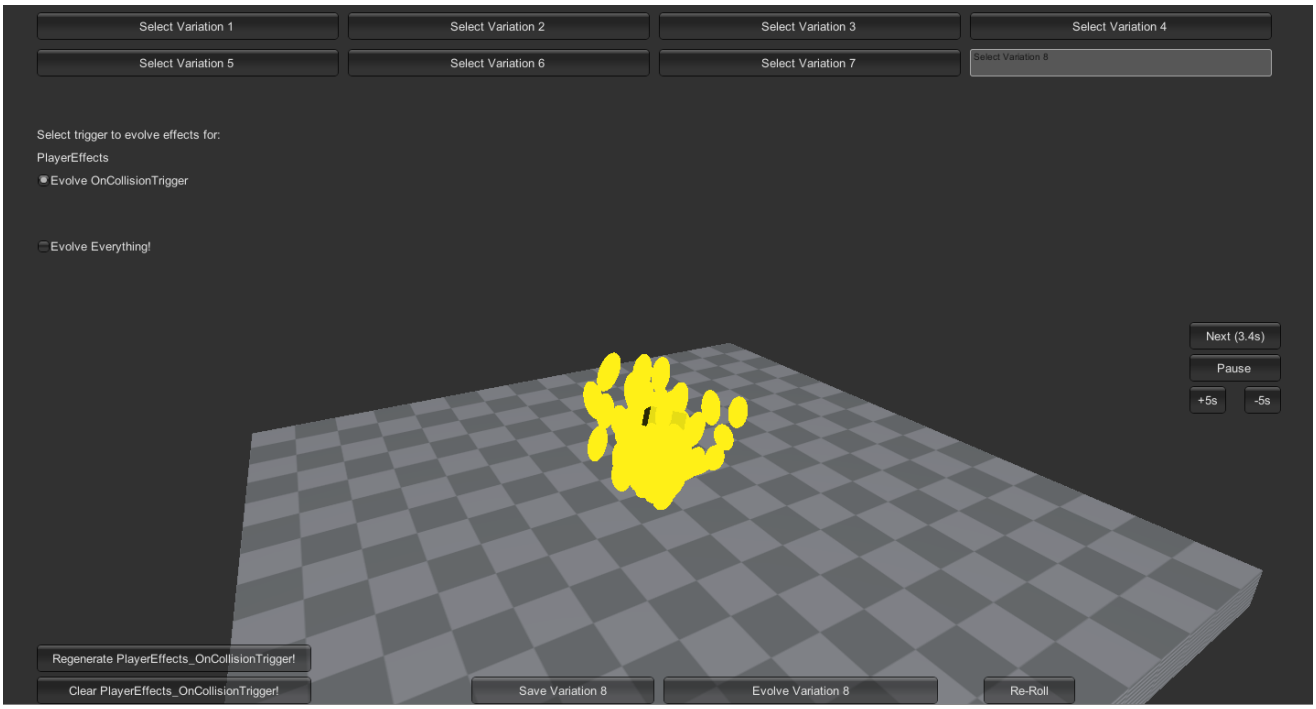


Figure 5: The run-time interactive evolution interface, seen here showcasing an explosion effect in a 3D scene.

Our original demo scene [18], a breakout clone, mainly consisted of events triggered by collisions. These new scenes revealed issues with trigger data as the initial version of Squeezer only passed a single 3D vector. The particle effect called "ParticlePuffEffect" provided the perfect test case for the amount of control and data needed when triggering events in the new demo scenes. The particle puff effect requires two parameters, a position, and a direction. Without changing the data provided by the triggers, the effect would only get a direction and would be forced to use the target object's position. However, both in the Jump and Shoot demo scenes, triggering particles at the center of a game object would be incorrect. For events such as collisions, the center of either colliding object would also not be an accurate position to spawn an effect. While technically feasible for a designer to add an invisible game object as the trigger target[7], allowing the system to pass a relative position offset removes the need to do that. Triggers can now provide effects with "TriggerData" specifying both direction and position, allowing the particle puff effect to accurately execute at the location of the impact between two objects. The particle puff effect itself provides a simple way to create a small cloud of particles, expanding in a selected pattern.

The demo scenes provide an easy way to design and test effects in isolation. However, a more realistic testing scenario presented itself as the user study was about to begin. The developers of DISC

[7]This is indeed a well known "hack" designers exploit to implement position offsets visually in Unity

ROOM [6] launched a game jam, where participants could create their different versions of the game. As part of this promotional event, a tutorial video for a simple disc room prototype with accompanying source files, sprites, and sound pack, was released[8]. Squeezer includes this simple DISC ROOM prototype, re-created in Unity with the sprite and sound assets.

3.5 Architecture

The initial problem of representing and triggering effects based on events or interactions has been solved many times. One solution is a hierarchical description approach, including an ontology. This approach is found in PyVGDL, JavaVGDL, and UnityVGDL [17, 22, 26]. VGDL frameworks execute entire games based on this structure and effectively hide complexity in the descriptions with the provided ontology. Both, Squeezer's data structure and its textual representation of effect sequences use this hierarchical description approach. Juice effect sequences require scheduling, both sequentially and simultaneously, which complicates the nesting logic.

The data structure in Squeezer is hierarchical and simplifies the complexity of triggers and effects by offering classes for specific purposes. Broken down from root to leaf, the data structure is as follows (the hierarchy can be seen in Fig. 3, in a partial view of the Description inspector):

- Description – in charge of attaching triggers to actual game objects in the game, contains a list of Triggers (The outer level containing the "Effect Tree" in Fig. 3).
- Triggers – define which events cause effects to occur. Selected from seven predefined trigger types. Contains a list of effect groups (In Fig. 3, only one trigger is attached to the description, namely "On Collision Trigger").
- Effect Groups – determine which game objects the effects will be executed on and contain lists of effects (In Fig. 3 only a default "Applies to Self" effect group has been added to the collision trigger).
- Effects – include functionality to execute the effects themselves and a list of effects to apply on completion. Effects that spawn objects, such as particle puff, additionally contain a list of effects to run on any generated objects. (In Fig. 3 we see four effects on the root level, two different particle puffs, a positional flash, and a camera shake. Each of those, in turn, contain effects to be executed on either their "spawned" offspring or on their target as they complete).

Effect trees represent sequences. Effects on the same level – with the same parent – will be queued simultaneously. Nested effects execute after the parent has completed. In addition to this, a delay can be added to effects, offsetting them from other effects on the same level. Delays are handled internally by the effects themselves and begin counting down as soon as the effects are triggered.

In [18] we selected effect triggering events by analysing industry talks [19, 21]. This allowed us to identify an initial set of event triggers:

- OnStart, triggered when creating an object/when the game starts.
- OnCollision, triggered when a collision occurs.
- OnMove, triggered while moving or changing movement state.
- OnRotate, triggered when the object rotates in some way.
- OnDestroy, triggered when destroying an object.
- OnDisable, triggered when an object becomes disabled (a common way of "destroying" objects, without invoking garbage collection in Unity)
- OnCustomEvent, triggered when the system receives a custom event (e.g., named custom events such as Shoot, Jump or FadeInComplete)

Since a lot of games rely on Finite State Machines (FSMs) to control game logic, we added a new trigger recently that is called **OnStateChanged**. It is capable of tracking and activating based on changes to specific script parameters (like FSM states), as well as of checking simple conditions. The trigger detects when the value of observed parameters changes (e.g., when the state of a character changes from OnGround to Jumping or when the player's speed exceeds a specified value). Reacting directly to changes in the game state non-invasively can be an excellent alternative to adding code for triggering custom events in multiple places.

Although triggers are part of the hierarchical data structure, each trigger has an accompanying detection script attached to game objects in the game. These detection scripts react to game events and activate their associated effect groups. The effect groups determine the targets of the effect sequences defined within them. Usually, the effect groups only target the triggering object. Yet, sometimes they can help make other game objects seem to respond to specific events[9].

The main groups of effects are; sound effects, color effects, particle and trail effects, transform effects (translate, rotate, scale), time dilation effects, flashing effects (full-screen or localized), shake (quick random translations), and wiggle (rotational shake) effects. However, common for all effects is that they can be delayed, applied to various targets, sequenced (scheduled in relation to other effects), and executed dependent or independent of in-game timescale. Apart from those common properties, feedback effects tend to be durational. The most common are tween effects. The word tween comes from "inbetweening" [25], which originated in cartoon animation, where a senior would draw keyframes of animation sequences, and juniors would then fill in the timelines between those keyframes. A tween interpolates between a beginning and end value over a duration. The interpolation can be linear or eased in and out using easing curves[10]. The easing functions simulate acceleration and deceleration when starting and stopping, based on various functions like a sine wave or an exponential function. Another class of effects is those that spawn additional objects in the game instead of manipulating objects that already exist. Trails and other particle effects that generate objects contain a separate list of effects executed on their offspring. The four types of effects are *One shot, Durational, Tween, and Spawner effects.*

3.5.1 Extending Functionality. Squeezer is open source and can easily be extended with custom effects. Those can be based on one

[8] DISC ROOM JAM Tutorial, available at https://youtu.be/Dtt5X7twhxA

[9] This can be seen in the Squeezer showcase [available at: https://github.com/pyjamads/Squeezer/tree/master/Showcase] when all blocks in breakout react to the destruction of a single block

[10] see examples on http://easings.net/

of four effect types – one shot, spawning, durational and tween – depending on what is appropriate. Adding new effects to the system is as easy as copying or inheriting other effects, changing the name and performed logic, and saving the file. Squeezer uses reflection[11] to show available effects. The inheritance structure and serialization allow the system to display and execute the effects correctly.

Similarly, new triggers can be added. That process is slightly more complicated, as the detection scripts need to be created and the attaching logic has to be implemented as well.

It should be noted that spawning effects – Trail, Shatter, Copy and Particle Puff – all instantiate regular Unity game objects, which makes them very versatile. The objects in question can be custom-made for this purpose by the designer. Creating a large amount of game objects in Unity is resource-intensive, so game performance has to be monitored when effects spawn objects.

3.6 Content Categories

In [18] we identified and proposed the idea of generating effect sequences based on different categories. We started out with SFXR's eight categories of effects (see above) but removed Blip/Select, Coin/Pickup and Power-up. We limited the Squeezer categories to a set of simple arcade game mechanics, that suited our test cases. We added two new options: Player Move and Projectile Move. Meant for continuous triggers and have no counterpart in SFXR. Finally we renamed some of them to make their use clearer. The set of categories is a starting point for effect sequence generation, and we expect this list to be expanded in the future. In the end we ended up with the following content categories (SFXR name on the left, Squeezer name on the right):

- Random ⟶ Random
- Explosion ⟶ Destroy/Explode
- Jump ⟶ Jump
- Laser/Shoot ⟶ Shoot
- Hit/Hurt ⟶ Impact
- N/A ⟶ Projectile Move
- N/A ⟶ Player Move

Fig. 1 shows the generator interface, where the category and intensity can be selected. The intensity value controls size, severity and sound volume, on a scale between one and ten (where one is the least intense). The intensity scale is split internally into four levels, [1-3] is low intensity variations, [4-6] is medium intensity, [7-9] is high intensity, and [10] is extreme intensity. The different intensity levels, have slightly altered effect sequences for some categories, but otherwise scale parameters linearly.

3.7 Generation, Mutation and Evolution

We handcrafted base sequences for each category. These sequences are the foundation of the generator. In the generator effects are initialized with parameters randomized within predefined ranges. This allows Squeezer to generate distinct effects with reasonable initial values.

Additionally, the sequences themselves are mutated at initialisation by randomly adding and removing effects in the sequence. The idea is that even significant mutations maintain some traits from the base sequence of the specified category. The logic for mutating effect trees combines genetic programming's [1] tree mutations with effect parameter mutations. This means that when mutating the effect sequence, a control parameter is used as a probability measure for adding or removing effects to the tree. While mutating each individual effect, the control parameter controls the probability that a value changes and how much it should be adjusted.

However, Squeezer is a design tool, and that means having a designer in the loop to determine fitness and guide the mutations along. To assist in this process, the user interface offers a locking mechanism[12]. Locked effects will not be mutated or removed while the rest of the tree gets mutated. This allows designers to 'freeze' parts of the effect sequence they like while evolving the remaining tree (one-armed bandit style). Designers can use this functionality and repeatedly mutate selected parts of the effect tree, gradually homing in on a final result.

3.8 User Interface

Squeezer offers interactive evolution through a run-time interface as well as through an inspector interface that is integrated into the engine's editor. In both instances, the genotype is a tree structure of effect class instances and the phenotype is the resulting sequence of "synthesized" effects that appear in the game.

The run-time interface consists of four control areas, presented as an overlay on top of the game view, as seen in Fig. 5. The top area lists each individual in the population (default is eight), and which one is currently selected. The left area lists the various triggers available for evolution. Each trigger is evolved separately, in order to avoid information overload. However, the last option on the list does allow bold designers to evolve everything at once. The bottom area contains buttons for starting over, saving the current individual, evolving the current individual or re-rolling the evolution from the previous stage. Both, saving and evolving, cause the current selected individual to be bookmarked, so that a designer can return to any step in the evolution they selected or saved at a later time[13]. The flow controls are found on the right side. These handle the timer that automatically switches between different individuals in the population. The interface shows only one individual at a time, and the game is reloaded when switching between them. Reloading the game scene, functions as a "palate cleanser", clearing out any lingering settings, colors or game objects the previous individual may have left behind. An individual can be skipped quickly, the display time can be changed and the timer can be paused entirely, allowing a designer to study the effects at the pace they see fit.

The inspector interface, that we integrated into the Unity editor, enables mutation of each effect group or individual effects. It can be used on its own or in combination with the run-time interface. In the latter case, locked and disabled parts of the effect tree will get copied to the next generation of individuals, guiding the evolution. The inspector interface allows the mutation of individual effects in the tree, and of entire effect groups. Effects can be reordered, deleted and copied between different parts of the effect tree. Effects

[11] https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection

[12] The padlock icons in Fig. 1 indicate which elements are locked

[13] These bookmarked descriptions have to be loaded back individually and manually through the inspector interface on the specific description game object. A workflow that should be improved in the future.

in an effect group can be mutated and regenerated from a selected category and with a specified intensity.

4 USER TEST

We invited five experienced game designers, two identifying as female and two identifying as male, to evaluate Squeezer. The participants had between two and seventeen years of experience as game designers.

The evaluation was carried out in a two phase process. We recorded each video session and collected usage data (telemetry). The two test phases were designed so the participants would first test the interactive evolution interface, without being able to modify effects directly and later get full access. This way, they could provide focused feedback of that part of the interface. In the second phase, we allowed the participants to explore the effect manipulation more thoroughly. Here, we were gathering their feedback on the interplay between the interactive evolution interface and the effect manipulation. Our test design was inspired by tutorial design, where games often slowly increase the number and complexity of the shown features to allow the user to learn. The two test phases allow us to ask questions about the interface preferences among the participants, and the pros and cons of each interface.

During the test, participants were asked to open the software for the first time. Then they were introduced to the concept and purpose of Squeezer. They were guided to open the DISC ROOM demo scene, and add the interactive evolution scene, to begin the test. We asked them to evolve effects for the player character colliding with discs, death animations, as well as effects for disc bouncing, using the run-time interactive evolution interface. During the test they were encouraged to provide feedback on the process, the interface, and the generated effects while they worked. Once they had sufficiently explored interactive evolution and either felt satisfied or mentioned they wanted to manipulate or tweak effects more directly, the test proceeded to the second phase. In the second phase, they were asked to pause the evolution and have a look at the Unity inspector interface (see Fig. 3 and 1). From here they were instructed on how the interface worked, and the interaction between the inspector and run-time interactive evolution. They were allowed to mutate, regenerate and change anything manually, through the inspector interface, and they also had the option of resuming the evolution using the run-time interface. After about an hour, they were asked to provide immediate thoughts and feedback, to sent us their usage data and fill out a short questionnaire.

The survey asked them about their game design experience, and experience with adding effects to games. Additionally they were asked to provide feedback on interactive evolution as an introduction to Squeezer and effect generation, their thoughts on using interactive evolution as part of the workflow, and if they found it satisfying and/or surprising to work with. Lastly, they were asked if they would be interested in using a tool such as Squeezer in the future, or what would need to be changed in order for them to use it.

5 RESULTS

Given that we conducted expert interviews, we focused on individual statements and general agreements between the participants. Additionally to the expert evaluation of the tool, the user test also helped us to find bugs and usability issues. Since the user tests were carried out over a few weeks, a few bug fixes and quality-of-life improvements were added between the tests.

An example of this is the following: The first participant noted that, while working with interactive evolution, they did not want to change details of an effect manually. Rather, they wanted to change the effect slightly in a random directly. They suggested the addition of a mutate button for all effects in the effect tree, to allow for the mutation of a single effect or sub-tree. Because the underlying functionality was already in place, the button was added to the inspector interface before the next test.

Several users expressed at times that they wanted to roll back one generation and re-roll the current population. Some wanted to completely start over because they ended up with effects they didn't like. To avoid the latter and support the former, buttons to re-roll, restart and clear parts of the effects were added to the run-time interface between the second and third participant.

5.1 Qualitative User survey

The qualitative survey included questions about the user's proficiency in the area of game feel design. Four out of five participants write custom code or use libraries such as DOTween[14] for animations, and trigger the corresponding events themselves from code. But only two of the five were using any tools currently. One uses their own custom tools, the other one libraries like DOTween or Unity's Coroutines[15].

Three of the five participants found interactive evolution was beneficial to getting started with Squeezer and helped them get a perspective on the possibilities. One participant found it mostly useful as a way to discover new ideas, while the last participant found it "*cool*" but was missing some transparency in how the evolution actually would change the effects.

When asked about their thoughts on using interactive evolution in a tool like Squeezer, three of the participants said that they imagined it would be good for new designers and developers without the knowledge to "*easily make cool looking stuff*" or just as a tutorial for Squeezer. One participant found it "*chaotic, but also playful*" pushing them out of their comfort zone, while helping them to "*explore a possibility space too large to contain*" in their head.

Four of the participants found that the generated and evolved effects surprised them, and led to unexpected but interesting results. A few of the participants ran into bugs, that caused the exploration to mostly generate certain kinds of effects, and noted that it felt like the system mostly liked the color white and transparency effects.

Three of the five participants said they might use this in the future for game jams and personal projects, and a fourth said it might be useful for coming up with new ideas for effects. The last participant relies more on triggering effects manually from code, and suggested a more "lightweight" version of the API, would allow them to bypass Squeezers trigger system, but agreed that Squeezer should still have that capability as a visual design interface. One of the participants did raise the concern that Squeezer would probably not transfer well from the prototyping phase into production.

[14] http://dotween.demigiant.com/
[15] https://docs.unity3d.com/Manual/Coroutines.html

5.2 Participant Usage

The authors noted that the depth of the evolution was at most around ten steps. It usually took only two to four steps before users felt either content or wanted to start over. This could either be caused by the loose definition of the task they were asked to perform or by the fact that users felt they had run into "dead ends".

The sound effect synthesizer caused several participants to mute the audio, because it kept generating oddly loud sounds. Squeezer allows full control and visibility of all the mutated values, providing users with insight into the changes the system is making [37]. One participant noted that the inspector interface has quite a lot of information. They proposed that adding icons and changing the names of effects could improve the user interface.

Several participants were surprised by the effects generated during evolution, and mentioned how those effects changed the game mechanics in surprising ways. Mixed-initiative systems can provide surprisingly creative insights when added to game artifacts [35]. To one participant Squeezer felt like "*an animation showreel for effects.*" and we took that as a compliment.

6 REFLECTION

In this paper we presented how adding interactive evolution to Squeezer can assist game designers in exploring game feel design [24] in game prototypes. The two interfaces for interactive evolution in Squeezer each have merits of their own. They can be functionally combined. However, the user experience of working with it, is still far from optimal. Future research is needed to explore ways of presenting a range of different effect variations for interactive systems. Previously, we created a Breakout clone with an AI playing the game [18], to allow for designing and testing at the same time. However, adding artificial agents to any game requires a lot of effort, and is thus a bad fit for the prototyping phase. While it is currently possible to edit the effect sequences while the game is running, this feature is not fully useful without a preview window.

While the effect tree generator supports generating effect sequences for a lot of opportunities in a game, several additional cases were discovered while preparing for the user testing. Categories such as 'Starting/Spawning', with objects fading, sliding or scaling into view, as well as options to distinguish between 2D and 3D effects would be great additions. Adding these additional categories and options would enhance the potential of applying Squeezer in game development.

Similarly, creating variations of the underlying handcrafted trees would allow an even more diverse set of generated trees to be created. This would present more variation to designers when generating effects for a certain event. Additionally it might be interesting to align variations across different event types, by creating for instance "Explosion type A", which worked well with "Movement trail A" or "Muzzle flash A".

Like with every prototyping tool, a potential pitfall of Squeezer is that the designer can become too attached to the placeholder effects. That they are locked into a particular path of exploration and limited in their creativity. The value lies in quickly exploring feasibility and examples of juice, and then being able to "start over" using the lessons learned from the prototype. Squeezer does not contain every possible effect, and it might even be missing some critical effects for game types we overlooked. However, it provides a good starting point for exploring and implementing effects early in a game design process, when the final look and feel has yet to be determined.

As one participant pointed out in their evaluation, Squeezer could benefit from streamlining the API for triggering effects directly from code. While this is already possible, with the current *Squeezer.Trigger(...)* API, there are several ways to tie code and effect descriptions together. An API that easily allows programmers to trigger, cache, mutate and save effects based on categories and intensity from code would open the evolutionary aspects of Squeezer to programmers.

7 FUTURE WORK

Comparing this version of Squeezer to the initial version presented and tested in [18], a lot has changed. The interface has been overhauled completely, many of the initial interaction design-concerns have been addressed. The generator, interactive evolution, and the mutation and locking system have been added. It is hard to compare the two interfaces, because the user interface has been upgraded with many small quality of life features. Yet, while we believe the new version is a lot better than the initial version, further studies of designers using Squeezer for their own projects, would highlight more thoroughly the issues and qualities of Squeezer.

As we stated in [24], Automated Game Generation [8] could benefit from the addition of effects. However, it requires implementing measure that make sure that effects are not conveying misleading feedback. A way of analysing what message a player gleams from a given effect sequence, and which effects fit generated game mechanics must be provided to add fitting feedback to generated games. Similar to having categories align across different events in a game, verifying that all effects match the game-play of a generated game consistently, requires further research.

8 CONCLUSION

This paper shows how generative and evolutionary strategies can be applied in game design, specifically in the realm of juice and feedback. It demonstrates how Squeezer, a tool for rapidly designing juice effects, became a "synthesizer", combining modular sequencing and presets to generate complex chains of effects. The tool can help exploring the effect design space and it can also support determining which kinds of juice effects enhance feedback of a game prototype [18]. In order to further support this kind of exploration and discovery of the design space of game feedback, we expanded Squeezer with an interface for generation, mutation and interactive evolution of effects.

To verify the quality of our implementation, we conducted a user study with game designers who had no prior experience with the software. The study indicated that Squeezer could be a great tool to help designers discover and explore the possibilities of juice effects and interesting game mechanics. Additionally our test indicates that interactive evolution could ease the process of learning, creating and exploring juice effects with Squeezer.

ACKNOWLEDGMENTS

We would like to thank the test participants for participating and providing their feedback about working with Squeezer.

REFERENCES

[1] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. 1998. *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[2] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep Interactive Evolution. In *Computational Intelligence in Music, Sound, Art and Design*, Antonios Liapis, Juan Jesús Romero Cardalda, and Anikó Ekárt (Eds.). Vol. 10783. Springer International Publishing, Cham, 267–282. https://doi.org/10.1007/978-3-319-77583-8_18

[3] Mark Brown. 2015. Secrets of Game Feel and Juice. https://www.youtube.com/watch?v=216_5nu4aVQ. https://www.youtube.com/watch?v=216_5nu4aVQ Accessed: 2020-04-17T11:52:14Z.

[4] Mark Brown. 2019. Why Does Celeste Feel So Good to Play? | Game Maker's Toolkit. https://www.youtube.com/watch?v=yorTG9at90g. https://www.youtube.com/watch?v=yorTG9at90g Accessed: 2020-04-17T13:21:54Z.

[5] N. Burtnyk and M. Wein. 1971. Computer-Generated Key-Frame Animation. *Journal of the SMPTE* 80, 3 (March 1971), 149–153. https://doi.org/10.5594/J07698

[6] Kitty Calis, Jan Willem Nijman, Terri Vellmann, and Adam Drucker. 2020. Disc Room.

[7] Jeff Clune, Jason Yosinski, Eugene Doan, Nabeel Samad, Sijie Liu, and Hod Lipson. 2012. EndlessForms.Com - Design Objects with Evolution and 3D Print Them! http://endlessforms.com/. http://endlessforms.com/ Accessed: 2020-11-17T17:22:46Z.

[8] Michael Cook, Simon Colton, and Jeremy Gow. 2014. Automating Game Design In Three Dimensions. In *AISB Symposium on AI and Games*. AISB, 1–4. http://research.gold.ac.uk/id/eprint/17354/

[9] Frank Force. 2020. ZzSprite - Tiny Sprite Generator. https://killedbyapixel.github.io/ZzSprite/. https://killedbyapixel.github.io/ZzSprite/ Accessed: 2020-11-17T17:25:13Z.

[10] Renaud Forestié. 2019. How to Design with Feedback and Game Feel in Mind - Shake It 'til You Make It. https://www.youtube.com/watch?v=yCKI9T3sSv0. https://www.youtube.com/watch?v=yCKI9T3sSv0 Accessed: 2020-04-17.

[11] Tracy Fullerton. 2014. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games* (3rd ed.). A K Peters/CRC Press.

[12] Kyle Gray, Kyle Gabler, Shalin Shodhan, and Matt Kunic. 2005. How to Prototype a Game in Under 7 Days. https://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php. https://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php Accessed: 2019-10-27.

[13] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing Communications and Applications* 9, 1, Article 1 (Feb. 2013), 22 pages. https://doi.org/10.1145/2422956.2422957

[14] Kieran Hicks, Patrick Dickinson, Juicy Holopainen, and Kathrin Gerling. 2018. Good Game Feel: An Empirically Grounded Framework for Juicy Design. In *Proceedings of the 2018 DiGRA International Conference: The Game Is the Message*. DiGRA, 17. http://www.digra.org/wp-content/uploads/digital-library/DIGRA_2018_Paper_35.pdf

[15] Amy K. Hoover, Paul A. Szerlip, Marie E. Norton, Trevor A. Brindle, Zachary Merritt, and K. Stanley. 2012. Generating a Complete Multipart Musical Composition from a Single Monophonic Melody with Functional Scaffolding. In *ICCC*, Vol. PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON COMPUTATIONAL CREATIVITY. ICCC, Proceedings of the Third International Conference on Computational Creativity, 111–119.

[16] Robin Hunicke. 2009. Loving Your Player With Juicy Feedback. http://2009.dconstruct.org/podcast/juicyfeedback

[17] Mads Johansen, Martin Pichlmair, and Sebastian Risi. 2019. Video Game Description Language Environment for Unity Machine Learning Agents. In *2019 IEEE Conference on Games (CoG)*, Vol. 2019 IEEE Conference on Games (CoG). IEEE, 1–8. https://doi.org/10.1109/CIG.2019.8848072

[18] Mads Johansen, Martin Pichlmair, and Sebastian Risi. 2020. Squeezer - A Tool for Designing Juicy Effects. In *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '20)*. Association for Computing Machinery, New York, NY, USA, 282–286. https://doi.org/10.1145/3383668.3419862

[19] Martin Jonasson and Petri Purho. 2012. Juice It or Lose It. https://www.youtube.com/watch?v=Fy0aCDmgnxg. https://www.youtube.com/watch?v=Fy0aCDmgnxg Accessed: 2019-02-27.

[20] Jesper Juul and Jason Scott Begy. 2016. Good Feedback for Bad Players? A Preliminary Study of 'Juicy' Interface Feedback. In *Proceedings of First Joint FDG/DiGRA Conference*, Vol. Proceedings of first joint FDG/DiGRA Conference. DiGRA, Dundee, 2. https://www.jesperjuul.net/text/juiciness.pdf

[21] Jan Willem Nijman. 2013. The Art of Screenshake. https://www.youtube.com/watch?v=AJdEqssNZ-U. https://www.youtube.com/watch?v=AJdEqssNZ-U Accessed: 2021-04-22.

[22] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couetoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. 2016. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8, 3 (Sept. 2016), 229–243. https://doi.org/10.1109/TCIAIG.2015.2402393

[23] Tomas 'DrPetter' Pettersson. 2007. SFXR. http://www.drpetter.se/project_sfxr.html. http://www.drpetter.se/project_sfxr.html Accessed: 2020-07-11.

[24] Martin Pichlmair and Mads Johansen. 2021. Designing Game Feel. A Survey. *IEEE Transactions on Games* IEEE Transactions on Games (Early Access), IEEE Transactions on Games (Early Access) (2021), 1–20. https://doi.org/10.1109/TG.2021.3072241

[25] William T. Reeves. 1981. Inbetweening for Computer Animation Utilizing Moving Point Constraints. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '81*. ACM Press, Dallas, Texas, United States, 263–269. https://doi.org/10.1145/800224.806814

[26] Tom Schaul. 2013. A Video Game Description Language for Model-Based or Interactive Learning. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, Niagara Falls, ON, Canada, 1–8. https://doi.org/10.1109/CIG.2013.6633610

[27] Tom Schaul. 2014. An Extensible Description Language for Video Games. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (Dec. 2014), 325–331. https://doi.org/10.1109/TCIAIG.2014.2352795

[28] Jimmy Secretan, Nicholas Beato, David B. D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. 2008. Picbreeder: Evolving Pictures Collaboratively Online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. Association for Computing Machinery, Florence, Italy, 1759–1768. https://doi.org/10.1145/1357054.1357328

[29] Noor Shaker, Julian Togelius, and Mark Nelson. 2016. *Procedural Content Generation in Games*. Springer, USA. http://www.springer.com/gp/book/9783319427140

[30] Jiesang Song. 2005. Improving the Combat 'Impact' Of Action Games.

[31] Steve Swink. 2007. Game Feel: The Secret Ingredient. https://www.gamasutra.com/view/feature/130734/game_feel_the_secret_ingredient.php?print=1. https://www.gamasutra.com/view/feature/130734/game_feel_the_secret_ingredient.php?print=1 Accessed: 2020-04-17T11:52:32Z.

[32] Steve Swink. 2009. *Game Feel.* Morgan Kaufmann.

[33] H. Takagi. Sept./2001. Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation. *Proc. IEEE* 89, 9 (Sept./2001), 1275–1296. https://doi.org/10.1109/5.949485

[34] Frank Thomas and Ollie Johnston. 1981. *The Illusion of Life: Disney Animation.* Abbeville Press, New York.

[35] Georgios N Yannakakis, Antonios Liapis, and Constantine Alexopoulos. 2014. Mixed-Initiative Co-Creativity. In *Foundations of Digital Games 2014*. Society for the Advancement of the Science of Digital Games, Proceedings of the 9th International Conference on the Foundations of Digital Games, 8.

[36] Jinhong Zhang, Rasmus Taarnby, Antonios Liapis, and Sebastian Risi. 2015. DrawCompileEvolve: Sparking Interactive Evolutionary Art with Human Creations. In *Evolutionary and Biologically Inspired Music, Sound, Art and Design*. Springer, Cham, 261–273. https://doi.org/10.1007/978-3-319-16498-4_23

[37] Jichen Zhu, Antonios Liapis, Sebastian Risi, Rafael Bidarra, and G. Michael Youngblood. 2018. Explainable AI for Designers: A Human-Centered Perspective on Mixed-Initiative Co-Creation. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, Maastricht, 1–8. https://doi.org/10.1109/CIG.2018.8490433

9.4 Designing game feel. A survey.

Designing Game Feel. A Survey

Martin Pichlmair and Mads Johansen*

***Abstract*—Game feel design is the intentional design of the affective impact of moment-to-moment interaction with games. In this paper we survey academic research and publications by practitioners to give a complete overview of the state of research concerning this aspect of game design. We analysed over 200 sources and categorised their content according to design purposes. This resulted in three different domains of intended player experiences: physicality, amplification, and support. In these domains, the act of polishing, which determines game feel, takes the shape of tuning, juicing, and streamlining respectively. Tuning the physicality of game objects creates cohesion, predictability, and the resulting movement informs level design. Juicing is the act of polishing amplification and it results in empowerment and provides clarity of feedback. Streamlining allows a game to act on the intention of the player, supporting the execution of actions in the game. These three design intents are the main means through which designers control minute details of interactivity and inform the player's reaction. The presented framework and its nuanced vocabulary can lead to an understanding of game feel that is shared between practitioners and researchers as highlighted in the concluding future research section.**

***Index Terms*—Game feel, game design, affect, juice, feedback.**

I. Defining our Scope

THIS paper is a review of existing research about game feel. It uses reflections of practitioners and research publications for defining a more precise vocabulary to talk about the design of how interacting with a game feels[1]. The logical starting point for defining what "game feel" means is the book of the same name by Steve Swink, who defines it as "real-time control of virtual objects in a simulated space, with interactions emphasised by polish" [1]. He further expands on that definition by stating that great-feeling games convey five kinds of experiences, namely:

- The aesthetic sensation of control
- The pleasure of learning, practising and mastering a skill
- Extension of the senses
- Extension of identity
- Interaction with a unique physical reality within the game

While Swink's definition of game feel covers a wide range of video games, it is too limited to encompass all kinds of them. With the above list he excludes games without real-time controls from possessing the quality of game feel. Wilson [2] challenges this aspect of Swink's book and extends the notion of 'aesthetic sensation of control' by connecting game feel with the cultural history of gestures. Wilson distinguishes between 'Game Feel' and 'game feel', the first being the positive feeling of control that Swink describes, the second being any feeling a game communicates. Jesse Schell does not mention the term 'game feel' in his book [3] at all. Yet he writes that designers should consider how their game feels in the context of required skills, learnability, and balance. The journalist and game maker Rogers wrote an exhaustive article [4] about what he calls 'friction', an alternative term for how a game feels. Friction is the experience of the player pushing against the boundaries of the system. It is the feeling of the inertia of the design working against the user's force. Friction is often experienced by the player over a longer duration than moment-to-moment interaction, as it extends over several game elements. Friction can be the defining element of a game, or just a part of the experience. Rogers' monolithic article is difficult to parse and, while detailed, does not go deep into specific aspects of game feel. It mostly recounts different feelings the author had during particular situations in games, establishing a wide vocabulary for talking about the aesthetic experience of playing. Similarly, Anthropy & Clark [5] establish a 'game design vocabulary' in their book of the same name. In it, they approach friction from the designer's perspective, and call it 'resistance'. The resistance of the game determines the experienced friction. It decides how the game feels to the player.

Ehrndal [6] approaches the topic of game feel by linking reflections of practitioners with aesthetic theories of games. Larsen [7] starts from a similar point and attempts to define an 'aesthetics of action'. He builds on Swink [1] and Nijman [8], both game developers more than researchers, in order to analyse the components of a game that contribute to what he calls a 'thrilling experience'. Yang [9] on the other hand expands the theory of game feel to include the metaphorical aspects of game objects and their relations to players. Building on queer theory, he includes political aspects of games in their 'feel' in order to communicate the diversity of the gameplay experience over diverse players, and also in order to provide game makers with a richer set of design tools.

While games are multi-sensory experiences, we are focusing on the haptic and visual aspects of game feel in this article, aware that narrative content, music, art, and many other aspects of a game influence how it feels. Very similar techniques to the ones described in this paper exist for example for designing the feel of the story of a game [10], writing the voices of in-game characters, balancing its rules, and tuning its atmosphere. We will leave those aspects and the social interaction between players to future research, while acknowledging that they are at the core of a lot of games. Instead, this paper is concerned with

Manuscript received November 12, 2020; revised March 10, 2021.

*The authors are with the Center for Computer Games Research, Digital Design Department, IT University of Copenhagen, Copenhagen, Denmark (e-mail: mpic@itu.dk and madj@itu.dk)

[1]It is important to stress that we only refer to game examples in order to clarify design details but did not survey games.

moment-to-moment interactivity [11]–[13], microinteractions [14] and interactions with core loops [15] and their design. Unlike Swink's precise but narrow definition of 'Game Feel', we will look at game feel more broadly as the affective aspect of interactivity. Our focus, thereby, is on the player and the design of their interaction with the game.

II. The Physicality of Interactivity

This survey paper gives an overview of the history, context, and state of the art understanding of game feel and how to design it. It is based on research in the field and publications by practitioners in order to capture both conceptual and practical knowledge. This chapter gives an overview of academic lines of thinking that lead to an understanding of game feel.

A. From Flow to Feel

In the years before Swink wrote the book 'Game Feel' in 2009 [1], research about the link between emotions and gameplay was most often connected to Csikszentmihalyi's famous 'Flow theory', one of the results emerging from a global research project about experiences that are "so gratifying that people are willing to do it for its own sake, with little concern for what they will get out of it, even when it is difficult or dangerous" [16]. Tasks that allow for this quality of experience feature the following eight elements:

1) a task that can be completed;
2) the ability to concentrate on the task;
3) that concentration is possible because the task has clear goals;
4) that concentration is possible because the task provides immediate feedback;
5) the ability to exercise a sense of control over actions;
6) a deep but effortless involvement that removes awareness of the frustrations of everyday life;
7) concern for self disappears, but sense of self emerges stronger afterwards; and
8) the sense of the duration of time is altered.

The concept of Flow has had an enormous influence on the understanding of experiential qualities of games. Sweetser & Wyeth [17] adapted Flow theory to games and Juul [18] discusses and criticises the theory's relevance for describing enjoyable challenges in games. Both texts contain multiple references to how games make a player feel, so it is natural to think of them as stepping stones towards a closer examination of game feel. Ciccoricco [19] links Flow to the gameplay experience of Mirror's Edge [20], a game that was sold on its merits in fluidity of movement, and contrasts it with the feminist concept of fluidity. Chen [21] famously not only based his graduation thesis on Flow but also released three successful commercial games based on his understanding of this concept with That Game Company, a studio he co-founded; Flow [22], Flower [23], and Journey [24].

Game feel is most strongly reliant on points three, four, and five in the above list of criteria. Clarity of goals will be discussed in the context of streamlining of the player experience in Section IV-C. Immediate feedback is at the heart of this paper and of the link between a game and how it feels. The sense of control will be mentioned at various points, for example in relation to the illusion of control and immersion. While some of the other items in Csikszentmihalyi's list apply to games too, they do so in a more indirect way.

B. The Purpose of Juice

While Flow is very well suited for understanding the dynamics of immersion, game feel is more focused on the role that interactivity plays in this process. A design concept often mentioned when talking about how interactivity can be intensified is 'juice'. Juice amplifies interactivity by providing excessive amounts of feedback in relation to user input [25] (see also [26]). The goal of juice is to make actions feel significant and the results can be measured [27]. It is superfluous from a strictly mechanical perspective, but turns interacting with the system into a more pleasurable experience. Too much juice, though, makes it hard to learn what aspects of interactivity have mechanical importance [28]. Decoding the actual system behind the game becomes cumbersome when the whole screen is filled with wobbling particle effects — unless this is a conscious aesthetic choice and itself part of the game's mechanics.

At the same time, the diversity of the medium allows that some games — we can call them 'toys' or 'autotelic experiences' [29] — are almost purely made of juice. Interacting with those toys is still playful and based on feedback amplified by juice. Only through feedback can we learn (to play), and all play is learning ([30]–[32], see also [33] and [34]). One could even go so far as to argue that all cognition is rooted in feedback from the real world that we actively engage with in a process of interactive cognition (see [35], based on [36]).

Overall, the goal of the application of juiciness is to enhance the feedback when interacting with game objects. Kao [37] conducted a large-scale study on the amount of juice appropriate for a specific gaming situation, concluding that juice has to be applied adequately to the situation. In their study, medium and high levels of juice outperformed extreme levels and the absence of juice across the measures of player experience, intrinsic motivation, play time, and in-game performance. Hicks et al. [26] bridge industry knowledge and academic analysis, building on Juul's [25], [38], Schell's [3] and Deterding's [39] work on juice in video games (see also [40]). They present a framework for the analysis of "good game feel" in games.

Sometimes juice exists not for the player but for the audience watching the game. Rogers hints at that when he says "The player knows where the hit range of the weapon is. He doesn't see the little juice-dance of the chain-daggers." [4]. Swink attributes a similar effect to ragdoll physics: "The ragdoll raptors have 'over the shoulder' appeal. People walking by someone playing the game often stop and want to know more..." [1]. Gage [41] describes the upsides of having a game that is readable from distance in their talk on 'subway legibility'. More specifically, there are elements of juiciness that designers implement for the audience, especially for streaming and e-Sports [42]. Some elements might also draw in the player but become invisible to them over time.

Hunicke [43] remarks that "juiciness can be applied to abstract forms and elements and it is a way of embodying arbitrarily defined objects and giving them some aliveness, some qua, some thing, some tenderness." Interestingly, Swink [1], Larsen [7] as well as Fullerton [44] use the term 'polishing' to describe something very similar to this. Fullerton describes the act of polishing as "the impression of physicality created by layering of reactive motion, proactive motion, sounds, and effects, and the synergy between those layers" [44]. In other words she sees polish as a means of giving physicality to inanimate objects in order to render them more tangible, which is remarkably similar to Hunicke's reasoning for juiciness. Practitioners call many things 'polish', e.g. fixing the timing of voice cues, or fixing bugs in the code (see e.g. [45]). Polish is linked to juiciness in that all juicy elements are polished at some point, but it is seen as a mostly aesthetic endeavour that stops short of changing the basic rules of a game, its core narrative, or its principal game mechanics. In practice, this separation is not always maintained and the connection between juicing, polishing, designing, and the feelings elicited by the feedback loop of interacting with a game is complex.

C. Juicing that Feeling

The intentionality of polishing and juicing apparent in Hunicke's and Fullerton's comments is at the centre of Brown's assertion that "you're not juicing your game — you're actually picking a feeling that your game should communicate and juicing that feeling" [46]. Feelings are emotions, but it would go too far to cover complex emotions like love, hate, and guilt in this paper. The moment-to-moment interaction we are concerned with is instrumental to the feelings that Baumeister et al. [47] call 'automatic affect'. Automatic affect is closely linked to experience via feedback loops in that the affective reaction to a stimulus has an effect on future experiences of stimuli and those have an effect on the person experiencing them and so on. Emotions are generally characterised by physiological arousal, the quality of the experience, and valence, which can be either positive or negative [47]. Game designers are of course concerned with positive as well as negative emotions, because stretches of sadness and near-frustrating challenges provide the perfect breeding ground for happiness and relief. It is important to note that humans are capable of experiencing multiple and even conflicting emotions simultaneously [48]. Further, experiments in mood regulation have shown that humans exhibit a 'homeostatic mood management mechanism' [49]. After initial mood-congruent responses, we spontaneously reverse and replace those by mood-incongruent reactions. So, in addition to the feedback between the outside world — including mediated experiences like video games — and our emotional state, there is a feedback loop built into our mood.

These affective feedback loops are central to how we interact with the world, including the virtual world of video games. The connection between emotion and cognition is a vast research field and proponents of that field like Okon-Singer et al. often speak of how central emotions are to cognition [50]. The emotional aspect of design has been reflected by design thinkers like Löwgren, Kirkpatrick, Hodent, and Karhulati. Löwgren provides a vocabulary for linking aesthetics, design, and emotional responses [51], [52]. The sensibility and precision he employs to talk about design elements and design choices is valuable for better discussions about game design. Kirkpatrick [53]–[55] and Karhulahti [56] build an aesthetic theory of video games that encompasses kinaesthetics as a foundational building block. Hodent [57] successfully bridges the chasm between interaction design and user experience design for games and accurately summarises the links between Norman's work [58] and video game design. In general, User Experience (UX) Design is an area that is concerned with the experiential aspects of interactivity within the vast field of Human-Computer Interaction. Hassenzahl [59] presents an in-depth study of the complex links between needs, affect, and interactivity. Methods originating in UX design have found their way into games [60]. Their main influence is indirect. They inform the design and iteration process by offering a portfolio of tools and techniques. For example, Dan Saffer's proposal of microinteractions [14] links to game design in that the basic components of microinteractions are triggers, rules, feedback, and loops (or modes) — all basic building blocks of game design.

Playing is an embodied experience [61] that can be viewed through the lens of phenomenology. Keogh [62] argues that the phenomenology of play, rooted in the understanding of embodiment by Merleau-Ponty [63], Bateson [64], and Weiss [65], "must not start with the experience of the player's body, but with the experience through which the player's amalgam embodiment in and as part of the videogame performance emerges." [62]. Surman [66], Davnall [67], Putney [68] offer three personal takes on three different games, echoing similar struggles of coming to terms with the bodily experience of playing games and the implications of the act of doing so. Sudnow [69] can be regarded as the trailblazer of writing about video games from a phenomenological standpoint.

In summary, game feel research is concerned with how our minds and our bodies experience the emotions of playing games. The question of how to design the emotional aspect of the play experience has been at the centre of a lot of research that connects design theory, psychology, phenomenology, philosophy, and many more areas. The following section will go into the building blocks designers use to intentionally elicit emotions, when "juicing that feeling" [46].

III. Game Feel Design Elements in Practice

Gameplay designers, of which some have a programming background and some have a design background, have analysed their own practice in countless blog posts, podcast episodes, conference presentations, and, sometimes, scholarly publications. The majority of works concerned with topics of game feel are descriptive in nature. They usually focus on either a single game or a specific feature or set of features that the designers have worked on. What can be learned from these texts, more than anything, is that experienced gameplay designers are very conscious about which aspects of their game are relevant for shaping its feel.

Some practitioners talk about how to structure game development processes around the design of game feel [70]–

[72]. Others focus on giving broad overviews of techniques [8], [13], [73]–[75]. Podcasts and video series by experienced practitioners such as Eggplant: The Secret Lives of Games [76]–[79], The Clark Tank [80] and Game Maker's Toolkit [72], [81], frequently discuss game feel design as a part of their coverage of game design topics.

Most practitioners focus on details and material aspects of design instead of connections between their work and the context they are working in. Noticeable exceptions are Hodent [57], who links game feel to classical concepts of game development like the '3Cs' [82], User Experience Design, as well as to Norman's theories on emotional design [58] and Song [83], who provides an excellent overview of how to model the feeling of impact in action games. Turner [84] wrote one of the few articles on how to influence game feel via sound design, based on his own work in game audio. Ismail [85] speaks about community development, explaining how communities of makers establish more and more sophisticated discourse about their practice over time. Yang [9], who has been mentioned before, always connects his personal practice to his academic research.

In general, the topics that these practical articles cover cannot easily be isolated from each other. They all concern feedback and how it relates to controls of a game. If the game is regarded as a feedback system (following [86], [87] and [88]), then game feel can be seen as a modulation of said feedback system. Designing game feel is designing the adequate feedback for eliciting a specific feeling or affective reaction. The following chapters list different design elements that determine the game feel, the feel of moment-to-moment interaction. We cluster design elements into classes according to the game's discussed subsystem. Table I presents an overview of the areas we're looking at and lists the most relevant examples mentioned. The table is not an exhaustive overview of all aspects of game feel from a practitioner's perspective. It is a starting point for understanding the practices that are most relevant, or most discussed, in regard to designing game feel.

TABLE I
GAME FEEL DESIGN ELEMENTS OVERVIEW.

Design Element	Physicality	Amplification	Support	Key References
Movement and Actions				
Basic Movement	•			[89]–[95]
Gravity	•			[92], [96], [97]
Terminal Velocity			•	[92]
Coyote Time			•	[98], [99]
Invincibility Frames			•	[100]–[102]
Corner Correction			•	[28], [103]
Collision Shapes	•		•	[104]
Button Caching			•	[92]
Spring-locked Modes	•			[5], [105], [106]
Assisted Aiming			•	[28], [107], [108]
Event Signification				
Screen Shake	•	•	•	[8], [73], [109], [110]
Knock-back & Recoil	•	•		[8], [111]
One-shot Particle Effects		•	•	[73], [112]–[117]
Cooldown Visualisation			•	[118]–[121]
Ragdoll Physics	•	•	•	[1], [122]
Colour Flashing		•		[8], [71], [123]
Impact Markers		•	•	[83], [124]
Hit Stop		•	•	[81], [83], [125], [126]
Audio Feedback	•	•	•	[127]–[129]
Haptic Feedback	•	•	•	[83], [130]
Time Manipulation				
Freeze Frames		•	•	[83]
Slow Motion		•	•	[83]
Bullet Time		•	•	[131], [132]
Instant Replays		•	•	[83]
Persistence				
Trails			•	[112], [127], [133]
Decals & Debris			•	[127], [134]
Follow-Through	•			[135]
Fluid Interfaces	•		•	[136]–[138]
Idle Animations			•	[135], [139], [140]
Scene Framing				
Highlighting			•	[141]–[143]
Dynamic Camera		•	•	[71], [144]–[149]

A. Movement and Actions

The first category of design elements is concerned with movement of the character and other objects and with what happens if the character or an object collide. Controlling an on-screen character means navigating the game world and interacting with other characters and objects. Most writing on this aspect of game feel is concerned with 2D games. Dahl & Kraus [89] provide a good starting point for exploring this topic. Normoyle and Jörg [90] look at the trade-off between naturalness of movement and responsiveness of controls. Pignole [91] describes 10 different aspects of how to design controls that feel responsive. While purely grounded in his own experience, these recommendations are easy to pick up and adapt to any game with 2D character movement. In a more extensive study, Fasterhold et al. [92] (see also [95]) provide an overview of parameters for modelling running and jumping in games. This paper also contains an extensive literature review and insights into implementation details of various platformer games. The authors' model features 21 different parameters to describe basic 2D movement. The key argument in this paper is that movement parameters afford [150] level patterns. Mario's [151] jump curve, for example, excellently facilitates precision descents thanks to featuring terminal velocity that makes future positions easier to predict. Super Meat Boy [152], as another example, abruptly interrupts a jump when the jump button is released, which makes hazardous ceiling elements a viable level design choice, since they can be avoided more easily than if the jump would continue. This is shown in Fig. 1.

Hamaluik [153] used screen recordings to measure and reconstruct all relevant parameters for Super Mario World [153]. Game Makers' Toolkit ([81], see also [154]) runs a more informal analysis of the platformer Celeste [155]. Celeste's player controller's source code was published [156],

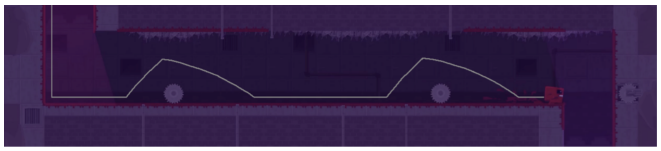


Fig. 1. Super Meat Boy allows the player to interrupt a jump, when the jump button is released, to avoid ceiling elements. Image from [92]

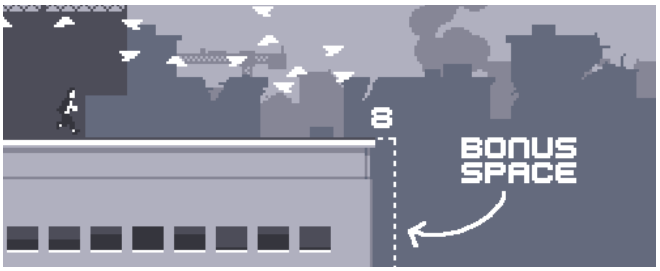


Fig. 2. Illustration of 'Coyote Time' in Canabalt [158], showing the extra distance from the building where a jump is still accepted. Image from [98]

allowing for even deeper analysis. Fiedler [157] provides good starting points for implementing advanced controls and simulations.

1) Basic Movement: This design element is concerned with the most basic parameters defining the interactive movement of an on-screen object, in most cases the player character. The parameters in question are speed, acceleration, friction, and breaking speed (see [89], [90] and [91]). If the player can jump, the strength of the jump force as well as eventual air friction come into play, too. In the case of 2D games, Fasterhold et al. [92] list these and more parameters and how they are related. Saltsman [98] covers movement in one specific platformer, Canabalt [158], in greater detail. Pittman [93] explains the mathematics behind jump mechanics. The exact requirements for tuning the movement of a game is often so deeply connected to the gameplay that it is hard to generalise. An in-depth analysis of the car ball game hybrid Rocket League [159] is presented by Cone [94] and demonstrates how steering of a vehicle is tuned in similar ways to platformer movement.

2) Gravity: The strength of gravity defines how much force pushes an object towards the ground. Games rarely feature earth-like gravity, opting for higher values instead, in order to create a more controlled feeling. Fasterhold et al. [92] list the strength of gravity for a number of platformer games. Earth has a gravitational acceleration constant of 9.807 m/s^2, whereas Super Meat Boy, assuming that the character is 1 meter tall, has a constant of 41 m/s^2 and Super Mario Bros. even features 91.28 m/s^2 [96]. Gravity is used as a sophisticated game mechanic in Super Mario Galaxy [160], where the character can jump from planet to planet and always aligns appropriately with the surface of the cosmic body. Alessi [97] wrote up an explanation and prototypical implementation of this gravity system.

3) Terminal Velocity: The existence of terminal velocity in a movement system means that a falling object does not perpetually get faster. It stops to accelerate at a predefined speed, the terminal velocity. As mentioned above, Mario's [151] jump curve [92] facilitates precision descends thanks to terminal velocity. The additional predictability, that results from the curve becoming a line, supports precision.

4) Coyote Time: The term 'Coyote Time' refers to a movement system that allows a player to still instigate a jump a short time span after running off a cliff[2]. It is perhaps the most famous example of supporting the intent of the player. A detailed account of its technical implementation in the minimal platforming game Canabalt [158] can be found in [98], shown in Fig. 2. Coyote Time is sometimes called 'Coyote Jump' or 'Ghost Jump' [99]. A similar time-based accessibility feature can be found in Disc Room [161], where hitboxes that kill the player get activated only after a delay of up to 50 milliseconds.

5) Invincibility Frames: Short time spans where the player character is invincible. They are a side-effect of player actions like rolling, dodging, respawning, or attacking. SmashPedia [100] lists 23 different cases of invincibility in Super Smash Bros. Ultimate [162], a fighting game, alone. These moments of invincibility are useful for normal players but essential to competitive play and speedrunning. Mora-Zamora and Brenes-Villalobos [101] describe invincibility frames as a tool for balancing risk and reward. Siu et al. [102] mention invincibility frames as part of boss fights. The purpose for introducing a few frames of invincibility is usually to support the player, to give them a carefully measured amount of safety that allows them to pull off even more spectacular actions than if they were vulnerable all the time.

6) Corner Correction: Adjusting a character's path if it would otherwise get stuck in level geometry. This is a common convenience in games where walking is a large part of gameplay. Gilbert [103] analyses how it is implemented in The Legend of Zelda [163] and Doucet [28] offers a detailed analysis of its implementation in Super Mario Bros. 3 [151].

7) Collision Shapes: Collision detection is the problem of determining the intersections of 'hitboxes' (also called 'colliders') of arbitrary shapes and, sometimes, preventing the interpenetration of these colliders to communicate that objects have physical extents in the world. In the case of a 2D game, collision shapes are usually either circles, triangles, or rectangles. In 3D games, they are often spheres, boxes, or capsules. The individual shapes and extents of hitboxes, as well as the coherence between collision shapes and visible game elements, determine how collisions between game elements feel to the player [104].

8) Button Caching: A common player support function is 'Jump Buffering' [92], where the controller code buffers the pressing of the jump button for a few frames and executes the jump after the player has landed. Mario [164] caches the button for 1-2 frames and Braid [165] for 0.23 seconds [92].

9) Spring-locked Modes: This is a user interface modality that is actively maintained by the player by pressing and holding a button. The object they are interacting with 'switches mode' for the duration that the button is held. This form

[2]The name 'Coyote Time' is based on the coyote in the Road Runner series, a character who possesses the power to only fall from a cliff after realising he had been running on thin air for a while.

of interaction is what Raskin [105] calls a 'quasimode' and Johnson & Engelbeck [106] refer to as 'spring-locked mode'. It is often used in order to create anticipation. Games where the player charges an action before unleashing it fall into this category (e.g. Angry Birds [166], SSX Tricky [167], R-Type [168]). Exiting the mode can have a specific effect like the charged shot in R-Type (see [169]), or it just returns the player to the previous mode, like in the case of Dark Souls [170] where the player raises the shield by pushing a button and lowers it by lifting their finger again. Drag and drop is another example of a spring-locked mode that is common in game interfaces.

Further, some games mirror the action of the player and the action of the character. Jumping in the snowboarding game SSX Tricky [167], for example, is charged by pressing a button and holding it. The character jumps at the moment when the button is released. This implementation creates a relation between the game mechanic and the physical action of the player [5] in that the bodily activity of the player releasing the button is more similar to the action of the controlled character than if the jump would be triggered by pressing the button.

10) Assisted Aiming: Some games help a player with the precision required for aiming. Many shooter games support assisted aiming (e.g. Gears of War [171]) and driving games come with countless driving assistance settings[3]. These features can be regarded as manifestations of what Doucet calls 'oil' [28], the measured exploitation of 'illusion of control', as discussed by Kayali & Purgathofer [107]. An extensive description of a particular case of assisted aiming for console shooters can be found in Zimmerman [108].

B. Event Signification

This class of design elements signifies gameplay-relevant events. Similar techniques are used when events are triggered by the player and when they are triggered by the system. All techniques listed in this section are only active for a limited duration. It is usual to layer several effects depending on the significance and kind of event being communicated.

1) Screen Shake: This effect, which is sometimes also referred to as 'camera shake', shakes the camera (or the world) in order to communicate a significant event —- often an explosion, taking damage, or similar high-impact actions. Nijman [8] and Jonasson & Purho [73] both mention screen shake. Lerping and easing functions [109], [110] form the technical basis of the implementation of dynamic cinematography like screen shake. Instead of randomly moving the camera, a carefully selected easing function in a semantically significant direction, communicates more information about what has happened, giving the designer more control over what is communicated to the player.

2) Recoil: When the player character is slightly pushed back, e.g. after firing a gun. Nijman [8] describes an implementation in detail, where the firing of a bullet shakes the screen while also pushing the player character a few pixels back, resulting in a side-effect with gameplay implications. A more sophisticated way of achieving something similar is to use inverse kinematics. God of War [173] uses inverse kinematics to model the reaction of the body of the player character when catching his axe [111]. Not only the arm but the whole body of the character reacts.

3) One-shot Particles: Particle systems [112]–[114] are a staple of juicy game design [73], [115]. Practitioners apply them according to context and sophisticated examples feature many layers of particles accompanied by other techniques from this list, like screen shake and sound effects. Some simple particle systems can be faked using textures (see [8]). Rockenbeck [116] demonstrates a state-of-the-art particle pipeline and explains how it was used in inFAMOUS: Second Son [174]. Vainio [117] describes how this particular system fits into the wider picture of a modern visual effect pipeline.

4) Cooldown Visualisation: Cooldown time is the time it takes until an ability that was just triggered can be used again. Its visualisation has to communicate how long the ability is unavailable as well as the moment it becomes available again. Cooldowns are mostly found in role-playing, where they govern how often spells can be cast or a character ability can be used, and in real-time strategy games, where they govern how long it takes to e.g. research a technology. The duration of the cooldown is usually communicated by greying out the button that triggered an action and gradually revealing it again over the cooldown time. A short overview can be found in [118] and [119]. A detailed study of optimising the display of cooldowns in a custom user interface can be found in [120]. Generally speaking, cooldown visualisations are a subset of progress indicators (see also [121]).

5) Ragdoll Physics: Modelling a character using joints, forces, and rigid bodies, instead of animations. Switching from animation to ragdoll is a staple for communicating that a character has died. Jakobsen [122] wrote about this design element before the name 'ragdoll' became common. Swink describes [1] how they used ragdoll physics in the game Offroad Velociraptor Safari [175] [176]. He also lists a number of games that derive their whole appeal from ragdoll physics.

6) Colour Flashing: This simple but effective technique communicates state changes by overlaying an on-screen graphical object or parts of the screen with a colour. Perry [71] mentions several practical examples of how to indicate damage or other state changes by e.g. flashing the object colour or flashing the whole screen. A special case is flashing an on-screen element that was destroyed before it gets removed from the screen, a technique that creates persistence over time, which is discussed in regards to several other design elements further down this paper. Research indicates that specific colour choices carry different meanings [123]. Nijman [8] demonstrates flashing the enemy sprite white for a frame or two in a 2D platformer to emphasise a hit.

7) Impact Markers: In the absence of a player character, for example in first-person games, other visual elements have to be used to indicate events. In action games, especially in shooters, getting shot at is information of prime importance. Stephenson [124] lists several different techniques for signifying direction, kind, and strength of impact, illustrated by

[3]DiRT 3 [172] features ABS, Dynamic Racing Line, Stability Control, Auto Steer, Corner Braking and Throttle Management — very similar systems can be found in real cars.

game examples. Song [83] explains a number of different elements, most of them covered in their own sections in this overview, specifically for signifying impact. A blend of the colour flashing mentioned above and impact marking is for example achieved with impact lighting, where a light source gets created on impact that illuminates the characters from the point of impact [83].

8) Hit Stop: Animations pause for a brief moment on impact. This effect, sometimes also called 'Impact Freeze', is a staple in fighting and action games and maybe the best researched phenomenon in the area of impact feedback visualisations [83], [125], [126]. Brown [81] describes frame freezes and their design purposes in Celeste [155]. Hit stops are usually introduced in order to communicate feedback about the severity of a hit, but can go further than that. Samurai Gunn [177] features a subtle variation of impact freeze when a character lands on a platform, 'stunning' it for a few frames depending on the height it dropped from. Kratos' axe in God of War [173] freezes when it hits an enemy [111].

9) Audio Feedback: Acoustic channels of communication are a very common way of layering information on top of the graphical representation of a game. Additionally to supporting immersion, audio can also communicate events that happen off screen. Berbece [127] not only highlights the importance of sound effects but also explains how to layer several in order to create an easy to read soundscape. Audio feedback in interaction design for games can be regarded as a specific application of Sonic Interaction Design [128]. Nacke and Grimshaw [129] present research on affective and aesthetic impact of game sound. Overall, sound design is a huge part of game development and offers a rich set of tools and techniques (see e.g., [178], [179]) that are relevant in relation to game feel but too general to cover in this paper.

10) Haptic Feedback: Haptic feedback, often called 'force feedback' or simply 'controller vibration', is a standard functionality of console controllers and built into most mobile phones. It is often used for emphasis rather than as a critical component to interactivity. Orozco et al. [130] provide a complete overview of the history and significance of haptic feedback for games. Most platform holders have clear guidelines about when to use haptic feedback, which means that platform-exclusive titles often exploit these features more than multi-platform games (see [83]).

C. Time Manipulation

While hitboxes and movement are spatial, the other dimension often exploited for game feel is time. Zagal and Mateas [180] give a good overview of game time from an analytical standpoint. The design intent of time manipulation is most often to amplify the experience or to clarify the intensity or direction of an impact. In this section, game time refers to the time of the world simulated in the game whereas real time refers to time in the real world.

All examples in this section have to do with slowing down or pausing game time because games only rarely speed up time to emphasise a moment. SSX Tricky [167] and Bubble Bobble [181] are among the few examples of games that do so[4]. In Drawkanoid [182], a brick destruction game, time speeds up while the player is waiting for the ball to return from a brick's destruction. No research about speeding up time has been found, so this chapter only covers the rest of the cases of time manipulation. Also left out from this section are rhythm and cadence of interactivity. While game designers are doubtlessly aware of the importance of those two aspects to game feel, there is not enough substantial writing to cover them.

1) Freeze Frames: The whole screen is frozen for a short duration, often just a few frames. The difference to hit stops, described above, is that those are a localised phenomenon where one or more on-screen objects get paused, excluding them from the temporal flow of the rest of the game, whereas freeze frames technically halt the progression of game time. Song [83] describes how some games pick the best frame of an animation to freeze on and what gameplay implications frame freezes have.

2) Slow Motion: Slowing down game time for a short duration. Whether applied to replays or to linear game time, slow motion helps to communicate events that would otherwise evolve too fast to be fully perceived by the player. A blend of impact freeze and slow motion can be found in Holedown [183]. The game does not fully freeze on impact, but slows down time to a near halt for a few frames instead. The ability to use slow motion to make an attack look more powerful is mentioned in Song [83].

3) Bullet Time: Bullet time [131] is spring-locked slow motion. It serves as a way to pull off more spectacular or precise actions than the player could accomplish in real-time. They empower the player, amplifying their actions. Porter [132] gives an overview of the history of bullet time in movies as well as games. Technically, bullet time is often eased in and out and maintained for a certain amount of time. This can be modelled using attack-decay-sustain-release 'ADSR' curves (see [1] for details on their various applications).

Turn-based games like XCOM [184] pause time while the player queues actions and subsequently advance it in order to show the results of these actions. Fallout [185] slows down time to a trickle in the V.A.T.S. mechanic, with very similar results. This pattern of letting a user plan a move without time pressure and then showing a lengthy and potentially intense payoff in real time is quite similar to the temporal dynamics of match-three games like Bejeweled [186]. This particular way of manipulating game time could be regarded as an extreme form of bullet time because it essentially fulfils the same purpose and has the same structure.

4) Instant Replay: While replays are technically taking control away from the player, and can as such be regarded as orthogonal to gameplay, they serve a similar purpose to juice insofar as they add clarity and weight to a gameplay situation. Replays of actions that have just happened [83], often slowed down, are usually triggered automatically. The design aspects of this technique, that originally comes from sports television, have not been researched in the context of

[4]A lot of strategy and puzzle games allow users to set the overall speed of the game, which is not part of moment-to-moment interaction.

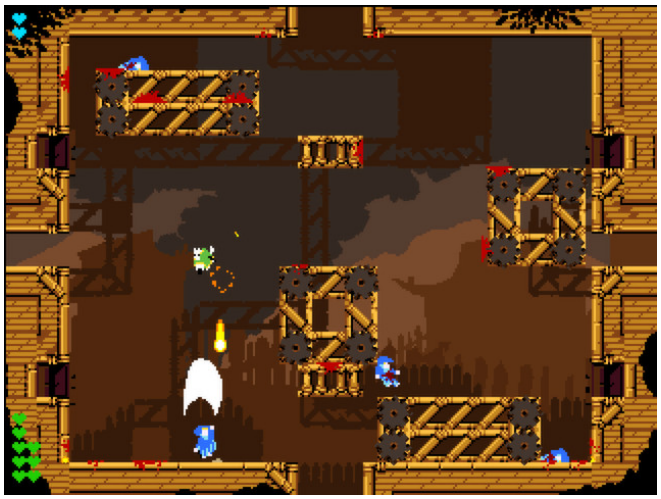


Fig. 3. An intense battle in Samurai Gunn [177], the history of motion and battle encoded in the white sword path, the bullet trajectory, smoke particles from where the gun was fired, as well as blood and gore traces all over the level.

games. The fact that replays communicate pivotal moments of gameplay means that replays might help players in identifying moments of importance. Interestingly, both games mentioned for their use of bullet time above, XCOM [184] and Fallout [185], feature slowed down replays.

D. Persistence

Another aspect related to time is persistence, which could also be called 'temporal consistency'. Broadly speaking, this cluster of techniques uses spatial representation to communicate time-dependent information. The problem being solved is, in the words of Bay-Wei and Ungar: "When the user cannot visually track the changes occurring in the interface, the causal connection between the old state of the screen and the new state of the screen is not immediately clear." [138].

From skid marks to particle trails, the purpose of the techniques listed below is always to encode information about the past in the currently displayed image. Even motion blur, which is mentioned further down the list in III-E, not only prevents temporal aliasing, but retains the history of movement as a lingering after-image. Very often, the below design elements are used in combination and additionally to other elements that communicate the dynamics of the on-screen action. An example of this can be seen in Fig. 3.

Temporal consistency also means a consistent frame rate. While this article neither covers technical details nor how bugs and implementation weaknesses affect game feel, it is important to mention that frame rate and especially the duration of the physics time step have a huge influence on how reactive a game feels. Swink also stresses this when he maintains that "real-time control relies on sustaining three time thresholds: the impression of motion, perceived instantaneous response and continuity of response" [1]. Fiedler [187] provides an excellent introduction into how to implement a stable and reliable feeling core game loop. Cone [94] describes in depth how they solved countless challenges of running a stable physical simulation of the fast moving cars and ball in Rocket League [159].

Overall, temporal consistency techniques are employed in order to allow the player to see either past events or quickly happening events for a longer time. They can be implemented so that world space is used as an interface layer by attaching trails and particle effects to objects that would be invisible in the real world [188]. The key role of these techniques, from a design perspective, is to support the player.

1) Trails: Traces left behind a moving object. The most prevalent example of temporal persistence in games is found in particle systems [112] and trails. Particle systems that amplify the result of player interaction extend the time that result is visible on screen, creating a dynamic and, for a short while, persistent representation of the player's interaction history, which could be seen as trails of a player interaction. Particle systems that leave a trail in space as well as time allow the reconstruction of the trajectory of movement of an object. These techniques increase the readability of a scene for the player as well as spectators [127], [133].

2) Decals & Debris: Decals and debris are stationary traces left in the game world. Birdwell [134] mentions how Valve used decals to acknowledge the actions of the player. Berbece [127] explains a specific design case, where the player character leaves a blotch of paint after being eliminated from a match, in great detail. Destructible levels are a special case of debris left by players that has physical implications and can be regarded as in-game level design.

3) Follow-Through: Follow-through, the effect where a part of an animated character or object keeps moving after the main motion has stopped [135] is also a way of encoding the history of the motion in subsequent frames. This time, the encoding is not done as a non-diegetic overlay or abstraction, but as movement of parts of the object in question.

4) Fluid Interfaces: Introduced by Apple in 2018 [136], 'Fluid Interfaces' aim at offering more natural interaction forms based on aligning and understanding of intent with physical simulation. They aim at maintaining smooth continuity whenever possible. Gitter [137] summarises the original presentation and provides a number of code examples. Continuity comes from temporal persistence and spatial coherence, for example when a user interface transition retains aspects of the previous view, while transitioning to a new one [138]. In games, fluidity can be found in the moment when coins that are earned at the end of a round fly into a virtual purse, each with a short trail. Or when representations of pick-ups linger on screen after being collected and then attach to the character.

5) Idle Animations: Small loops of animation that play after a while once the player stops interacting with their character — when the player character enters the 'idle' state [139], [140]. They are superficial in relation to the core mechanics of the game but nevertheless contribute to the overall experience of a game. Idle animations are of course not triggered directly by the player. On the contrary, they are triggered indirectly by not interacting. Idle animations enhance the illusion of life [135] of the character.

Fig. 4. Character camera-window in Rastan Saga [189], the camera only moves when Rastan pushes against this window. Image from [142]

E. Scene Framing

In racing and flying games, there is a tight link between the field of view, motion blur intensity, and speed. This link defines how the game feels. In 2D games, a variety of techniques are used to enable specific game mechanics, support specific player behaviours, and give specific feedback to players. Keren [141], [142] assembled a great overview of these techniques. An example of a camera window can be seen in Fig. 4. Eiserloh [190] describes the effects of the maths behind camera controls on game feel.

1) Highlighting: Gameplay-relevant elements highlighted on and off screen. A lot of games have sophisticated techniques to direct the gaze of the player by gradually transitioning the camera focus from the player character to a point of interest. A good example of this is mentioned by Keren [141], [142] and explained in further detail by Meyer [143] on hand of his game Insanely Twisted Shadow Planet [191], an exploration game.

2) Dynamic Camera: Articles by Christie et al. [144], Haigh-Hutchinson [145]–[147], and Perry [71] provide good starting points for game camera design. Burelli [148] and Yannakakis et al. [149] examine affective reaction and camera handling. Burelli concludes that interactivity is the key difference between film cinematography and game cinematography, since his study "demonstrates how the impact on the player experience is mediated by her interaction." [148]

F. Summary

This list of elements of game feel design is by no means exhaustive. There are some obvious gaps stemming from how we scoped the review, e.g. narrative design, sound design, and user interface design. We hope that future researchers will use this list as a starting point for further exploration of the topic area. We are also aware that these techniques are differently suitable depending on genre, hardware platform, target group, and other factors of a game. Nevertheless, if a game designer concerns themselves with the above design elements and regards them as a collection of methods to draw from, they will be supported in intentionally conveying a chosen game feel.

IV. Game Feel

The classes of game feel design listed above are connected and, just like Jonasson and Purho keep adding juice in their talk [73], most moments of interacting with a game are shaped by the presence of several layers of feedback. A good example is the backstabbing attack in Dark Souls [170], an action role-playing game. This attack sequence is triggered by sneaking up to a foe from behind and attacking its back. If successful, the camera locks in place, the enemy and the player character get positioned in predetermined spots relative to each other, and weapon-dependent animations and sounds are played. The player is invincible for the duration of the sequence. The design purpose of this feedback set is to give weight to the effect of a single, but carefully prepared, button press. In general, game designers are most concerned with the quality of interactivity in the core loop [15] of a game, but that does not mean that they do not employ a lot of the techniques presented in this paper in all parts of a game.

In the following paragraphs we describe three design domains and what polishing in these domains entails. The domains are physicality, amplification, and support. Physicality is the design domain describing the elements of the game that are a physical simulation. Amplification is the design domain concerned with intensifying the player experience. Support is the design domain where techniques that help the player to control the game are situated. Polishing means something different in each domain and this diversification helps us to talk about game feel design in a precise manner. Table II lists the three domains and their associated polishing task.

A. Tuning of Physicality

The first design domain is the experience of physicality of the system. Swink's [1] whole concept of Game Feel rests on this pillar. Designers shape the feel of the game by tuning the parameters of the physical simulation [92]. Depending on how much the game's core loop relies on the joy of movement, attention to detail can become extremely valuable [98]. Tuning of physicality leads to finely calibrated movement parameters, gravity, and collision shapes. The experience of control is enhanced by additionally applying screen shake, recoil, and knock-back. Appropriate audio design and haptic feedback additionally communicate the physical dynamics of gameplay. It is important to note that for the player, it often does not matter whether physicality is simulated or faked. Tweening [192], specifically with easing functions [109], [110], and various other animation techniques [135], [193] can be used to communicate the desired weight of an object. These can be far easier to read as well as implement than a realistic representation. Generally, tuning exploits our knowledge about physicality in order to make interactivity more predictable.

B. Juicing of Amplification

The second design domain is amplification. It primarily serves two purposes: first, it empowers the player. Second, it communicates the importance of events. Empowerment can take many shapes and forms. Bullet time, one of the most

TABLE II
GAME FEEL DESIGN DOMAINS AND THEIR ASSOCIATED POLISHING TASKS

Design Domain	*Physicality*	*Amplification*	*Support*
Domain Description	The physical simulation.	Intensification of experience.	Enabling the player.
Polishing Task	*Tuning*	*Juicing*	*Streamlining*
Task Description	Setting parameters to specify the behaviour of objects.	Adding feedback to emphasise, clarify and amplify.	Acting on player intent by interpreting the input in context of the gameplay situation.

iconic ways of amplifying player actions, empowers the player to pull off more precise activities than they could if time progressed linearly. At the same time it also signifies that the player has the opportunity to have greater impact during this time interval than during the rest of the game. Impact freeze on the other hand is mostly used to signify a successful interaction. Charging, which is based on spring-locking, is a technique that balances the reward of empowerment — the longer the player presses a button, the bigger the impact — with risk [194].

Juicing the amplification means providing adequate feedback to player actions and creating coherence between different aspects of feedback. Audio, haptic feedback, particle systems, and animation are the most important sources of juice. Juiciness requires exact timing of particle emissions, freeze frames, audio cues, perspective changes, and potentially many more parts of the game. Juicing empowers the player by structuring the reaction of the system to input in a way that amplifies actions adequately for the intended game feel.

C. Streamlining of Support

The third design domain is support. It covers techniques that help the player to execute a challenging action or just provide convenience. Doucet [28] calls the polishing of support mechanics 'oiling', whereas we adopt the less slick term 'streamlining' that he also mentions in his article. Streamlining prevents player frustration by making sure that the player receives help where it supports the experience of the game. Doucet lists a couple of examples of how games can be streamlined in order to support the player. The goal of streamlining is to make rough edges of the game disappear, in order to provide a smooth player experience. Most of the time, the player does not want to realise how much the game is supporting them. "If you do this right then the player wont suspect a thing" says Pulver [195]. Disc Room's [161] designer Nijman explains that their use of Coyote Time "has a bunch of good side effects that make it seem like the game knows your intentions." [104]. A large part of the 5400 lines of code that comprises the Celeste character controller is dedicated to providing forgiveness for the player (see [81] and [196] for an overview of a few of the features implemented for this purpose). This results in controls that are "working on the player's intent rather than making a precise simulation" [81]. A much more sophisticated approach was presented by Zimmerman [108] when they describe the selection mechanism for aiming targets when landing on the ground as one-dimensional optimisation problem.

Elements that enhance temporal consistency offer a different kind of support. Trails that follow projectiles are an example of temporal feedback, since they help to determine the speed and direction of the object by documenting its history. Often particle systems in games have a similar role, and so do skid marks or trails in simulated mud or water. They serve as a visualisation of the past and as an externalisation of information that the player would otherwise have to memorise. That makes them a service for the player. Continuously displayed game elements like status effects and idle animations similarly lift the burden of remembering the state from the player's shoulders by showing it in the game instead.

The act of polishing player supporting game elements is streamlining. Rather than explicitly informing the player about changes in the game's state, streamlining is making the player's experience adequately smooth. A good support system is often invisible [107]. User Experience Design [197], [198] can be used for this design area. A closely related cluster of research concerns game accessibility. While User Experience Design is concerned with setting up game development processes that encompasses user research, the role of accessibility is to widen the audience of games by providing guidelines and tools that make games accessible to players with different accessibility needs [199], [200].

D. Designing Game Feel

Game feel design is minute design work that evokes an affective reaction in the player. The precise reaction is subjective and highly dependent on context, inside and outside the game. Streamlining, tuning, and even juicing are techniques that help with consciously designing interactive challenges at the heart of the player experience. Game feel is a shortcut for describing how this experience feels. If game feel design is the act of fine-tuning the relationship between expected and actual outcome of an interactive process then it must be regarded central to the game design process.

V. FUTURE RESEARCH

'Game feel' is a value-neutral expression. While game designers and scholars are mostly concerned with what they refer to as 'good game feel' (see e.g. [26]), the subjective nature of game feel and the need for "good negative moments" [12] call for a more holistic terminology. What makes a negative moment "good" is a question for future research, especially in terms of how game feel affects players psychologically. In any case, those negative moments, if designed consciously, are a valid aesthetic choice, given that "aesthetics describes the desirable emotional responses evoked in the player, when she interacts with the game system" [88]. Since game feel is

the experience of a game's aesthetic — following Hunicke's use of that term — it spans visual elements, sound design, mechanics, as well as storytelling aspects. Continued exploration of these different game elements and how they contribute to game feel is a worthwhile research endeavour. Sound design, encompassing both sound effects and music, stands out as a field which demands further research. Isolating the aspects that represent polish in different design domains would also be very worthwhile, for example in narrative design, physics simulation, and animation. Applying this research to specific technological areas like virtual and augmented reality could also prove worthwhile. Social aspects and historical perspectives could also yield interesting results, especially if linked to related areas of design beyond digital entertainment, such as board games, physical play, sports, and other leisure activities.

Historical trends and useful insights could be gleaned from case studies that dig deep into various aspects of game feel across genres and over time. Another interesting observation we had while writing this paper is that game feel elements often replace senses that are not part of the vocabulary of the computer. For example, inertia, smell, and the balance of your own body are not directly part of the game. The methods described in this paper are often used as a substitute for them.

Another potential area for future research is game design tools. Recent advances have led to tools for designing feedback [74], [75] and generating it [201], [202]. AI agents and algorithms to help designers analyse and adjust game difficulty [203], [204] and the flow of game play and levels [205]–[207] have been explored academically. Tools that support automatic game design [208], [209] may also benefit from research into feedback readability and assistance in creating the right feedback. Systems that either partially or fully generate games could benefit from being able to evaluate whether different parts of a game fit together (see also [5]).

Research on the effect of feedback on the readability of a game by an autonomous agent is sparse for now. Most of the General Video Game Playing research [210]–[214] is powered by the Video Game Description Language [210], [215], where feedback is almost non-existent. If game feel can provide support to human players, it might also be able to help AI agents.

Designers often work on game feel in very intuitive ways, and literature about game feel is very domain-specific. Most written records are by practitioners who are discussing their own projects. There are large areas that have not been reflected upon. Some of these relate to unusual game mechanics, for example, speeding up time. Others are of more general relevance, for example the link between audio design and game feel. The purpose of this paper is to give an overview of existing research and techniques in order to make game feel more accessible to game designers and researchers.

Ultimately, we hope this paper stimulates the creation of more nuanced and reflective design processes, the development of better design tools, and improved game design. If, as Keogh [216] puts it, "Mechanics are the skeleton. 'Polish' or 'feel' or 'juice' is the meat.", then a more precise vocabulary is a step towards cooking up better games.

ACKNOWLEDGMENT

The authors would like to thank Sebastian Risi, Hans-Joachim Backe, Dom Ford, Karin Ryding, Sílvia Fornós, Christian Hviid Mortensen, Charlene Putney, Miruna Vozaru, and Miguel Sicart for their support, feedback, proof reading, and inspiration. We also want to thank Mike Cook, Martin Jonasson, and Steve Swink for great discussions about game feel. And finally the authors want to bow their heads to Petri Purho and Jan Willem Nijman for having great (and ultimately similar) ideas about game feel and continuously talking about them.

REFERENCES

[1] S. Swink, *Game Feel*. Morgan Kaufmann, 2009.
[2] D. Wilson, "A Tale of Two Jousts: Multimedia, Game Feel, and Imagination," 2016. [Online]. Available: https://www.youtube.com/watch?v=hpdcek4hLA8
[3] J. Schell, *The Art of Game Design: A Book of Lenses*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.
[4] T. Rogers, "In Praise of Sticky Friction," Jun. 2010. [Online]. Available: https://kotaku.com/in-praise-of-sticky-friction-5558166
[5] A. Anthropy and N. Clark, *A Game Design Vocabulary: Exploring the Foundational Principles behind Good Game Design*, 1st ed. Addison-Wesley Professional, 2014.
[6] M. Ehrndal, "A holistic approach to designing for a specific aesthetic experience in digital games," Master Thesis, Malmö högskola, Malmö, Sweden, 2012. [Online]. Available: http://muep.mau.se/handle/2043/13942
[7] L. J. Larsen, "Collision Thrills: Unpacking the Aesthetics of Action in Computer Games," *Journal of Computer Games and Communication*, vol. 1, no. 1, pp. 41–52, Apr. 2016. [Online]. Available: https://www.macroworldpub.com/makale_detay.php?makale_id=92&dergi_id=55#.VyMdGz9NjaY
[8] J. W. Nijman, "The art of screenshake," Dec. 2013. [Online]. Available: https://www.youtube.com/watch?v=AJdEqssNZ-U
[9] Robert Yang, "Queer Futures in Game Feel," Oct. 2018. [Online]. Available: https://www.blog.radiator.debacle.us/2018/10/queer-futures-in-game-feel.html
[10] K. Vonnegut, "The Shapes of Stories," 1985. [Online]. Available: https://www.youtube.com/watch?v=oP3c1h8v2ZQ
[11] S. Kumari, S. Deterding, and J. Freeman, "The Role of Uncertainty in Moment-to-Moment Player Motivation: A Grounded Theory," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play - CHI PLAY '19*. Barcelona, Spain: ACM Press, 2019, pp. 351–363. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3311350.3347148
[12] S. Sivak, "GAME 3400 Level Design - Moment Based Design," 2012. [Online]. Available: https://www.slideshare.net/sjsivak/game-3400-level-design-moment-based-design
[13] S. Swink, "Game Feel: The Secret Ingredient," 2007. [Online]. Available: https://www.gamasutra.com/view/feature/130734/game_feel_the_secret_ingredient.php?print=1
[14] D. Saffer, *Microinteractions: Designing with Details*. Reilly Media, Inc., 2013.
[15] M. A. Sicart, "Loops and metagames: Understanding game design structures," in *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015), June 22-25, 2015, Pacific Grove, CA, USA.*, 2015.
[16] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. New York: Harper and Row, 1990.
[17] P. Sweetser and P. Wyeth, "GameFlow: A model for evaluating player enjoyment in games," *Computers in Entertainment*, vol. 3, no. 3, p. 3, Jul. 2005. [Online]. Available: https://doi.org/10.1145/1077246.1077253
[18] J. Juul, *Half-Real: Video Games between Real Rules and Fictional Worlds*. MIT Press, 2005.
[19] D. Ciccoricco, "Narrative, Cognition, and the Flow of Mirror's Edge:," *Games and Culture*, Jul. 2012. [Online]. Available: https://journals.sagepub.com/doi/10.1177/1555412012454223
[20] DICE, "Mirror's Edge," DICE, 2008.

[21] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, p. 31, Apr. 2007. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1232743.1232769

[22] That game company, "Flow," That game company, 2006.

[23] ——, "Flower," That game company, 2009.

[24] ——, "Journey," That game company, 2012.

[25] J. Juul and J. S. Begy, "Good Feedback for bad Players? A preliminary Study of 'juicy' Interface feedback," in *Proceedings of First Joint FDG/DiGRA Conference*, Dundee, 2016, p. 2. [Online]. Available: https://www.jesperjuul.net/text/juiciness.pdf

[26] K. Hicks, P. Dickinson, J. Holopainen, and K. Gerling, "Good Game Feel: An Empirically Grounded Framework for Juicy Design," in *Proceedings of the 2018 DiGRA International Conference: The Game Is the Message*. DiGRA, Jul. 2018, p. 17. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/DIGRA_2018_Paper_35.pdf

[27] K. Hicks, T. Pike, K. Gerling, O. Burman, G. Richardson, and P. Dickinson, "Understanding the Effects of Gamification and Juiciness on Players," in *Proceedings of the IEEE Conference on Games 2019*. London, United Kingdom: IEEE, Aug. 2019, p. 8.

[28] L. Doucet, "Oil it or Spoil it!" Aug. 2016. [Online]. Available: https://www.fortressofdoors.com/oil-it-or-spoil-it/

[29] M. Sicart, *Play Matters*. MIT Press, 2014.

[30] J. P. Gee, *Good Video Games+ Good Learning: Collected Essays on Video Games, Learning, and Literacy*. Peter Lang, 2007.

[31] R. Koster, *A Theory of Fun for Game Design*. Scottsdale, AZ: Paraglyph Press, 2005.

[32] B. Sutton-Smith, *The Ambiguity of Play*, ser. The Ambiguity of Play. Cambridge, MA, US: Harvard University Press, 1997.

[33] I. Iacovides, A. L. Cox, P. McAndrew, J. Aczel, and E. Scanlon, "Game-Play Breakdowns and Breakthroughs: Exploring the Relationship Between Action, Understanding, and Involvement," *Human–Computer Interaction*, vol. 30, no. 3-4, pp. 202–231, May 2015. [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/07370024.2014.987347

[34] M. Pichlmair, "Designing for emotions: Arguments for an emphasis on affect in design," Ph.D. dissertation, Vienna University of Technology, Vienna, Austria, 2004.

[35] H. Gedenryd, "How designers work - making sense of authentic cognitive activities," *Lund University Cognitive Studies*, vol. 75, pp. 1–123, 1998.

[36] J. Dewey, "The Quest for Certainty: A Study of the Relation of Knowledge and Action," *The Journal of Philosophy*, vol. 27, no. 1, pp. 14–25, 1930. [Online]. Available: https://www.pdcnet.org/pdc/bvdb.nsf/purchase?openform&fp=jphil&id=jphil_1930_0027_0001_0014_0025

[37] D. Kao, "The Effects of Juiciness in an Action RPG," *Entertainment Computing*, vol. 34, p. 100359, Feb. 2020.

[38] J. Juul, *A Casual Revolution — The MIT Press*. MIT Press, 2009. [Online]. Available: https://mitpress.mit.edu/books/casual-revolution

[39] S. Deterding, "The Lens of Intrinsic Skill Atoms: A Method for Gameful Design," *Human–Computer Interaction*, vol. 30, no. 3-4, pp. 294–335, May 2015. [Online]. Available: https://doi.org/10.1080/07370024.2014.993471

[40] S. Atanasov, "Juiciness: Exploring and designing around experience of feedback in video games," Master Thesis, Malmö högskola, Malmö, Sweden, 2013. [Online]. Available: http://muep.mau.se/handle/2043/15692

[41] Z. Gage, "Building Games That Can Be Understood at a Glance," 2018. [Online]. Available: https://www.youtube.com/watch?v=YISKcRDcDJg&ab_channel=GDC

[42] C. Carlsson and A. Pelling, "Designing Spectator Interfaces for Competitive Video Games," Master Thesis, Chalmers University of Technology, Gothenburg, Sweden, 2015. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/224247/224247.pdf

[43] R. Hunicke, "Loving Your Player With Juicy Feedback," dConstruct 2009, 2009. [Online]. Available: http://2009.dconstruct.org/podcast/juicyfeedback

[44] T. Fullerton, *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*, 3rd ed. A K Peters/CRC Press, Apr. 2014.

[45] P. Suddaby, "5 Important Ways to Add Polish to Your Game," May 2013. [Online]. Available: https://gamedevelopment.tutsplus.com/articles/5-important-ways-to-add-polish-to-your-game--gamedev-7642

[46] L. Brown, "The Nuance of Juice," Vector, 2016. [Online]. Available: https://www.youtube.com/watch?v=qtgWBUIOjK4

[47] R. F. Baumeister, K. D. Vohs, C. Nathan DeWall, and Liqing Zhang, "How Emotion Shapes Behavior: Feedback, Anticipation, and Reflection, Rather Than Direct Causation," *Personality and Social Psychology Review*, vol. 11, no. 2, pp. 167–203, May 2007. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1088868307301033

[48] J. T. Larsen, A. P. McGraw, and J. T. Cacioppo, "Can people feel happy and sad at the same time?" *Journal of Personality and Social Psychology*, no. 81(4), pp. 684–696, 2001. [Online]. Available: https://doi.apa.org/record/2001-18605-010?doi=1

[49] J. P. Forgas and J. V. Ciarrochi, "On Managing Moods: Evidence for the Role of Homeostatic Cognitive Strategies in Affect Regulation," *Personality and Social Psychology Bulletin*, vol. 28, no. 3, pp. 336–345, Mar. 2002. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0146167202286005

[50] H. Okon-Singer, T. Hendler, L. Pessoa, and A. J. Shackman, "The neurobiology of emotion – cognition interactions: Fundamental questions and strategies for future research," *Frontiers in Human Neuroscience*, vol. 9, Feb. 2015. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fnhum.2015.00058/abstract

[51] J. Löwgren, "Pliability as an experiential quality: Exploring the aesthetics of interaction design," *Artifact: Journal of Design Practice*, vol. 1, no. 2, pp. 85–95, 2007.

[52] ——, "Toward an articulation of interaction esthetics," *New Review of Hypermedia and Multimedia*, vol. 15, no. 2, pp. 129–146, Aug. 2009. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/13614560903117822

[53] G. Kirkpatrick, "Between Art and Gameness: Critical Theory and Computer Game Aesthetics," *Thesis Eleven*, vol. 89, no. 1, pp. 74–93, May 2007. [Online]. Available: https://doi.org/10.1177/0725513607076134

[54] ——, "Controller, Hand, Screen: Aesthetic Form in the Computer Game," *Games and Culture*, vol. 4, no. 2, pp. 127–143, Apr. 2009. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1555412008325484

[55] ——, *Aesthetic Theory and the Video Game*. Manchester ; New York : New York: Manchester University Press ; distributed in the United States exclusively by Palgrave Macmillan, 2011.

[56] V.-M. Karhulahti, "A kinesthetic theory of videogames: Time-critical challenge and aporetic rhematic," *Game Studies*, vol. 13, no. 1, 2013.

[57] C. Hodent, "Skill-Building Series: Emotion in Game Design (A UX Perspective)," 2020. [Online]. Available: https://www.gdcvault.com/play/1026790/Skill-Building-Series-Emotion-in

[58] D. A. Norman, *Emotional Design: Why We Love (or Hate) Everyday Things*. New York: Basic Books, 2005.

[59] M. Hassenzahl, S. Diefenbach, and A. Göritz, "Needs, affect, and interactive products–Facets of user experience," *Interacting with computers*, vol. 22, no. 5, pp. 353–362, 2010.

[60] S. Long, "What Is Games User Experience (UX) and How Does It Help?" Oct. 17. [Online]. Available: https://www.gamasutra.com/blogs/SebastianLong/20171002/306649/What_Is_Games_User_Experience_UX_and_How_Does_It_Help.php

[61] K. Spiel and K. Gerling, "The surrogate body in play," in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 397–411. [Online]. Available: https://doi.org/10.1145/3311350.3347189

[62] B. Keogh, *A Play of Bodies: How We Perceive Videogames*. Cambridge, MA: MIT Press, 2018.

[63] M. Merleau-Ponty, *Phenomenology of Perception*. Routledge, 1982.

[64] G. Bateson, *Steps to an Ecology of Mind: Collected Essays in Anthropology, Psychiatry, Evolution, and Epistemology*. University of Chicago Press, 1972.

[65] G. Weiss, *Body Images: Embodiment as Intercorporeality*. New York: Routledge, 1999.

[66] D. Surman, "Pleasure, spectacle and reward in Capcom's Street Fighter series David Surman," *Videogame, player, text*, pp. 204–221, 2007.

[67] B. Davnall, "Dr Johnson's Sore Toe: Touch, Naturalism and Kingdom Hearts," Sep. 2016. [Online]. Available: http://startswithafish.blogspot.com/2016/09/dr-johnsons-sore-toe-touch-naturalism.html

[68] C. Putney, "Praise the Sun: On Yoga and Dark Souls," May 2016. [Online]. Available: http://alphachar.com/praisethesun

[69] D. Sudnow, *Pilgrim in the Microworld*. New York, N.Y: Warner Books, 1983.

[70] K. Gray, K. Gabler, S. Shodhan, and M. Kunic, "How to Prototype a Game in Under 7 Days," 2005. [Online]. Available:

https://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php
[71] L. Perry, "The single most useful advice I can give for making any game better.. feedback," 2013. [Online]. Available: https://gamasutra.com/blogs/LeePerry/20130506/191739/The_single_most_useful_advice_I_can_give_for_making_any_game_better_feedback.php
[72] M. Brown, "Secrets of Game Feel and Juice," 2015. [Online]. Available: https://www.youtube.com/watch?v=216_5nu4aVQ
[73] M. Jonasson and P. Purho, "Juice it or lose it," 2012. [Online]. Available: https://www.youtube.com/watch?v=Fy0aCDmgnxg
[74] R. Forestié, "Best Practices for fast game design in Unity," Unite LA 2018, 2018. [Online]. Available: https://www.youtube.com/watch?v=NU29QKag8a0
[75] ——, "How to design with feedback and game feel in mind - Shake it 'til you make it," Unite Copenhagen 2019, 2019. [Online]. Available: https://www.youtube.com/watch?v=yCKI9T3sSv0
[76] N. Suttner, A. Nealen, Z. Gage, and D. Wilson, "Eggplant: The Secret Lives of Games (formerly The Spelunky Showlike) 42: The Secrets of Simplicity with Martin Jonasson." [Online]. Available: https://thespelunkyshowlike.libsyn.com/42-the-secrets-of-simplicity-with-martin-jonasson
[77] ——, "Eggplant: The Secret Lives of Games (formerly The Spelunky Showlike) 39: The Tricks of the Toolkit with Mark Brown." [Online]. Available: https://thespelunkyshowlike.libsyn.com/39-gmtk
[78] ——, "Eggplant: The Secret Lives of Games (formerly The Spelunky Showlike) 38: The Rhythms and Layers of Ryan Clark." [Online]. Available: https://thespelunkyshowlike.libsyn.com/38-the-rhythms-and-layers-of-ryan-clark
[79] ——, "Eggplant: The Secret Lives of Games (formerly The Spelunky Showlike) 36: Game Feel as Procrastination with Jan Willem Nijman." [Online]. Available: https://thespelunkyshowlike.libsyn.com/36-game-feel-as-procrastination-with-jan-willem-nijman
[80] R. Clark, "The Clark Tank," 2019.
[81] M. Brown, "Why Does Celeste Feel So Good to Play? — Game Maker's Toolkit," 2019. [Online]. Available: https://www.youtube.com/watch?v=yorTG9at90g
[82] C. McEntee, "Rayman Origins," *Game Developer Magazine - October 2012*, pp. 26–31, 2012.
[83] J. Song, "Improving the Combat 'Impact' Of Action Games," Apr. 2005.
[84] J. Turner, "Oh My! That Sound Made the Game Feel Better!" 2015. [Online]. Available: https://www.gdcvault.com/play/1022808/Oh-My-That-Sound-Made
[85] R. Ismail, "Six stages of game dev community development," 2015. [Online]. Available: https://www.gamasutra.com/blogs/RamiIsmail/20150504/242486/Six_stages_of_game_dev_community_development.php
[86] D. Cook, "What are game mechanics?" Oct. 2006. [Online]. Available: https://lostgarden.home.blog/2006/10/24/what-are-game-mechanics/
[87] ——, "Loops and Arcs," Apr. 2012. [Online]. Available: https://lostgarden.home.blog/2012/04/30/loops-and-arcs/
[88] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A Formal Approach to Game Design and Game Research," in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, May 2004, p. 1722.
[89] G. Dahl and M. Kraus, "Measuring how game feel is influenced by the player avatar's acceleration and deceleration: Using a 2D platformer to describe players' perception of controls in videogames," in *Proceedings of the 19th International Academic Mindtrek Conference on - AcademicMindTrek '15*. Tampere, Finland: ACM Press, 2015, pp. 41–46. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2818187.2818275
[90] A. Normoyle and S. Jörg, "Trade-offs between responsiveness and naturalness for player characters," in *Proceedings of the Seventh International Conference on Motion in Games - MIG '14*. Playa Vista, California: ACM Press, 2014, pp. 61–70. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2668064.2668087
[91] Y. Pignole, "Platformer controls: How to avoid limpness and rigidity feelings," 2014. [Online]. Available: https://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php
[92] M. Fasterholdt, M. Pichlmair, and C. Holmgård, "You Say Jump, I Say How High? Operationalising the Game Feel of Jumping," in *Proceedings of the First International Joint Conference of DiGRA and FDG*. Dundee, Scotland: Digital Games Research Association and Society for the Advancement of the Science of Digital Games, 2016. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/paper_248.pdf
[93] K. Pittman, "Math for Game Programmers: Building a Better Jump," 2016. [Online]. Available: https://www.youtube.com/watch?v=hG9SzQxaCm8&ab_channel=GDC
[94] J. Cone, "It IS Rocket Science! The Physics of 'Rocket League' Detailed," 2018. [Online]. Available: https://www.gdcvault.com/play/1025341/It-IS-Rocket-Science-The
[95] A. Summerville, J. Osborn, C. Holmgård, and D. W. Zhang, "Mechanics Automatically Recognized via Interactive Observation: Jumping," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*. New York, NY, USA: Association for Computing MachineryNew YorkNYUnited States, 2017, pp. 1–10.
[96] A. Lefky and A. Gindin, "Acceleration Due to Gravity: Super Mario Brothers," 2007.
[97] J. Alessi, "Games Demystified: Super Mario Galaxy," 2008. [Online]. Available: https://www.gamasutra.com/view/feature/131997/games_demystified_super_mario_.php
[98] A. Saltsman, "Tuning Canabalt," 2010. [Online]. Available: https://www.gamasutra.com/blogs/AdamSaltsman/20100929/88155/Tuning_Canabalt.php
[99] M. Venturelli, "Game Feel Tips I: The Ghost Jump," 2014. [Online]. Available: https://gamasutra.com/blogs/MarkVenturelli/20140810/223001/Game_Feel_Tips_I_The_Ghost_Jump.php
[100] Smashpedia, "Invincibility frame." [Online]. Available: https://bit.ly/2JEPgrz
[101] R. Mora-Zamora and E. Brenes-Villalobos, "Integrated framework for game design," in *Proceedings of the IX Latin American Conference on Human Computer Interaction*, ser. CLIHC '19. New York, NY, USA: Association for Computing Machinery, Sep. 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1145/3358961.3358984
[102] K. Siu, E. Butler, and A. Zook, "A programming model for boss encounters in 2D action games," Experimental AI in Games: Papers from the AIIDE Workshop, Technical Report WS-16-22, 2016. [Online]. Available: https://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/14058
[103] T. Gilbert, "Movement Mechanics," 2012. [Online]. Available: https://troygilbert.com/deconstructing-zelda/movement-mechanics/
[104] A. Wiltshire, "How hitboxes work," Aug. 2020. [Online]. Available: https://www.pcgamer.com/how-hitboxes-work/
[105] J. Raskin, *The Humane Interface: New Directions for Designing Interactive Systems*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.
[106] J. Johnson and G. Engelbeck, "Modes Survey Results," *SIGCHI Bull.*, vol. 20, no. 4, pp. 38–50, Apr. 1989. [Online]. Available: https://doi.org/10.1145/67243.67248
[107] F. Kayali and P. Purgathofer, "Two halves of play-Simulation versus abstraction and transformation in sports videogames design," *Eludamos. Journal for Computer Game Culture*, vol. 2, no. 1, pp. 105–127, 2008.
[108] C. Zimmerman, "Reading the Player's Mind Through His Thumbs: Inferring Player Intent Through Controller Input," 2010. [Online]. Available: https://www.gdcvault.com/play/1012339/Reading-the-Player-s-Mind
[109] R. Penner, *Robert Penner's Programming Macromedia Flash MX*. New York: McGraw-Hill/Osborne, 2002.
[110] A. Sitnik and I. Solovev, "Easing Functions Cheat Sheet," Accessed: 2020-05-04 14:25:58. [Online]. Available: http://easings.net/
[111] C. Barlog, "Why Kratos' Axe Feels SO Powerful — Game Mechanics Explained," May 2018. [Online]. Available: https://www.youtube.com/watch?v=zpr-EE2In1M&ab_channel=Polygon
[112] W. T. Reeves, "Particle Systems A Technique for Modeling a Class of Fuzzy Objects," *ACM Transactions on Graphics*, vol. 2, no. 2, p. 17, Apr. 1983. [Online]. Available: https://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/fuzzyParticles.pdf
[113] T. Ilmonen and J. Kontkanen, "The Second Order Particle System," *Journal of WSCG*, vol. 11, no. 1, 2003.
[114] L. Latta, "Building a Million-Particle System," 2004. [Online]. Available: https://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php
[115] N. Lovato, "Squeezing more juice out of your game design!" Mar. 2015. [Online]. Available: https://bit.ly/2GsjSex
[116] B. Rockenbeck, "The inFAMOUS: Second Son Particle System Architecture," 2014. [Online]. Available: https://www.gdcvault.com/play/1020367/The-inFAMOUS-Second-Son-Particle
[117] M. Vainio, "The Visual Effects of inFAMOUS: Second Son," 2014. [Online]. Available: https://www.youtube.com/watch?v=o2yFxPY2b1o&ab_channel=GDC

[118] "Cooldowns can be used to balance games." [Online]. Available: https://game-design-snacks.fandom.com/wiki/Cooldowns_can_be_used_to_balance_games

[119] J. Griesemer, "Design by Numbers: Cooldowns," Jan. 2012. [Online]. Available: https://rewardingplay.com/2012/01/09/design-by-numbers-cooldowns/

[120] D. King, "Principals of UI Design in the World of Warcraft," Dec. 2019. [Online]. Available: https://medium.com/@d.w.king12/principals-of-ui-design-in-the-world-of-warcraft-19e1a33feb61

[121] N. Babich, "The UI/UX Design of Progress Indicator [Trends + Examples]," May 2019. [Online]. Available: https://usersnap.com/blog/progress-indicators/

[122] T. Jakobsen, "Advanced character physics," in *In Proceedings of the Game Developers Conference 2001*, 2001, p. 19.

[123] D. Kao and D. F. Harrell, "Exploring the Impact of Avatar Color on Game Experience in Educational Games," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 1896–1905. [Online]. Available: https://doi.org/10.1145/2851581.2892281

[124] J. Stephenson, "A UX Analysis of First-Person Shooter Damage Indicators," Mar. 2018. [Online]. Available: https://medium.com/@jasper.stephenson/a-ux-analysis-of-first-person-shooter-damage-indicators-59ac9d41caf8

[125] D. Daniels, "Why some games feel better than others - part 3," Mar. 2007.

[126] S. Hurricane, "Impact Freeze," Jan. 2010.

[127] N. Berbece, "Game Feel: Why Your Death Animation Sucks," San Francisco, CA, 2015. [Online]. Available: https://www.gdcvault.com/play/1022759/Game-Feel-Why-Your-Death

[128] K. Franinović, K. Franinovic, and S. Serafin, *Sonic Interaction Design*. MIT Press, 2013.

[129] L. E. Nacke, M. Grimshaw, L. E. Nacke, and M. Grimshaw, "Player-Game Interaction Through Affective Sound," in *Game Sound Technology and Player Interaction: Concepts and Developments*. IGI Global, Jan. 2011. [Online]. Available: https://www.igi-global.com/gateway/chapter/46796

[130] M. Orozco, J. Silva, A. E. Saddik, and E. Petriu, "The Role of Haptics in Games," in *Haptics Rendering and Applications*. London, United Kingdom: IntechOpen, Jan. 2012, pp. 217–234. [Online]. Available: https://www.intechopen.com/books/haptics-rendering-and-applications/-the-role-of-haptics-in-gaming-experience-

[131] M. Sabbagh, "The art of designing visceral and engaging Bullet Time gunplay," Oct. 2015. [Online]. Available: https://michelsabbagh.wordpress.com/2015/10/07/the-art-of-designing-visceral-and-engaging-bullet-time-gunplay/

[132] W. Porter, "A videogame history of bullet-time," 2010. [Online]. Available: https://www.gamesradar.com/a-videogame-history-of-bullet-time/

[133] C. Nutt, "The magic of TowerFall : Depth, simplicity, community," 2015. [Online]. Available: /view/news/241970/The_magic_of_TowerFall_Depth_simplicity_community.php

[134] k. birdwell, "The Cabal: Valve's Design Process For Creating Half-Life," 1999. [Online]. Available: https://www.gamasutra.com/view/feature/131815/the_cabal_valves_design_process_.php

[135] F. Thomas and O. Johnston, *The Illusion of Life: Disney Animation*. New York: Abbeville Press, 1981.

[136] C. Karunamuni, N. de Vries, and M. Alonso, "Designing Fluid Interfaces - WWDC 2018 - Videos," 2018. [Online]. Available: https://developer.apple.com/videos/play/wwdc2018/803/

[137] N. Gitter, "Building Fluid Interfaces," Aug. 2018. [Online]. Available: https://medium.com/@nathangitter/building-fluid-interfaces-ios-swift-9732bb934bf5

[138] B.-W. Chang and D. Ungar, "Animation: From cartoons to the user interface," in *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology - UIST '93*. Atlanta, Georgia, United States: ACM Press, 1993, pp. 45–55. [Online]. Available: http://portal.acm.org/citation.cfm?doid=168642.168647

[139] H. Alexander, "The Quiet Importance Of Idle Animations," 2019. [Online]. Available: https://kotaku.com/the-quiet-importance-of-idle-animations-1834564079

[140] J. Couture, "What makes a great idle animation? Devs share their favorites," 2018. [Online]. Available: /view/news/318163/What_makes_a_great_idle_animation_Devs_share_their_favorites.php

[141] I. Keren, "Scroll Back: The Theory and Practice of Cameras in Side-Scrollers," 2015. [Online]. Available: https://www.gdcvault.com/play/1022243/Scroll-Back-The-Theory-and

[142] ——, "Gamasutra: Itay Keren's Blog - Scroll Back: The Theory and Practice of Cameras in Side-Scrollers," 2015. [Online]. Available: https://gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php

[143] R. Meyer, "ITSP Camera Explained," May 2013. [Online]. Available: https://www.youtube.com/watch?v=aAKwZt3aXQM&feature=emb_title&ab_channel=RyanMeyer

[144] M. Christie, P. Olivier, and J.-M. Normand, "Camera Control in Computer Graphics," *Computer Graphics Forum*, vol. 27, no. 8, pp. 2197–2218, Dec. 2008. [Online]. Available: http://doi.wiley.com/10.1111/j.1467-8659.2008.01181.x

[145] M. Haigh-Hutchinson, "Fundamentals of Real-Time Camera Design," *GDC'05 talk*, p. 20, 2005.

[146] ——, *Real Time Cameras: A Guide for Game Designers and Developers*, 1st ed. San Francisco, Calif. : Oxford: CRC Press, Apr. 2009.

[147] ——, "Real-Time Cameras - Navigation and Occlusion," 2009. [Online]. Available: https://www.gamasutra.com/view/feature/132456/realtime_cameras__navigation_and_.php

[148] P. Burelli, "Virtual Cinematography in Games: Investigating the Impact on Player Experience," in *International Conference On The Foundations of Digital Games*, Chania, Greece, May 2013.

[149] G. N. Yannakakis, H. P. Martínez, and A. Jhala, "Towards affective camera control in games," *User Modeling and User-Adapted Interaction*, vol. 20, no. 4, pp. 313–340, Oct. 2010. [Online]. Available: http://link.springer.com/10.1007/s11257-010-9078-0

[150] D. Norman, *The Design Of Everyday Things*. Basic Books, 1988.

[151] Nintendo, "Super Mario Bros. 3," Game [Nintendo Entertainment System (NES)], 1988.

[152] Team Meat, "Super Meat Boy," 2010. [Online]. Available: http://www.supermeatboy.com

[153] K. Hamaluik, "Super Mario World Physics," Jul. 2012. [Online]. Available: http://blog.hamaluik.ca/posts/super-mario-world-physics/

[154] M. Thorson, "Level Design Workshop: Designing Celeste," 2017. [Online]. Available: https://www.youtube.com/watch?v=4RlpMhBKNr0&feature=youtu.be

[155] Matt Makes Games, "Celeste," 2018.

[156] N. Berry, "Celeste Player Controller Soruce Code," 2018. [Online]. Available: https://github.com/NoelFB/Celeste

[157] G. Fiedler, "Integration Basics," Jun. 2004. [Online]. Available: http://127.0.0.1:1313/post/integration_basics/

[158] A. Saltsman, "Canabalt," 2009.

[159] Psyonix, "Rocket League," 2015.

[160] Nintendo, "Super Mario Galaxy," 2007.

[161] K. Calis, J. W. Nijman, T. Vellmann, and A. Drucker, "Disc Room," 2020.

[162] Nintendo, "Super Smash Bros. Ultimate," 2018.

[163] ——, "The Legend of Zelda," 1986. [Online]. Available: https://en.wikipedia.org/w/index.php?title=The_Legend_of_Zelda&oldid=949668668

[164] ——, "Super Mario Bros. 2," Game [Nintendo Entertainment System (NES)], 1988.

[165] Number None, "Braid," 2008.

[166] Rovio Entertainment, "Angry Birds," Game [iOS], 2009. [Online]. Available: www.angrybirds.com

[167] EA Sports BIG, "SSX Tricky," Game [PlayStation 2], 2001.

[168] Irem Corporation, "R-Type," Game [Arcade], 1987.

[169] M. Alldridge, "'R-Type' - Irem. 1987," Mar. 2014. [Online]. Available: http://www.markalldridge.co.uk/r-type.html#

[170] From Software, "Dark Souls," Game [PlayStation 3], 2009.

[171] Epic Games, "Gears of War," Game [Xbox 360], 2006.

[172] Codemasters, "DiRT 3," 2011.

[173] Ready At Dawn Studios, "God of War," 2018.

[174] Sucker Punch Productions, "inFAMOUS: Second Son," 2014.

[175] Flashbang Studios, "Off-road Velociraptor Safari," 2008.

[176] Diction, "Velociraptor Massacre," Apr. 2012. [Online]. Available: https://www.youtube.com/watch?v=Uh7URFM8rxc&feature=youtu.be&ab_channel=Diction

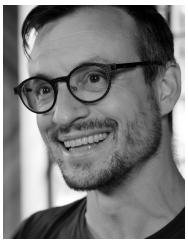
[177] Teknopants, "Samurai Gunn," 2013.

[178] J. Deighan, "Sound Design for Video Games: A Primer," Aug. 19. [Online]. Available: https://www.gamasutra.com/blogs/JamesDeighan/20190823/349296/Sound_Design_for_Video_Games_A_Primer.php

[179] A. Marks, *The Complete Guide to Game Audio: For Composers, Musicians, Sound Designers, and Game Developers*, 2nd ed. Burlington, MA ; Oxford: Focal Press/Elsevier, 2009.

[180] J. P. Zagal and M. Mateas, "Time in Video Games: A Survey and Analysis," *Simulation & Gaming*, vol. 41, no. 6, pp. 844–868, Dec. 2010. [Online]. Available: https://doi.org/10.1177/1046878110375594

[181] Taito Corporation, "Bubble Bobble," Game [Arcade], 1986.
[182] QCF Design, "Drawkaniod," 2018. [Online]. Available: http://www.drawkanoid.com/
[183] grapefrukt games, "Holedown," 2018. [Online]. Available: https://holedown.com/
[184] Firaxis Games, "XCOM: Enemy Unknown," 2012.
[185] Bethesda Softworks LLC, "Fallout 3," Game [PlayStation 3], 2008.
[186] PopCap Games, "Bejeweled," 2001. [Online]. Available: https://www.ea.com/en-gb/games/bejeweled
[187] G. Fiedler, "Fix Your Timestep!" Jun. 2004. [Online]. Available: http://127.0.0.1:1313/post/fix_your_timestep/
[188] I. Iacovides, A. Cox, R. Kennedy, P. Cairns, and C. Jennett, "Removing the HUD: The Impact of Non-Diegetic Game Elements and Expertise on Player Involvement," in *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play - CHI PLAY '15*. London, United Kingdom: ACM Press, 2015, pp. 13–22. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2793107.2793120
[189] Taito Corporation, "Rastan Saga," 1987.
[190] S. Eiserloh, "Math for Game Programmers: Juicing Your Cameras With Math," 2016. [Online]. Available: https://www.gdcvault.com/play/1023557/Math-for-Game-Programmers-Juicing
[191] Shadow Planet Productions, "Insanely Twisted Shadow Planet," 2011.
[192] N. Burtnyk and M. Wein, "Computer-Generated Key-Frame Animation," *Journal of the SMPTE*, vol. 80, no. 3, pp. 149–153, Mar. 1971.
[193] J. Lasseter, "PRINCIPLES OF TRADITIONAL ANIMATION APPLIED TO 3D COMPUTER ANIMATION," *Computer Graphics, Volume 21, Number 4, July 1987*, p. 10, 1987.
[194] M. Pichlmair and F. Kayali, "Intentions, Expectations and the Player," 2008. [Online]. Available: https://www.academia.edu/4989138/Intentions_Expectations_and_the_Player
[195] K. Pulver, "Platforming Ledge Forgiveness," 2013. [Online]. Available: http://kpulv.com/123/Platforming_Ledge_Forgiveness/
[196] M. Thorson, "A short thread on a few Celeste game-feel things." [Online]. Available: https://twitter.com/MattThorson/status/1238338574220546049
[197] R. Bernhaupt, W. Ijsselsteijn, F. F. Mueller, M. Tscheligi, and D. Wixon, "Evaluating user experiences in games," in *Proceeding of the Twenty-Sixth Annual CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI '08*. Florence, Italy: ACM Press, 2008, p. 3905. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1358628.1358953
[198] K. Isbister and N. Schaffer, *Game Usability: Advice from the Experts for Advancing the Player Experience*. San Francisco, Calif. : Oxford: Morgan Kaufmann ; Elsevier Science [distributor], 2008.
[199] T. Westin, I. Hamilton, M. Hinn, and R. van Tol, "Building a Manifesto for Game Accessibility," San Francisco, CA, USA, 2015. [Online]. Available: https://www.gdcvault.com/play/1021849/Building-a-Manifesto-for-Game
[200] "Game accessibility guidelines — A straightforward reference for inclusive game design." [Online]. Available: http://gameaccessibilityguidelines.com/
[201] M. Johansen, M. Pichlmair, and S. Risi, "Squeezer - A Tool for Designing Juicy Effects," in *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 282–286. [Online]. Available: https://doi.org/10.1145/3383668.3419862
[202] T. D. Pettersson, "SFXR," 2007. [Online]. Available: http://www.drpetter.se/project_sfxr.html
[203] A. Nealen, A. Isaksen, and D. Gopstein, "Exploring Game Space Using Survival Analysis," in *Foundations of Digital Games*, 2015.
[204] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Exploring game space of minimal action games via parameter tuning and survival analysis," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 182–194, 2018.
[205] G. Smith, J. Whitehead, and M. Mateas, "Tanagra: A mixed-initiative level design tool," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, ser. FDG '10. New York, NY, USA: Association for Computing Machinery, Jun. 2010, pp. 209–216. [Online]. Available: https://doi.org/10.1145/1822348.1822376
[206] M. Guzdial, N. Liao, and M. Riedl, "Co-Creative Level Design via Machine Learning," *arXiv:1809.09420 [cs]*, Sep. 2018. [Online]. Available: http://arxiv.org/abs/1809.09420
[207] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient Sketchbook: Computer-Aided Game Level Authoring," in *Foundations of Digital Games*, 2013, p. 8.
[208] M. Cook, S. Colton, and J. Gow, "The ANGELINA Videogame Design System—Part I," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, Jun. 2017.
[209] ——, "The ANGELINA Videogame Design System—Part II," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 254–266, Sep. 2017.
[210] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General video game playing," in *Artificial and Computational Intelligence in Games*, ser. Dagstuhl Follow-Ups, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 77–83. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2013/4337
[211] A. Khalifa, M. C. Green, D. Perez-Liebana, and J. Togelius, "General video game rule generation," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. New York, NY, USA: IEEE, Aug. 2017, pp. 170–177. [Online]. Available: http://ieeexplore.ieee.org/document/8080431/
[212] R. D. Gaina, A. Couetoux, D. J. N. J. Soemers, M. H. M. Winands, T. Vodopivec, F. Kirchgesner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 Two-Player GVGAI Competition," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, Jun. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8100955/
[213] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General Video Game AI: A Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms," *arXiv:1802.10363 [cs]*, Feb. 2018. [Online]. Available: http://arxiv.org/abs/1802.10363
[214] M. Johansen, M. Pichlmair, and S. Risi, "Video Game Description Language Environment for Unity Machine Learning Agents," in *2019 IEEE Conference on Games (CoG)*, Aug. 2019, pp. 1–8.
[215] T. Schaul, "A video game description language for model-based or interactive learning," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. Niagara Falls, ON, Canada: IEEE, Aug. 2013, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/6633610/
[216] B. Keogh, "An Incomplete Game Feel Reader," Mar. 2017. [Online]. Available: https://brkeogh.com/2017/03/31/an-incomplete-game-feel-reader/



Martin Pichlmair received a Dr. techn. from Vienna University of Technology in 2007.
He is Associate Professor and Head of the Games Programme at IT University of Copenhagen and co-founder of the Vienna-based indie games studio Broken Rules. Martin is currently focusing his research and practice on procedural and AI-assisted tools for game making.

Mads Johansen recieved a MSc in Games from IT University of Copenhagen in 2014.
He is a PhD Fellow at IT University of Copenhagen and a co-founder of the former Copenhagen-based game studio Glitchnap. Mads is researching mixed-initiative game development tools, with a focus on the prototyping phase of game development.

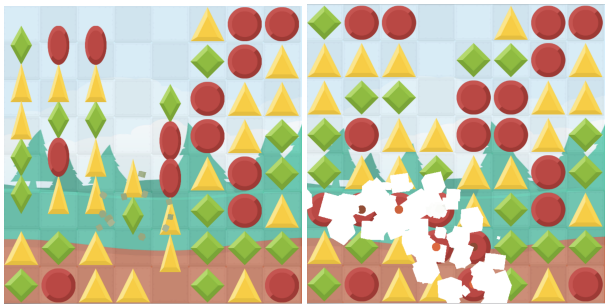
9.5 Challenges in generating juice effects for automatically designed games

Challenges in Generating Juice Effects For Automatically Designed Games

Mads Johansen, [1] **Michael Cook**[2]
[1]IT University of Copenhagen
[2]Queen Mary University of London
madj@itu.dk, mike@possibilityspace.org

Abstract

Automated game design research is usually most concerned with the mechanics and systems of a game, while aesthetics and effects are left to a minimum, if they are considered at all. In this project we integrate *Squeezer*, a tool for generating visual and audio effects (sometimes called *juice*) for games, with *Puck*, an automated game designer. The resulting hybrid system can design games and then generate appropriate sets of effects, making it the first automated game design system that directly engages with 'juiciness' in design. We support this work with a user study, measuring player responses to games with simple animations, effects automatically designed and arranged by Puck, and effects designed and arranged by an expert human designer. We then dissect the engineering challenges presented by integrating the two systems, and the new research questions raised by applying juice through automated game design systems.



(a) Falling pieces squash and stretch as they descend.

(b) Clouds explode out of pieces as they are destroyed.

Figure 1: Example of a generated effect in *SameGame* when pieces are destroyed. This was part of a set used in our study.

Introduction

"Good" videogames are more than just their rules and goals. Modern game design focuses just as much on the minutiae of how these rules and goals are expressed, interpreted and fed back to the player as they interact with the game (Pichlmair and Johansen 2021). Of course, a good set of rules and goals is often vital for games to be solid, playable or challenging. To truly keep a player engaged, however, the game's presentation must also be equally fine-tuned and precise. This is achieved through numerous subtle effects, such as adding animations to a game to help the player understand the transitions between different game states; adding freeze frames, screen flashes or sound effects to punctuate important moments; or adding feedback effects to help the player see the game's response to their action, whether positive or negative (Anthropy and Clark 2014). When a game responds appropriately to player input, it becomes more satisfying to interact with, mimicking tactile and other sensory experiences we have with real-world objects (Norman 1988). In game design parlance, this is often referred to as 'juice' (Jonasson and Purho 2012).

Automated game design (AGD) is a relatively young subfield of game AI research concerned with building AI systems that can autonomously engage in designing and developing games. However, while modern game design has refined the art of juice and it is now a part of everyday vocabulary, AGD research has not yet tackled this topic and primarily focuses on rules, goals and systems (Cook and Smith 2015). This not only misses out on a whole area of interesting research questions and technical challenges in its own right, but also limits the impact and quality of the games they design since even an innovative and groundbreaking game idea can fail to connect with players if it is presented in a way that is flat, limp and lacking in feedback. Indeed, many classic games rely on their juice to deliver their ruleset effectively, which is why juice must not be considered window-dressing for AGD research, but a first-class challenge for automated game designers to tackle.

In this paper, we combine Puck, an AGD system, with Squeezer (Johansen, Pichlmair, and Risi 2020, 2021), a specialist tool for generating effects, thereby complementing the rule/mechanics and goal generation with sets of juicy feedback. We do this to make the game events look and feel more significant, as well as adding feedback to input events in order to let the player know the game acknowledges their actions. This was a considerable engineering task and revealed many new research questions, as well as posing complex technical problems that challenged our standard approach to AGD system design. To evaluate our work, we conducted a user study with more than 100 players, in which we compared three different sets of generated effects (Figure 1 shows an example of a generated effect) against

a simple baseline game with basic 'tweens' (Reeves 1981) and a set of effects hand-designed by the first author, an experienced designer. The designer created this hand-designed effect set using Squeezer, utilising its user-facing GUI and previewing options to design the effects. Our study shows that getting AGD systems to generate effects can improve the perception of their games, but that we still have a way to go before we can outperform human experts. It raises several other important questions about how we evaluate more complex qualities of a game's design, which has many implications for the future of AGD research. Finally, we feel it underlines how important it is to bring more practice-based aspects of game design into AGD research and broaden the field's scope.

The remainder of this paper is as follows: in *Background* we expand on the role of juice in game design, and the history of automated game design, as well as listing work related to this paper; in *Integrating Squeezer With Puck* we describe the engineering process we went through and the challenges we encountered; in *Evaluation* and *Results* we report on the system's output and our user study targeting generated effects in games; in *Future Work* we outline how future AGD systems can respond to and extend these ideas; finally in *Conclusions* we summarise our work.

Background - Juice

The term *juice* first appeared around 2005 in a blogpost by Gray et al. (Gray et al. 2005), where they describe it as

> *...our wet little term for constant and bountiful user feedback. A juicy game element will bounce and wiggle and squirt and make a little noise when you touch it. A juicy game feels alive and responds to everything you do – tons of cascading action and response for minimal user input. It makes the player feel powerful and in control of the world, and it coaches them through the rules of the game by constantly letting them know on a per-interaction basis how they are doing.*

Several practitioners have explained how they go about adding *juice* and other parts of the overarching topic of *game feel* to games, such as Nijman (Nijman 2013), Söderström (Söderström 2009) and perhaps most notably the 'Juice it or Lose it' talk by Jonasson and Purho (Jonasson and Purho 2012), presents the various elements that can take a game from being a flat boring prototype to something that feels fun, exciting and alive.

In an attempt to assist game developers during their prototyping and game jam escapades, Petterson (Petterson 2007) created a small sound effect synthesizer called SFXR. This open-source tool has taken many forms over the years but remains a stable way to quickly generate placeholder sound effects for game developers today. SFXR allows users to generate sound effects from 8 different categories [Coin/Pickup, Laser/Shoot, Explosion, Power-up, Hit/Hurt, Jump, Blip/Select, Random], which randomises values within different parameter presets.

However, the juice design space encompasses not only sound effects, but time dilation, shaking both screen and objects, animating object parameters such as position, rotation and size. Additionally, leaving trails or making clouds of particles is a big part of signalling events using juice. Pichlmair and Johansen (Pichlmair and Johansen 2021) surveyed the most common elements that go into game feel design, which includes a more thorough list of juice effects.

Squeezer

Squeezer[1] (Johansen, Pichlmair, and Risi 2020, 2021) is a conceptual successor to SFXR, expanding the effect synthesis from just sounds to many of the other *juice* effects described in (Jonasson and Purho 2012). Squeezer even includes a version of SFXR for sound effect generation. Squeezer has a GUI that allows designers to build an effect sequence as a tree of cascading effects. Additionally, Squeezer allows a designer to "synthesize" an effect sequence from a category selection, just like SFXR did before it. Squeezer includes various types of simple particle effects, flashes, time manipulation effects, shake effects, scaling, rotating and translating effects, as well as both synthesising sound effects and playing sound files. The simple particle systems even allow effects to be added and designed for individual particles in the same effect sequence. Each effect is designed to expose as simple a set of parameters as possible while still allowing the designer as much flexibility as possible.

While SFXR also allows parameters to be tweaked by the user, it can be hard to tell how each parameter influences the generated sound. In Squeezer, parameters are related to individual effects. It is easy to preview either the entire sequence or a subset of the sequence through enabling/disabling parts of the tree. The GUI also allows setting up events that trigger the effect sequences in Squeezer, which in turn works a lot like the similar *juice* tool MMFeedbacks (Forestié 2018, 2019). Besides, the GUI mode Squeezer contains an API for generating, designing and executing effects.

Background - AGD

Automated Game Design (AGD) is the design and engineering of AI systems that can take on roles in the game design process. This most often takes the form of AI systems that generate games, such as ANGELINA (Cook, Colton, and Gow 2017a,b; Cook, Colton, and Pease 2012), the Game-o-Matic (Nelson and Mateas 2008), or Gemini (Summerville et al. 2018). These systems typically focus on the generation of game rules and mechanics, which biases AGD research towards certain kinds of game, and certain kinds of evaluation and goals, as noted in (Cook and Smith 2015).

Related Work

AGD research stretches back into the history of games research, with a prominent early example being (Pell 1992) in which the author generates chess variants. However, the bulk of AGD research has taken place since around 2010, with systems such as Ludi (Browne and Maire 2010) which was

[1]Squeezer is available at http://github.com/pyjamads/Squeezer and the generator is described under Effect Sequence Generator in the README.md

an AGD system that designed abstract board games. Since Ludi designed games that were to be played with physical game sets, the lack of juice makes some sense (although an AGD system that considers the weight, feel, scale and shape of pieces is an interesting consideration for the field).

Many videogame-based AGD systems emerged concurrently with Ludi or soon after. Some, like ANGELINA 1, focused purely on ruleset generation with little or no consideration for aesthetics. Others took different approaches; for example, the Game-o-Matic allowed users to apply visual theming to games as a co-creative activity, while the system described in (Nelson and Mateas 2007) uses natural language processing to connect input prompts to an existing corpus of game art. In all cases, the AGD system's responsibility only extends to choosing assets for individual game elements. No attempt is made to add visual effects for emphasis or feedback, and juice or user experience is not modelled in any other way.

Some AGD systems do produce juicy games. However, this is almost exclusively due to the system's designers adding juice to the templates used by the AGD system. The best example of this is *Variations Forever* which is a juicy and engaging game that redesigns its rulesets using constraint-based programming (Smith and Mateas 2010). The system's developers added the juice, so although the AGD system does not control it, the resulting generated games benefit from it. A possible exception can also be found in ANGELINA 3, which dynamically sources sound effects and includes them in game when triggered by certain actions (Cook, Colton, and Pease 2012).

Puck

Pumuckl is an AGD system currently under development. It uses an Entity-Component System to describe its games, commonly used in popular game development tools such as Unity. This allows Puck to be easily configured by adding, removing and shaping the components it has available to design with, which makes the system useful both as an autonomous research tool, as well as a co-creative tool guided by a designer. The design of Puck draws inspiration from the open-source AGD system *Bluecap*[2], as well as recent research into how AGD systems can be more tailored towards industry applications (Cook 2020).

We are yet to publish a full report on Puck, but for the purposes of this paper, the system represents a fairly ordinary AGD system prior to its integration with Squeezer. It generates candidate game designs using sets of components and then plays those designs with AI agents to gather data on the game. This data is then used to filter the games and select promising games to extend, perform more playtesting on, or release. Importantly, at the commencement of this work, Puck's game model was separated from any code relating to interactivity or visualisation. This was to facilitate simple testing of the games using AI agents without rendering or user interaction. When presenting one of Puck's games for a human player, we use a separate *visualiser* that can display the game state on the screen, and a *game manager* for handling input. Prior to the work described in this paper, the visualiser was called after each player action, rendering the current game state on the screen with no visual effects or animation.

[2]Available at https://github.com/possibilityspace/Bluecap

Integrating Squeezer With Puck

We aimed to augment Puck with the ability to generate a set of effects for a given game and then load and execute these effects in a standalone build of a generated game. We intended to use Squeezer's existing templates and categories to help guide generation, connected with Puck's internal system with fixed game events that can reliably be listened for. Nevertheless, this integration was more challenging than anticipated and raised interesting engineering questions and problems for Squeezer and Puck.

Generation and Storage

Puck operates by running several different processes in sequence to build and evaluate a game design, from sketching a ruleset through to testing variations of the game with different agents. In order to integrate Squeezer into this, we added a new process to the end of the design phase, which takes a partial game design and generates a set of effects for it. It does this by extracting a list of all the possible events the game can trigger and then generating an effect for any event that has a matching appropriate Squeezer recipe (for example, the event `DestroyPiece` and the Squeezer recipe *Explode*). We decided on appropriate event/category pairings because randomly generated recipes describe huge generative spaces of effects.

The question of where to store these effects was not straightforward. One reason for this is that ontologically speaking, up until now Puck had considered two games with the same ruleset to be identical. This was to help it search the space more effectively by avoiding testing the same game twice. However, two games with the same ruleset but different visual effects should be considered as different. This happens both during a game's development, in playtesting, and after release – games such as Tetris have been remade countless times with variations of their presentation, juice and game feel[3].

We took a compromise solution, where games are still considered unique for the purposes of generation and evaluation, but effect sets can be generated and stored separately. This allows multiple sets of effects to be saved for the same game and loaded dynamically using unique tags. When an effect set is generated, it is saved to the filesystem with a filename that combines the game's name, the targeted effect's name, and a suffix indicating the set it belongs to. For example, `SameGame-DestroyPiece-A` is an effect for a game called SameGame, which is designed to activate in response to the DestroyPiece effect and is from the 'A' set of effects (which in this case means it was the first effect set saved for this game. In Puck's normal execution, it will currently only generated a single set for a game since the system has no way of preferring one effect set over another.

[3]Examples include NES Tetris, Tetris 99 and Tetris Effect

However, our user study makes use of this notation to store multiple effect sets for a single game.

Rendering and Execution

As is common with many AGD systems, Puck's game model is separate from its rendering and view code. This is for two reasons: first, so that we can use AI agents to playtest games rapidly without rendering or player interaction; and second, so that renderers/visualisers could be changed out (indeed, to facilitate research such as this). This poses a problem because Squeezer's effects require the renderer to have specific code to setup, configure and execute effects (for example, it must keep track of objects so that an explosion effect is applied to the right game object).

We resolved this by creating a specialised renderer for Puck's games. It searches the file system when the game begins for the appropriate effect set, adding any effects it finds and matching them to events based on the filename (as described above). It then listens for events from the game and fires effects as required. This means that using a different renderer for the game will result in a playable game but with no effects, thus arguably making it a partial experience of the fully designed game.

From Human Users To AI

After integrating Squeezer and using its recipe templates to generate random effects, we realised that the full scope of effects Squeezer could generate, while useful for a human user who can rapidly filter and curate, were inappropriate for an AGD system that does not experience the effects in the same way as a human user. For example, Squeezer could generate effects that resulted in game pieces becoming invisible at the end of the effect. AI players can still interact with the game board and play, but a human player cannot - even if the game is still theoretically interactive.

A human using Squeezer could easily spot such errors and fix them – or have the awareness to use them effectively (many games temporarily make game objects invisible for specific effects). An AI system needs to be given an understanding of such issues, however. We added filters to the effect design mode to ensure it avoided certain worst-case combinations of parameters. However, we wanted to avoid being too restrictive here and allowed it enough freedom to create effect sets that were bad or unusual (such as rotating/colouring a game object's sprite or permanently making it too big or small). This ensures as much expressivity as possible while also ensuring that situations that render the game completely unplayable are avoided.

Evaluation - User Study

In order to evaluate our system, we conducted a user study in which participants rate several games based on their visual and audio effects. More specifically, we designed the study to test two hypotheses:

- Automatically generating and applying juice effects to a game improves the user's perception of the game.
- Hand-designed juice effects by expert designers are preferred over automatically generated juice effects.

Although juice is designed to improve the user experience, it is not obvious that automatically generated effects will have the same impact on the user. Our first hypothesis seeks to establish whether automated systems can generate and use juice effectively. Well-designed effects go beyond the use of simple templates and event matching, though – they cohere with the overall game design and match the theme, tone and cadence of gameplay. Our second hypothesis seeks to establish that although our automated effects improve the user experience, there is still a gap between our approach and high-level human design intuition.

We chose two games to use in our study: *SameGame* (see Fig. 1), an existing classic game design, and *Antitrust*, a new game designed by Puck, both described below. Our motivation behind this was to test our hypotheses both on well-known and novel game designs, thus accounting for situations where both our participants and our expert designer would be familiar with the underlying game and the converse. For each game, we built five versions, one baseline tween version, one version with effects designed by a human expert, and three versions with generated effect sets. The baseline and generated versions also had their background and sprite art randomly chosen from sets curated by a non-expert. In contrast, the human-designed version had the expert select background and sprite art from the same pool. We describe the baseline and expert setups in more detail below.

Participants were presented with the baseline version, the human-designed version, and a randomly selected generated version for each of the two games. Both the order of the two games and the order of the three versions of each game were randomised to correct for presentation order bias (the games were not interleaved, however – the player sees all versions of one game, then all versions of another). The participants filled out a form where they rated each game version they played on their 'aesthetics and visuals' (with no further definition, to allow for subjective interpretation), on a scale from 1-10 where one is "Very bad" and ten is "Excellent". The participants could fill in the form as they played each version of the game, and at the end, we noted that they were allowed to readjust their ratings. This way, each participant could give an initial rating as they went along and adjust the ratings when they had played all three versions. We did this in the hope that participants rate each version differently and provide a relative ranking of the three versions they saw. Evaluations of the two games were separate, and as a result, a small number of participants only completed one of the halves of the study. Our survey was advertised primarily through social media. We did not collect demographic data on our participants, and participants did not need to complete the games in order to rate them.

Game Versions

Baseline When we initially planned the user study, our most basic visualiser simply showed the current game state, with no animations between the states. However, we quickly realised that with no animations many games become confusing, such as Match-3 games or any games where pieces move, are destroyed, or change state. Even knowing the rules, we would sometimes be confused during testing. In

order to avoid that confusion, especially for generated games that players have probably never played before, we decided to add a minimalist set of tween effects (Reeves 1981; Penner 2002), we added smooth translation when moving pieces around the board, a scale-in tween for spawning and a scale-out tween for destroying pieces. The scale-in/out tweens simply scales the piece in from or out to a size of zero. All rules the game generator can create boil down to spawn, move and destroy. The games do notify the visualiser of other events, such as game start, gain score and end turn, but they are unused by the tween visualiser. These games have the '-Base' suffix in our result section.

Expert Human We decided on making a designed effect set, using the same tool as the game generator, to compare the generated effects with effects designed by a human. We limited ourselves time-wise because it would otherwise be a very unfair comparison. The generator, once running, could potentially generate hundreds of different effect sets in seconds or minutes. However, implementing the juice generator did take a few days, which users of Squeezer would not need to wait for. So we decided to allow the expert designer a day making an effect set for each game. The designed effect sets include effects for game start and gain score events that the juice generator did not have any good preset options for generating. Apart from that the both the designed and generated effects generate an effect set for tapping, spawning, moving[4] and destroying. These games have the '-Expert' suffix in our result section.

Generated The generated effect sets were produced through the method described in the previous section. For each game, the system identifies which messages are in its ruleset and generates an effect for each message according to a preset set of recipes (that is, a `DestroyPiece` effect will always use the 'explode' recipe template, regardless of the game it is generating for). The system uses the default intensity. The generated effect sets were not curated in any way; we generated exactly three effect sets and used them as-is. These games have the '-Gen1/2/3' suffix in our result section. The effect sets were generated uniquely for each game – that is to say, SameGame-Gen1 does not use the same set as Antitrust-Gen1.

Studied Games

SameGame *SameGame* is a popular arcade game originally designed in 1985 and ported to many different platforms. The game takes place on a grid that is randomly filled with coloured tokens. Tapping on a coloured token will destroy it and all tokens of the same colour in a contiguous region, as long as the region is above a certain size (in our implementation, three or more). The player scores more points the larger the region is, thus making the challenge both efficiently clearing the board and maximising the size of the cleared regions. SameGame is particularly popular as a webgame due to its simplicity and accessibility. One of our motivations for including this game was that it is such a well-known casual game, and such games are prone to excessive use of juice (Juul and Begy 2016). It serves as an example where players expect emphasis of their interactions to make them feel fun to keep doing.

[4]Moving is only implemented for *SameGame*, as *Antitrust* does not include a moving mechanic.

SameGame	Base		Generated		Expert
Gen1	4.11	$\sim$	4.06	$\sim$	4.49
Gen2	4.33	$<$	5.36	$\sim$	5.31
Gen3	4.17	$<$	5.23	$\sim$	5.63
Antitrust	Base		Generated		Expert
Gen1	4.69	$\sim$	5.31	$<$	6.54
Gen2	3.95	$\sim$	3.39	$<$	5.08
Gen3	4.42	$\sim$	4.24	$<$	5.24

Table 1: Average ratings given to each build. Ratings are grouped by which generated build participants were shown, indicated by the row. Symbols between columns show significant orderings, or $\sim$ for cases where significance could not be established.

Antitrust *Antitrust* is a two-player game designed by Puck. Players take turns placing tokens of their assigned colour on a 5x5 board. If a player makes a line of four tokens in any direction, those tokens are removed from the board at the start of the opposing player's turn. When the board is full, the game ends, and the player with the most tokens on the board wins. The game plays like a combination of Connect 4 and Gomoku, where players must balance taking over the board, forcing the opponent into making mistakes, and avoiding bad moves which limit their future options. For our study, Antitrust is useful as it is an example of a game both users and the authors are unfamiliar with. There are no existing examples of how to design effects for this game, and therefore it represents the kind of challenge an automated game designer might encounter regularly.

Results

In total, our survey attracted 113 participants. Seven participants failed to complete the form for Antitrust, possibly because the game took longer to play than SameGame, leaving us with 106 participants for the Antitrust games. One participant mistakenly entered data for SameGame in such a way that their ratings were not recoverable, leaving 112 participant records for SameGame. The average ratings for each build along with significance comparisons, are shown in Table 1. Results are broken down according to which version of Gen each participant saw.

Generated Versus Baseline As stated earlier, our first hypothesis is that generated effect sets improves the user's perception of the game. We compared the ratings for the Gen1, Gen2 and Gen3 builds with the Baseline build and ran Welch's t-test to reject the null hypothesis that the average rating for the Base game is not distinguishable from each of the generated builds.

For SameGame, Gen1 is inconclusive, but Gen2 and Gen3 reject the null hypothesis ($p < 0.05$), suggesting they are significantly preferred over the baseline. For Antitrust, the

mean differences are not significant for Gen1, Gen2 or Gen3. Out of six generated builds, two were significantly preferred over the baseline, and four were not significant enough to conclude either way. We discuss these results later in this section, especially with respect to the difference between SameGame and Antitrust.

Expert Versus Generated Our second hypothesis was that hand-designed effect sets would be preferred over automatically generated juice effects. Again, we compared the averages for the three generated builds against the average ratings for the expert build, running Welch's t-test to reject similar null hypotheses.

For SameGame, the expert build was not significantly preferred over Gen1, Gen2 or Gen3. For Antitrust, however, the expert build was significantly preferred over Gen1, Gen2 and Gen3 ($p < 0.05$). For completeness, we also tested that the expert build was preferred over the baseline, which it was in both cases, with very high significance ($p < 0.001$).

Results Discussion

From these results, there is evidence to cautiously support both of our hypotheses, with caveats. One-third of our generated effect sets were significantly preferred over the baseline, and none were significantly less preferred, supporting our first hypothesis, while our expert designs were significantly preferred over half of our generated effect sets, supporting our second hypothesis. However, some results were not significant enough to draw conclusions from.

We believe our first hypothesis is supported, especially when considering the context of this research within automated game design. An automated game designer using our baseline would use the same effects for every game it designed, whereas Puck, augmented with Squeezer, can produce varied and different effect sets for each game it produces. Even if some games are no better than the baseline, over time, it will maintain variation while the baseline becomes familiar, which is important both from the perspective of automated game design, as well as computational creativity (Liapis, Yannakakis, and Togelius 2014).

We also believe our second hypothesis is well-supported. While we are delighted that Puck was able to be competitive with our expert builds in three of the six builds, it failed to be significantly preferred in all six. That the expert build comfortably outperformed both baselines, with significance, is also evidence that good expert design adds something to the perception of a game, reaffirming the importance of this line of research.

Given that some of our participants did not complete the evaluation of Antitrust and none of the generated Antitrust sets was preferred over the baseline, we believe Antitrust itself may have been a less enjoyable game experience than SameGame, resulting in more generally negative ratings from more flashy effects. Tuning the difficulty of the AI player prior to release was difficult, compared to SameGame which naturally tends towards a conclusion after a few turns and cannot really be 'lost' in a meaningful sense. It is also possible that players prefer fewer effects in an adversarial game, with more tension and focus, compared to SameGame's casual, lighter mood. This only serves to underline the importance of the relationship between aesthetics and mechanics, and why AGD research should consider this interplay more prominently.

This user study is not intended to be the final word on the applications of juice to automated game design. Rather, we see this as a jumping-off point. We have clearly shown that simple automatically generated juice can improve the perception of an automatically designed game, but also underlined that even manually designed juice improves the performance of automated game design systems. We hope that this is evidence enough to both encourage AGD researchers to add juice to their systems and to stimulate more research into how AGD systems can understand, model and apply juice to its full potential.

Future Work

This work represents a starting point for research into juice generation for automated game design. A key area of future work from an automated game design perspective is to use data from automated playtesting of games to influence the juice generation process. Many automated game designers use AI agents for playtesting games and gathering data on balance, difficulty and other factors. We believe the same data can be used to identify ways to juice the game - for example, by identifying player actions that are rewarding, rare or significant. We believe this would greatly focus the juice generation and result in more unique and distinct games.

We propose that effect evaluation is another key area future research, that poses a hard challenge for automated game design systems. One way to get around this issue, is to allow the system to make "A/B tests" comparing two sets of generated effects for the same game, through user play testing. By allowing users to rank two variations, the system could potentially find the best option out of a set of generated effects, and in time it might be possible to improve the generation process based on the gathered data.

Additionally, both Squeezer and Puck would benefit from many small additional features to expand their capabilities for this task. Extending Squeezer with a broader range of generator options would give Puck more expressivity in the effects it can generate – in particular, Squeezer lacked a specialised recipe for some common puzzle effects such as pieces being added to the board. Giving Puck an understanding of other key design skills such as basic colour theory and the selection of colour schemes would allow it to augment its effects, backgrounds and sprite selections to cohere together.

Conclusions

In this paper, we reported on the integration of Squeezer, a tool for generating juicy effects for games, and Puck, an automated game designer – the first example, to our knowledge, of an automated game designer that incorporates juice into its model. We discussed the importance of juice in game design and the challenges we encountered in integrating juice into an automated game design workflow and argued for its importance in light of expanding AGD research. We reported on a user study where over 100 participants gave

feedback on their perception of several games, which evidenced that juice can add to the player's perception of a game and that there is still much work to be done before juice generation can match up to human-level design.

Juice and game feel are vital parts of modern game development. Including these disciplines in the scope of AGD research is crucial for the field to grow and properly explore the breadth of game design as a practice. At the same time, researchers interested in building tools like Squeezer, investigating co-creativity and supporting game developers in adding juice to their games will also benefit from working with AGD systems. AGD research helps challenge us to create formal models of juice design and test theories about juice generation at a large scale. We hope this paper is a starting point for much additional research in this area.

Acknowledgements

The authors thank the reviewers for their enthusiastic feedback which improved the paper. The second author was supported by the Royal Academy of Engineering under the Research Fellowship scheme.

References

Anthropy, A.; and Clark, N. 2014. *A Game Design Vocabulary: Exploring the Foundational Principles behind Good Game Design*. Addison-Wesley Professional, 1st edition. ISBN 0-321-88692-5.

Browne, C.; and Maire, F. 2010. Evolutionary Game Design. *IEEE Trans. Comput. Intell. AI Games* 2(1): 1–16.

Cook, M. 2020. Software Engineering For Automated Game Design. In *2020 IEEE Conference on Games (CoG)*, 487–494. ISSN 2325-4289. doi:10.1109/CoG47356.2020.9231750.

Cook, M.; Colton, S.; and Gow, J. 2017a. The ANGELINA Videogame Design System—Part I. *IEEE Transactions on Computational Intelligence and AI in Games* 9(2): 192–203. ISSN 1943-068X. doi:10.1109/TCIAIG.2016.2520256.

Cook, M.; Colton, S.; and Gow, J. 2017b. The ANGELINA Videogame Design System—Part II. *IEEE Transactions on Computational Intelligence and AI in Games* 9(3): 254–266. ISSN 1943-068X. doi:10.1109/TCIAIG.2016.2520305.

Cook, M.; Colton, S.; and Pease, A. 2012. Aesthetic Considerations for Automated Platformer Design. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, USA, October 8-12, 2012*.

Cook, M.; and Smith, G. 2015. Formalizing Non-Formalism: Breaking the Rules of Automated Game Design. In *Proceedings of the 10th International Conference on the Foundations of Digital Games (FDG 2015)*, 5. FDG.

Forestié, R. 2018. Best Practices for Fast Game Design in Unity. https://www.youtube.com/watch?v=NU29QKag8a0. Accessed: 2020-04-17.

Forestié, R. 2019. How to Design with Feedback and Game Feel in Mind - Shake It 'til You Make It. https://www.youtube.com/watch?v=yCKI9T3sSv0. Accessed: 2020-04-17.

Gray, K.; Gabler, K.; Shodhan, S.; and Kunic, M. 2005. How to Prototype a Game in Under 7 Days. https://www.gamasutra.com/view/feature/130848/how_to_prototype_a_game_in_under_7_.php. Accessed: 2019-10-27.

Johansen, M.; Pichlmair, M.; and Risi, S. 2020. Squeezer - A Tool for Designing Juicy Effects. In *Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '20, 282–286. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-7587-0. doi:10.1145/3383668.3419862. https://doi.org/10.1145/3383668.3419862.

Johansen, M.; Pichlmair, M.; and Risi, S. 2021. Squeezer - A Mixed-Initiative Tool for Designing Juice Effects. In *Proceedings of the Foundations of Digital Games Conference (FDG 2021)*, 11. Online.

Jonasson, M.; and Purho, P. 2012. Juice It or Lose It. https://www.youtube.com/watch?v=Fy0aCDmgnxg. Accessed: 2019-02-27.

Juul, J.; and Begy, J. S. 2016. Good Feedback for Bad Players? A Preliminary Study of 'Juicy' Interface Feedback. In *Proceedings of First Joint FDG/DiGRA Conference*, volume Proceedings of first joint FDG/DiGRA Conference, 2. Dundee: DiGRA. https://www.jesperjuul.net/text/juiciness.pdf.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2014. Computational Game Creativity. In *ICCC*, 8.

Nelson, M.; and Mateas, M. 2007. Towards Automated Game Design. In *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence*.

Nelson, M. J.; and Mateas, M. 2008. An Interactive Game-Design Assistant. In *Proceedings of the 13th International Conference on Intelligent User Interfaces - IUI '08*, 90. Gran Canaria, Spain: ACM Press. ISBN 978-1-59593-987-6. doi:10.1145/1378773.1378786. http://portal.acm.org/citation.cfm?doid=1378773.1378786.

Nijman, J. W. 2013. The Art of Screenshake. https://www.youtube.com/watch?v=AJdEqssNZ-U. Accessed: 2021-04-22.

Norman, D. 1988. *The Design Of Everyday Things*. Basic Books.

Pell, B. 1992. METAGAME in Symmetric Chess-Like Games. In *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*.

Penner, R. 2002. *Robert Penner's Programming Macromedia Flash MX*. New York: McGraw-Hill/Osborne. ISBN 978-0-07-222356-9.

Pettersson, T. D. 2007. SFXR. http://www.drpetter.se/project_sfxr.html. Accessed: 2020-07-11.

Pichlmair, M.; and Johansen, M. 2021. Designing Game Feel. A Survey. *IEEE Transactions on Games* IEEE Transactions on Games (Early Access)(IEEE Transactions on

Games (Early Access)): 1–20. ISSN 2475-1510. doi: 10.1109/TG.2021.3072241.

Reeves, W. T. 1981. Inbetweening for Computer Animation Utilizing Moving Point Constraints. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '81*, 263–269. Dallas, Texas, United States: ACM Press. ISBN 978-0-89791-045-3. doi:10.1145/800224.806814. http://portal.acm.org/citation.cfm?doid=800224.806814.

Smith, A.; and Mateas, M. 2010. Variations Forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*.

Söderström, J. 2009. The Four-Hour Game Design by Cactus. https://www.gdcvault.com/play/1243/(304)-The-Four-Hour-Game. Accessed: 2020-07-11.

Summerville, A.; Martens, C.; Samuel, B.; Osborn, J.; Wardrip-Fruin, N.; and Mateas, M. 2018. Gemini: Bidirectional Generation and Analysis of Games via ASP. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 14(1): 123–129. ISSN 2334-0924. https://ojs.aaai.org/index.php/AIIDE/article/view/13013.