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

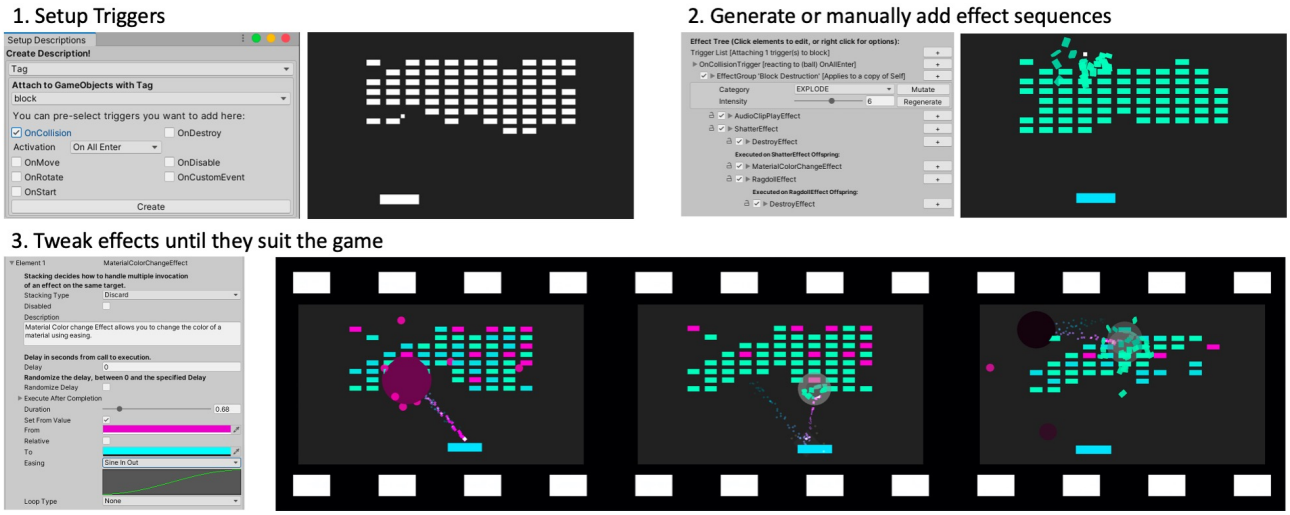

Figure 1: Step (1) take a game prototype and setup the Squeezer event triggers using the setup window. Step (2) add some color and generate or manually add initial effect sequences. Step (3) explore or design parameters of individual effects until they suit the game mechanics.

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.