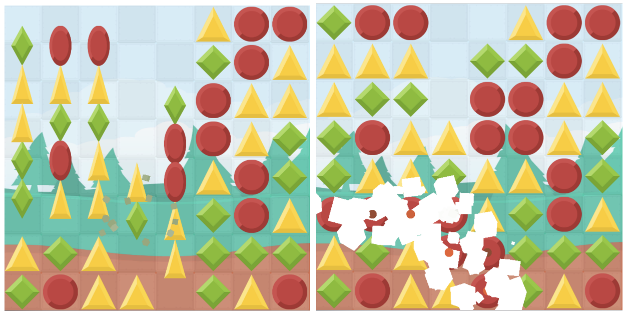
Challenges in Generating Juice Effects For Automatically Designed Games

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 (Pettersson 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

[1]Squeezer is available at http://github.com/pyjamads/Squeezer and the generator is described under Effect Sequence Generator in the README.md

of AGD research has taken place since around 2010, with systems such as Ludi (Browne and Maire 2010) which was 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.

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

[2]Available at https://github.com/possibilityspace/Bluecap

[3]Examples include NES Tetris, Tetris 99 and Tetris Effect

system has no way of preferring one effect set over another. 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.