Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2022)

Puck: A Slow and Personal Automated Game Designer

Michael Cook
School of Electronic Engineering and Computer Science
Queen Mary University of London
mike@possibilityspace.org

Abstract

In this paper we introduce *Puck*, a new automated game design system which combines continuous creativity with an exhaustive approach to content generation. We explain the motivation behind Puck, and in particular its focus on users and small communities. Puck is, to our knowledge, the first automated game designer that can be downloaded and individualise itself through testing and design. We then describe the engineering and structure of the system, detail some initial outputs and evaluation of the system, and future work.

Introduction

Automated game design (AGD) is the science and engineering of AI systems that model, participate in or support the game design process. This can include systems which help users explore game designs (Guzdial et al. 2019; Charity, Khalifa, and Togelius 2020); models of game design theories or frameworks (Togelius and Schmidhuber 2008; Barros et al. 2019); and autonomous game-designing systems (Khalifa et al. 2019; Summerville et al. 2019). It is still an emerging field with no concrete definition, but is becoming increasingly important as artificial intelligence is integrated into more development tools, and the study of computational creativity grows.

The broader field of game AI is inextricably tied to the classical goals of computer science: resource efficiency, solution quality, and high-end scalability. These goals are inherited wholesale by most new subfields in AI, like automated game design, as they provide tried and tested ways of understanding scientific progress in a systems-focused discipline. However, as fields establish themselves these metrics become less useful, and it becomes productive to re-examine them to ensure they are not steering us away from interesting research questions.

In this paper we present Puck, an automated game designer that extends ideas from AGD research (such as continuous design (Cook and Colton 2018)) and procedural content generation research (such as exhaustive generation (Sturtevant and Ota 2018)) and defines a different set of goals for automated game design research, based around community engagement and long-term creativity. Puck is freely downloadable and designed for personal use, making it the first publicly available autonomous videogame designer of its kind, to our knowledge.

The remainder of this paper is organised as follows: in *Background* we introduce the reader to some of the prior work in automated game design and computational creativity which influenced this work, as well as the notion of *exhaustive procedural content generation* which we adapt here for automated game design. In *Motivation* we talk about the reasoning behind developing Puck and our philosophy and design intent for AGD research. In *System* we provide a fuller description of Puck's underlying AI system and its user-level interface. We then discuss the evaluation of a system like Puck and link it to future work for the system.

Background

Automated Game Design

AI research into systems which modify game designs dates back to at least the early 1990s, with Pell's METAGAME (Pell 1992), but the modern community began in the late 2000s, with projects such as Browne's Ludi (Browne and Maire 2010) (which focused on physical games), the Game-o-Matic (Treanor et al. 2012) and projects by Nelson (Nelson and Mateas 2007) and Smith (Smith and Mateas 2010). In the decade and a half since the field has expanded to consider a wide variety of genres ((Cook, Colton, and Gow 2017), (Barros et al. 2019), (Togelius and Schmidhuber 2008), (Khalifa et al. 2019)) as well as attempts to reflect on the nature of the field itself, particularly regarding the clash of rules-as-design versus a more holistic approach (Cook and Smith 2015).

While the boundaries of the field remain blurry, the last decade has also seen increased interest in the development of intelligent tools that participate in the design process such as the Sentient Sketchbook (Liapis, Yannakakis, and Togelius 2013) or Tanagra (Smith, Whitehead, and Mateas 2010). With the proliferation of machine learning we have seen an increase in more active tools, such as research into co-creative design tools (Guzdial et al. 2019). Regardless of the exact boundary lines of automated game design as a field, we can see a growth in AI systems developed to facilitate the design and refinement of core systems and critical content within games.

Continuous Creativity and Presence

In (Cook and Colton 2018) we introduce the notion of *continuously creative* AI systems. Within computational creativity the prevailing paradigm for creative software is akin to a vending machine - the user opens the software, pushes a button requesting a creative artefact, which is eventually dispensed to them. We argued that this misses out on opportunities to position creative systems more strongly in the real world, and proposed that AI systems working in creative domains be built as 'always-on' systems that move between many different activities, only one of which is outputting works.

While many areas of AI and creativity research emphasise the quality of a system's outputs, or *product*, computational creativity emphasises the AI's *process*, as described in (Colton and Wiggins 2012), and what we called the AI's *presence*. Exposing the AI's process is common within computational creativity, using techniques such as *framing* to describe how the AI has made creative decisions. We define presence as "*the impact a computationally creative system has on its environment, and the impact the environment has... in return*". Building a system that has a purpose to exist in the world besides vending content, for example by engaging with its audience or developing its skills in private, improves the audience's perception of that system as creative.

Exhaustive PCG

In (Sturtevant 2013) Sturtevant introduces the notion of *exhaustive procedural content generation*. Reasoning that storage is now no longer the bottleneck it once was for computers, Sturtevant argues that it makes sense to explore exhaustive generation of some types of content rather than trying to search through the space. This not only circumvents the need for search heuristics and operations which transform our position in the search space (which may be hard to define) but also allows for the creation of richer tools that use their exhaustive understanding of the space to visualise the design space in greater detail.

Sturtevant and others have since expanded this idea and applied it to level design for several puzzle games including *Fling!*, *The Witness* and *SnakeBird*. In all cases, constraints are set for the space (such as the level size in *Fling!*) and then all possible permutations of content within that space are generated. This is aided by two properties of the games studied: they are known in advance (meaning specialised solvers can be implemented, and constraints can be selected to focus the design space); and they are typically lightweight (a level is usually represented as a two-dimensional array, and the primary analysis of the level is to find a single solution).

Motivating Puck

Related Work in AGD

Two trends are evident in the current direction and application of automated game design research. One trend is the support of mixed-initiative game design tools, often as a way to lower barriers to game development as a creative form. Recent examples of this include *Germinate*, a casual game creator aimed to support the creation of rhetoric-driven games (Kreminski et al. 2020), and *Mechanic Maker*, a mechanical design tool driven by visual examples (Saini and Guzdial 2020). Some supportive systems are designed for expert or more experienced users – Anhinga is an example of this, where the tool is best used by someone with a high level of domain knowledge, able to take advantage of the tool's insights (Sturtevant et al. 2020). In both cases, the AI aims to solve and support design tasks, keeping a human in the loop at all times. This branch of AGD research has clear value both to everyday and expert creators.

The other trend is towards autonomous, independent AGD systems which work alone, which Puck is an example of. Other examples of systems like this include Ludi (Browne and Maire 2010), which focuses on the design of boardgames but uses similar approaches to videogame-focused AGD systems, and our previous work on ANGELINA (Cook, Colton, and Gow 2017) which used computational evolution applied to several different genres and platforms. While the goals of mixed-initiative AGD are clear (providing AI support to game designers), the purpose of autonomous systems is not as well-explored, and their potential role in a future creative society is not discussed. Often, one stated goal for such systems is the creation of games of a similar quality to human expert designers.

Related Work in HCI & Design

Reviewers noted the similarity between our approach with Puck and the *slow technology* movement, first proposed in (Hallnäs and Redström 2001). Hallnäs and Reström present slow technology as an agenda for design explicitly opposed to a trend towards speed and efficiency in technological development. The movement has expanded and better defined areas of concern in the years since. In a workshop in 2012, Odom et al. identify the key themes of 'designing for slowness', 'designing for [use] across multiple generations' and 'designing for... less consumptive lifestyles' (Odom et al. 2012), all of which relate very closely to our goals for Puck, and the broader social issues we have previously identified in game AI research (Cook 2021).

Slow technology, and the idea of reflective systems, has influenced game AI researchers in the past. Smith, in her exploration of procedural generator design philosophy, poses that 'a reflective design process' might be a good contrasting value to existing trends in commercial procedural generation (Smith 2017). Kreminski and Mateas explore the role of reflection in the context of mixed-initiative systems in (Kreminski and Mateas 2021), and suggest a number of design patterns which, while intended for support tools, have parallels with Puck if we consider the audience to be analogous to a 'user'.

Infinite Rembrandts

Scientific research is affected by the general perception we have of a problem or field, and vice versa (The Royal Society 2018). Outside of AGD research, creative AI today are often portrayed as *Infinite Rembrandt Machines*, by which we mean their ultimate goal is to produce a system that can produce an endless amount of masterpiece-quality work, on demand, usually instantly, and for zero cost. This narrative can be clearly seen in the presentation of DALL-E 2, for instance. This tells us something about how the people who work on these systems think about creativity and wish to frame its role in the world.

We believe that alternative narratives and goals are important, to provide competing visions for what AI can do for society. Puck is designed as a creative AI system, but our goal is not for it to effortlessly create perfect output. Instead, we are interested in building a system that can become a participant in a creative community. In order to achieve this, we had a few guiding principles in mind when designing Puck:

1. Human-Scale Puck is slow. Although we optimise Puck and intend to make it faster at doing certain tasks, our aim is for Puck to remain operating at human-like scales of creativity. This means that working on a full game might take days, weeks or even months or work. If Puck were to become drastically faster at making games, we would simply redesign the system so that it spent more time doing other things (such as playing other games, giving feedback, and so on) rather than making games much faster.

Our goal is to develop a system that works on timeframes comprehensible to people. For Puck to participate in a community of game developers it should work like them, allowing time for feedback and sharing. Similarly, for Puck to develop an audience as a designer, it must give that audience time to engage with and anticipate its work as it develops. Although we think of technology as something getting faster and more efficient all the time, to do so in the case of creative work is to ignore and ultimately do away with certain processes that are at the core of what it means to be creative.

2. Open We make no attempt to hide Puck's artificiality, or obscure access to its data (except where it impacts the system's individuality). For example, when generating text to describe a game or report information to the user, we tag parts of the text based on how we generated it (e.g. grammar templates, live data). The user can reveal these tags to better understand our contribution to Puck's 'voice'. We hope this builds trust between us and the user that we are not pretending Puck is capable of things it is not.

Similarly, Puck's created games and associated data are stored as plain-text JSON files in the user's file system. This allows the user to view Puck's work and even edit files (to the point of breaking Puck, even). Our aim here is to expose as much of the system as possible, partly to help the user feel that the system can be understood, and partly to foster a more homebrew, open relationship with the software. This is also a response to modern software trends towards closed systems, cloud-based computation that is invisible to the user, and inaccessible data and decision-making. We want the system to feel as open, customisable and locally-editable as possible.

3. Individual Although, as we describe below, we currently provide Puck with approximate objective functions with which to score games, our primary aim is not to design 'good' games (although this is still a good outcome). Instead, we are keen for Puck to develop an individual creative practice and grow as a designer. To achieve this, Puck retains information about every playtesting session, game design and user interaction, which provides a unique history for each instance of Puck from the first minutes. We call this approach *eventually exhaustive* design – it adopts Sturtevant et al.'s approach to exhaustive data retention, but we do not expect to ever achieve exhaustion, and instead use partial data to draw conclusions about the design space.

This full history of its actions allows Puck more flexibility to make mistakes (since it can go back and revisit them), contextualise its actions meaningfully by referencing the past, and have reasons to hold subjective but consistent beliefs about games. We are not interested in building a game design oracle which has every answer. Instead, we seek to build a system that can hold bad opinions about games but do interesting things with them – like all good game designers we know do.

The motto of the Glorious Trainwrecks development community, which has been running for over fifteen years, is 'Make Games Constantly Forever'. They explain 'it doesn't matter if you've got talent, so long as you've got *gusto*'(Penner 2007). We have been inspired by many small game-making communities in the creation of Puck, and owe a lot to our own circles of game developer friends who have inspired us, challenged us, and helped us to grow. We want to develop AI with gusto.

System Description

Puck does not have a linear process, but instead moves between different activities or focuses, which we call *moods*. Each mood has a specific focus, such as inventing brand new game designs, and uses a core set of capabilities to achieve this. In this section we describe the structure of Puck's design space, Puck's various moods, how it evaluates games, how it generates new games, and how the user-facing part of the system functions.

Game Structure

Puck is engineered as a flexible platform for AGD research. Evaluation, visualisation and game-playing functionality are entirely modular, meaning they can be switched out with ease. This modularity extends to the type of game being designed. In this subsection we describe the current design space Puck can work in, but we will extend this in the future so that Puck can cover multiple domains at once, opening up new research questions relating to cross-genre work.

Puck currently designs games that are played on a grid of up to 8x8 squares. Each square can contain zero or one game pieces, of which there can be any number of types (although practically we limit it to eight types at most). The board size limitation means we can use a 64-bit integer as a bitmask for pieces on the board, and then use one such integer as a mask for each piece type (which we call a 'layer', internally). Checkers, for example, is represented by three layers: one for white pieces, one for black pieces, and one layer which represents all pieces (every game has this layer).

Game rules are expressed using a modular component system, where rules exist as self-contained chunks of game logic that subscribe to event messages sent out by the game itself. For example, a rule that allows the current player to place a piece onto the board might listen for the `BOARD_TAP` event and then add a new piece to the board at the tapped location. Modules can be customised to trigger off of different events, and some modules have custom parameters (for example, a Match-X win condition can take varying values for X).

The current system has 11 events, including Game Start and End, Turn Start and End, Destroy, Create or Move Piece, and Gain Score. We have provided Puck with a set of 25 game modules, some of which are entirely self-contained (such as a module which ends the turn) and others which do not have much of an effect on their own but can combine with other modules (such as a module which tags any row of 3 or more pieces; other modules can destroy or add score for tagged pieces). We do not provide Puck any prior knowledge of how to use these composite rules. Our hope is also for Puck to eventually be able to create its own rule modules per (Cook 2020).

Moods & Mood Selection

Puck currently has two moods in the release version, and one more experimental mood not yet released:

- **Generate New Ideas (M1)**: Puck generates a new game design, either from scratch or based on an existing game design. It plays the new design once.
- **Test Promising Games (M2)**: Puck tests an existing game which has only been played once, playing it in greater depth with more varied agent configurations.
- **Test for Degenerate Strategies (EM1)**: Puck tests a game with a specialised agent to try and detect a particular kind of degenerate strategy where a small set of actions can be repeated over and over to win.

Deciding which mood to move to next is a key part of the system's personality, and something we are developing slowly as we expand the system's capabilities. Our long-term goals are to settle on a process by which:

- Puck can make intentional, high-level choices about what work to do that affects its creative growth.
- Observers can make sense of these choices and understand them as being purposeful and intentioned.

In order for mood selection to feel meaningful there must be meaningful decisions to make, which requires us to have a wider selection of lower-level behaviours to choose between. This remains an area of future work, given the small number of moods currently available. For now, we have devised a few simple measures that balance Puck's behaviour between creating new games and exploring existing ones:

- If fewer than thirty games have been generated, generate a new game.
- If a game in the top 10%, ordered by score, has not been played more than once, fully evaluate it using mood M2.
- If any game in the top 15% has not been mutated then generate a new game via mutation.
- Otherwise, generate a new game.

We focus on a fairly conservative percentage of the highest-scoring games for two reasons: first, the design space has quite a high proportion of uninteresting games, and although we wish to retain these in Puck's database, we don't want to prioritise their evaluation. Secondly, these rankings are temporary. We intend for Puck's understanding of games to change over time, which means evaluations of games will change and rankings will alter accordingly, as we add new metrics and new systems for Puck to develop its own metrics. In the future Puck may reconsider the quality of a game based on the knowledge it has, which might allow it to rise up the rankings into the top 10-15% and thus be eligible for evaluation further down the line. Thus we can afford to overlook games *now*, as we retain access to them (exhaustively) for later.

Generating Games

New game generation is mostly intuitive (for example, there is a 50% chance a new game will be single- or multi-player, and there are lower and upper bounds set for board size and the number of game modules). Puck's generation also includes several instances of what we call "design space sculpting" – small adjustments to the generation process that alter the generative space and bootstrap the generation of games. Currently, these are:

- Games must include a win condition.
- Games must include at least one interaction rule.

We list these here partly for transparency's sake, but also because we do not consider these to be permanent features of Puck's generation. In the future we intend for Puck to be able to sculpt its own design space through inference (see Future Work), and for the user to be able to specify their own suggested sculpts for Puck to decide on. We can think of these current conditions as training wheels for the system, rather than inherent features.

Mutating from a source game has a 2.5% chance to remove an existing module, add a new module, or change a feature of the game specification (such as the number of players or board size). We check to see if the resulting mutation has already been seen before (since Puck retains everything) and reattempt mutation a fixed number of times until we mutate an unseen game. If this fails to find a fresh game, we simply abort mutation and generate a new game conventionally instead.

Game Evaluation

As part of mood M1, new game designs are evaluated using game agent playouts, a common technique in automated game design (Cook, Colton, and Gow 2017; Browne and Maire 2010). We use open-loop MCTS agents (OLMCTS) which store actions in the search tree rather than game states (Liebana et al. 2015). This makes them flexible enough to handle games both with and without randomness, with little loss of efficiency. New game designs are initially played using mid-level OLMCTS agents, stronger than our 'weak' agent preset, but weaker than our 'strong' agent preset: 2000 iterations, a rollout depth of 20. The dynamic selection of a C value appropriate to an unseen game design is still a point of future work for us – we use a low C for deterministic multiplayer games and a high C for singleplayer games with randomness and scoring.

The game is played for a number of turns proportional to the size of the board. The reason for this is to ensure that as many win conditions as possible can be validated. Common win conditions include covering the board, which may initially start empty. Thus, running the game for at least as many turns as there are squares on the board is important to test if the game can be won. Games are currently played for a number of turns equal to twice the number of squares on the board, to account for other rules which may reverse progress. The primary objective with this initial testing is to rule out obvious deficiencies, such as being unable to influence the game state, or being unable to end the game.

The second stage of evaluation, in mood M2, aims to gather more data to calculate metrics that measure different qualities of the game and its gameplay. For singleplayer games this consists of a playout by: a Random Valid Agent (RVA), which is a modified random player that only makes actions that affect the game in some way[1]; a weak OLMCTS player (500 iterations, 10 rollout depth); and a strong OLMCTS player (4000 iterations, 30 rollout depth). Two-player games are more complex: we perform three mirror matchups of the RVA, Weak and Strong agents, and then two asymmetric matchups: RVA vs Weak, and Weak vs Strong. In both singleplayer and two-player cases, each setup is played five times. The aim of these assessments are to allow a preliminary assessment of the following metrics:

- **First-Player Advantage (2P Only)**: measures the expectation of the first-player winrate against the measured winrate. Mirror matchups should be approximately equal, imbalanced matchups should favour the stronger player.
- **Skill Advantage (2P Variant)**: measures the proportion of games in imbalanced matchups in which the higher-skill player wins.
- **Skill Advantage (1P Variant)**: measures the difference in performance between tiers of agent (RVA, Weak, Strong), and then computes a capped, weighted sum of these ratios. We expect higher-skill players to perform better, but we cap the maximum they can demonstrate as some games have exponential scoring growth.
- **Unique States**: measures the number of unique game states encountered across all playthroughs. This is currently capped at 75% of a theoretical maximum, as it is not a property Puck should always seek to maximise. This is a clear candidate for variation (see Future Work).
- **Progress**: measures the number of playouts in which either score was gained (if scoring is possible) or a player won (if a win condition is present).

User-Facing Application

The underlying AI system, described above, was initially developed as a headless application for experimentation. Once we were satisfied with the basic implementation and structure, we began work on a separate interface that would eventually form the downloadable version of Puck. We omit many minor engineering details here, which sadly are hard to justify including in six-page paper, but the system has been designed to be as flexible and extensible as possible.

Puck's application currently consists of two modes: *Design Mode*, which gives control over to Puck to freely design games according to the mood logic described above, and *User Mode*, where the user can explore Puck's work and play with the results. Design mode is largely self-explanatory, as the AI acts autonomously. The AI's actions are always visualised, with game playouts clearly visible, alongside information about the current game design.

In User Mode the user is presented with an index of every game Puck has generated and tested, sorted initially by Puck's own internal scoring. We have implemented a few quality of life features for the user, including the ability to 'star' games, which cause them to appear first in the game list, regardless of their score according to Puck. The user can also rename games, and add notes to refer to later. Most importantly, the user can click on any game to be taken to an interactive interface for playing the game. They can add AI agents to take over any player role, allowing them to play against other people, the AI, or to stage AI vs. AI matches.

While not part of the traditional user-facing application, our own research version of Puck has additional features designed specifically for streaming on Twitch. We have integrated Puck with Twitch chat, allowing it to play games against Twitch viewers, running polls to decide where the Twitch player should play next. In addition to being a fun feature for viewers, we hope that this may become a form of dynamic and quick playtesting for our instance of Puck, as it allows immediate evaluation of a game by a large group of people.

Game Descriptions Puck utilises game descriptions throughout both User and Design modes, and this feature will only become more important as the system becomes better able to create and release complete games. Describing a game from its design outline is not straightforward, and is considered a challenging research problem even for games which are well-formed. As is often the case in AGD, the requirement to be able to describe even badly-formed games greatly complicates this task.

In the current version of Puck we compose rule descriptions by delegating to each individual rule module, requesting a description of the module itself, then composing individual descriptions into a whole. This mostly produces legible descriptions, however some filtering is required as certain modules can never execute in some game designs, while others execute invisibly to the player (such as modules which mark the board for other modules to respond to). Additionally, such composite rule descriptions treat each module with equal importance, whereas succinct descriptions of games tend to give high-level overviews of the important rules, or may even imply a particular strategy.

Figure 1 shows an example game description generated by Puck for *Triple Flip*, a game we describe in depth in the

[1]For example, swiping randomly in *Bejewelled* will not count as a game action unless it forms three-in-a-row.

```
Tapping the board changes the tapped
piece, adds a piece to the board, drops
a piece onto the board, and ends the
turn.
If any players has [sic] a line of 5 or
more, they win.
```

Figure 1: A generated description of *Triple Flip*.

next section. The rules are primarily contained in a single line, which describes all the consequences of tapping the board. However the description misses or confuses some details. For example, tapping an existing piece and tapping an empty space have two different effects, but the phrasing of the rule implies they both happen at once. 'Changing the tapped piece' is also dependent on the number of piece types active in the game, which is not included in the game description. These problems sound like they have simple solutions, but fixing them for this game would cause problems when describing a different kind of game. We hope to improve game descriptions as an area of future development.

Evaluating Puck

Evaluating automated game design systems is something researchers in the field have often struggled with. Past attempts include internal algorithmic evaluations, for example to show that an evolutionary process is improving in fitness (Cook, Colton, and Gow 2017). Other systems were evaluated through user studies, such as the *Data Agent* system where users were asked to evaluate the system's ability to create a coherent and enjoyable mystery scenario. The Gemini project has recently framed evaluation in terms of the interpretable meanings a designed game conveys. Comparisons between different systems are rare, besides qualitative discussions in Related Work sections, because AGD systems are typically very distinct from one another, using different game engines, technology, and design philosophy.

For a field that already struggles with evaluation, Puck's unusual aims make it even more awkward to evaluate. For example, showing the best or average fitness of the population is meaningless, since no individual is ever removed and elites are always retained. Measuring the speed with which good solutions can be found is also unhelpful, since this is explicitly not our aim. We hope to perform user studies in the near future to better understand the audience's relationship with the system but there is little to compare against, so in isolation these studies are hard to identify as good or bad.

This paper is primarily a systems description paper, and a philosophy paper of sorts: a statement of intent for us about the project and its future trajectory. However, by way of a supporting argument for our work, we include below some examples of output from our personal version of the system, as a way of showcasing some of its output and its unusual qualities, both good and bad.

Example Games

Antitrust designed by Puck in 2021, for two players on a 5x5 board. Players take it in turns to place pieces onto the board. At the start of a turn any line of four or more pieces of the same colour is removed from the board. When the board is full, the player with the most pieces wins. This game was used in an earlier study (Johansen and Cook 2021).

This game was discovered during Puck's development, identified as one of its three top-scoring games at the time of running the study mentioned above. It should be noted that the other two higher-scoring games were in our opinion not as good[2], and thus this should be considered a curated/cherrypicked output. There is a much longer debate to be had about the validity of such outputs.

At first glance, the first player has a distinct advantage since they place a piece on the board first, and thus maintain a piece advantage as the board fills up, which is crucial to winning the game. However, the numerous ways to make four-in-a-row means it is easy to force a player into making a line, which is then removed, reducing their board presence and extending the game. The player closest to winning is therefore always at a disadvantage, which gives the game a pleasant back and forth. The game suffers from a long game duration for players of even average skill.

Triple Flip designed by Puck in 2022, for two players on a 6x6 board. Players take it in turns to either place a piece of their colour on the board, or flip an existing piece. Pieces have three states: *green*, owned by the first player, *orange*, owned by the second player, or *pink*, owned by neither player. Flipping the piece advances it to the next state (flipping a pink piece makes it green). The first player to get four-in-a-row of their colour wins.

At first glance this game sounds similar to Noughts and Crosses. However, each player has a different relationship with the flipping mechanic. For Player 1, their own pieces are dangerous, as they can be flipped into ownership of Player 2. This means that if Player 1 places a piece carelessly near a line of Player 1's pieces, it might be flipped into four-in-a-row and lose the game. On the other hand, Player 2 has no such fear of their own pieces. Instead, the neutral pink pieces are a concern as they can be flipped by Player 1. At the same time, pink pieces provide no utility to Player 2, meaning Player 1 can leave pink pieces on the board as blockers to constrain Player 2's options, who cannot use them without first giving them to Player 1. Thus the game creates a certain kind of asymmetric play out of what is, on the face of it, a perfectly symmetric ruleset. This game is not quite as balanced as Antitrust – Player 1 has a slight advantage. But it is an interesting example of the kind of game Puck currently is capable of identifying, and well outside of the expected space of results we had in mind for Puck's initial version.

Exhibition and Release

During pre-release testing, we asked testers to comment on how they found the software to use. The testers are a self-selecting group who follow game AI researchers on social media, however their feedback was encouraging as they engaged with the software exactly as we hoped. One tester

[2]Puck's rating metrics remain a work in progress, and at the time lacked some of the metrics it uses today.

noted that they found 'the beginnings of a good game' after leaving Puck to run for a couple of hours (as we recommended). Users were confused by the game descriptions provided by Puck, which is a priority for us to improve next.

Puck was accepted for inclusion in the WASD games expo in London in April 2022. Over the course of three days Puck was installed on the show floor, with attendees able to come and watch Puck work on games, and play games that Puck had designed. This was the longest test of the system to date. It ran without problems for the duration of the event, designing hundreds of new games, and with attendees favouriting several games in the list, offering additional useful data.

Puck was also released to download on itch.io[3] where, at the time of writing, it has been downloaded 260 times. We were also fortunate to receive some press coverage (via *Rock, Paper, Shotgun*) despite not advertising the release. Puck does not track or monitor user activity at all, but we do plan to notify users of a survey in the future to better gauge how users are finding the software, to compare the feedback from testers with a broader group of users. We hope to grow this audience into a small community of enthusiasts who share their creations and give feedback on the project as a whole. Hopefully this can combine with an audience on Twitch following our version of Puck's online design work.

Future Work

Design Inference

In general, hard-and-fast "design rules" for creative work do not exist. Yet creators do develop informal guidelines and preferred practices, based on their feelings, their audience's feedback, and the experiences of the work they are making. We believe that Puck can bootstrap its own design work through an inference process that provides it with higher-level beliefs about the design space it inhabits.

We have experimented by building a simple inference system based on its own design experience. Puck can survey its database of games and note 1-, 2- and 3-tuples of modules that score consistently high or low when appearing together in the same game, or that consistently appear in games which violate design constraints (such as being winnable). From this it can suggest possible guidelines, such as "games with module X must have module Y to be playable".

These inferences are currently not strong enough to be built into Puck, and remain experimental. We would like Puck to be able to assign confidence values to each inferred guideline (how likely it believes the evidence) and strength (how consistently the guideline should apply). Ideally Puck would also be able to explicitly test theories by generating many games in sequence that violate or abide by the guideline. We feel this would not only help instances of Puck individualise themselves and demonstrate growth, but also more effectively isolate areas of the design space to explore.

Knowledge Exchange

Unlike other creative forms such as writing, visual art or music, there is no common representation format for videogames, nor is there a standardised way to run and interact with them. This poses a problem for automated game design researchers that is somewhat unique among AI and creativity research, as there is no straightforward way for AI systems to engage with, and therefore learn from, existing examples of work in the same domain.

Our short-term aims in this area are twofold: first, to add specific interface options for exporting and importing games designed by Puck, so that people can share output with other instances of Puck. We envision this as also embedding the originating Puck's experiences, as a form of argumentation that explains why this particular instance of Puck believes the game to be good (or bad). This dovetails with our other work aiming to individualise each instance of Puck.

We also aim to build tools within Puck for users to design games with, using the same toolset Puck has access to. By using Puck's language, users can both recreate famous games as well as invent their own games in a format Puck can understand. This offers two interesting new research challenges – enabling Puck to learn from one-shot examples of design from trusted sources, and enabling Puck to provide feedback on a user's game design, suggesting changes or making observations about elements it thinks work well.

Conclusions

In this paper we introduce *Puck*, a new automated game design system that is designed to embed in people's local creative communities. Puck combines principles from automated game design and computational creativity with the notion of exhaustive content generation, to create a slow design system that intentionally takes a scalable, low-stakes approach to creative work. Puck never discards its work, and thus opens up new opportunities to engage its audience with its growth, creating opportunities to build on its past mistakes and learn from its successes. Puck has already produced several interesting games, and is proving to be a fruitful platform for further automated game design research.

Acknowledgements

This work was supported by the Royal Academy of Engineering under the Research Fellowship scheme. Thanks to Younès Rabii, Florence Smith Nicholls, Mads Johansen and Sara Cardinale for their contributions to Puck, and to the excellent reviewers.

References

Barros, G. A. B.; Green, M. C.; Liapis, A.; and Togelius, J. 2019. Who Killed Albert Einstein? From Open Data to Murder Mystery Games. *IEEE Transactions on Games*, 11(1): 79–89.

Browne, C.; and Maire, F. 2010. Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1): 1–16.

Charity, M.; Khalifa, A.; and Togelius, J. 2020. Baba is Y'all: Collaborative Mixed-Initiative Level Design. In *IEEE Conference on Games*.

[3]gamesbypuck.itch.io/

Colton, S.; and Wiggins, G. A. 2012. Computational Creativity: The Final Frontier? In *Proceedings of the 20th European Conference on Artificial Intelligence*.

Cook, M. 2020. Software Engineering For Automated Game Design. In *IEEE Conference on Games*.

Cook, M. 2021. The Social Responsibility of Game AI. In *Proceedings of the IEEE Conference on Games (CoG)*. IEEE.

Cook, M.; and Colton, S. 2018. Redesigning Computationally Creative Systems For Continuous Creation. In *Proceedings of the Ninth International Conference on Computational Creativity, ICCC*.

Cook, M.; Colton, S.; and Gow, J. 2017. The ANGELINA Videogame Design System - Part I. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2): 192–203.

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*.

Guzdial, M.; Liao, N.; Chen, J.; Chen, S.; Shah, S.; Shah, V.; Reno, J.; Smith, G.; and Riedl, M. O. 2019. Friend, Collaborator, Student, Manager: How Design of an AI-Driven Game Level Editor Affects Creators. In Brewster, S. A.; Fitzpatrick, G.; Cox, A. L.; and Kostakos, V., eds., *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, 624. ACM.

Hallnäs, L.; and Redström, J. 2001. Slow Technology Designing for Reflection. *Personal Ubiquitous Computing*, 5(3): 201212.

Johansen, M.; and Cook, M. 2021. Challenges in Generating Juice Effects for Automatically Designed Games. In *Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Khalifa, A.; Green, M. C.; Barros, G. A. B.; and Togelius, J. 2019. Intentional computational level design. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*. ACM.

Kreminski, M.; Dickinson, M.; Osborn, J.; Summerville, A.; Mateas, M.; and Wardrip-Fruin, N. 2020. Germinate: A Mixed-Initiative Casual Creator for Rhetorical Games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Kreminski, M.; and Mateas, M. 2021. Reflective Creators. In *Proceedings of the Twelfth International Conference on Computational Creativity*.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2013. Sentient World: Human-Based Procedural Cartography - An Experiment in Interactive Sketching and Iterative Refining. In *Proceedings of the Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design*.

Liebana, D. P.; Dieskau, J.; Hunermund, M.; Mostaghim, S.; and Lucas, S. M. 2015. Open Loop Search for General Video Game Playing. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*. ACM.

Nelson, M. J.; and Mateas, M. 2007. Towards Automated Game Design. In *Proceedings of the Congress of the Italian Association for Artificial Intelligence*.

Odom, W.; Banks, R.; Durrant, A.; Kirk, D.; and Pierce, J. 2012. Slow Technology: Critical Reflection and Future Directions. In *Proceedings of the Designing Interactive Systems Conference*. Association for Computing Machinery.

Pell, B. 1992. METAGAME in symmetric chess-like games. Technical Report UCAM-CL-TR-277, University of Cambridge, Computer Laboratory.

Penner, J. 2007. Glorious Trainwrecks. glorioustrainwrecks.com. Accessed 2022-08-01.

Saini, V.; and Guzdial, M. 2020. A Demonstration of Mechanic Maker: An AI for Mechanics Co-Creation. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Smith, A. M.; 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*.

Smith, G. 2017. What do we value in procedural content generation? In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Smith, G.; Whitehead, J.; and Mateas, M. 2010. Tanagra: An Intelligent Level Design Assistant for 2D Platformers. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Sturtevant, N. 2013. An Argument for Large-Scale Breadth-First Search for Game Design and Content Generation via a Case Study of Fling! In *Proceedings of the 2nd Workshop on Artificial Intelligence in the Game Design Process, at AI-IDE*. AAAI.

Sturtevant, N. R.; Decroocq, N.; Tripodi, A.; Yang, C.; and Guzdial, M. 2020. A Demonstration of Anhinga: A Mixed-Initiative EPCG Tool for Snakebird. In *Demonstrations @ AIIDE*.

Sturtevant, N. R.; and Ota, M. J. 2018. Exhaustive and Semi-Exhaustive Procedural Content Generation. In *Proceedings of the Fourteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI.

Summerville, A.; Martens, C.; Harmon, S.; Mateas, M.; Osborn, J. C.; Wardrip-Fruin, N.; and Jhala, A. 2019. From Mechanics to Meaning. *IEEE Transactions on Games*, 11(1): 69–78.

The Royal Society. 2018. Portrayals and perceptions of AI and why they matter. tinyurl.com/rs-portrayals. Accessed 2022-08-01.

Togelius, J.; and Schmidhuber, J. 2008. An experiment in automatic game design. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games*.

Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. Game-O-Matic: Generating Videogames That Represent Ideas. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*.