The TekaTekiTech System for Automated Minigame Generation: First Steps

Edward J. Powley, James Clewett and Simon Colton
The MetaMakers Institute
Academy for Innovation and Research
Falmouth University, UK
{edward.powley, james.clewett, simon.colton}@falmouth.ac.uk

Abstract

Automated generation of entire games is a challenging task in computational creativity. This paper presents some initial explorations with a system for generating small puzzle games with simple mechanics but complex emergent gameplay. Our system is designed to enable collaboration between human designers and creative computer agents, and to explore the notion of writing code which itself writes code. Our aim is to produce games in which deciphering the mechanics of the game is itself a meta-mechanic, and where some puzzles may be unsolvable (but not obviously so). We draw an analogy between unsolvable puzzles and unprovable mathematical conjectures: we are intrigued to see whether the same force that compels mathematicians to persevere with a possibly futile proof extends to the wider population of gamers.

Introduction

This paper introduces *TekaTekiTech*[1], a system for the generation of puzzle minigames. Our work on this system has only just begun; this paper is a motivation and discussion of our recent, present and future work.

Our aim is to create a design space which is both sufficiently flexible to admit a wide variety of games, and in which the density of interesting games is high enough to allow them to be found easily. To keep the density of games high enough, we chose to limit the space to puzzle-type games, of the type which exhibit fairly simple mechanics to test the player's reactions, dexterity and/or problem solving abilities. We did not set out with specific existing games in mind for the game designer to produce; instead our aim was to make available a range of interesting mechanics and see how the designer combined them. The results so far, based on random generation and tweaking by hand, are quite unique puzzles which are difficult to classify in terms of existing commercial games; some are reminiscent of the minigames featured in the Mario Party and WarioWare series of games by Nintendo. However we are confident that games in the style of more traditional puzzle games such as Tetris, Minesweeper and Sokoban, as well as some arcade games such as Pacman and Frogger, are also present in the space.

Our wider aim in this work is to investigate how players respond to games which are *discovered* rather than *created*. If a human game designer produces a game which is impossible to win, or only winnable through superhuman dexterity, then the player might reasonably say that the designer has made a mistake. However if such a game is discovered by an automatic game generator, we conjecture that the player will be much more tolerant of the unwinnable game, and indeed that coming up with an explanation for why it is unwinnable might be a pleasure in itself. Ensuring solvability is often a concern when automatically generating game content (Smith, Butler, and Popović 2013); instead, we are interested in the region of design space where even the designer does not know whether the content is solvable. Determining solvability then becomes an objective for the player rather than the designer.

Our work here fits into the context of *automated whole game generation*. Due to the complexity of constructing a multi-faceted object such as a game, only a few projects have studied how to automate the entire process of generating a game. An important project in this area is the work of Cook et al. on the ANGELINA system, which designs entire games in a variety of genres using a co-operative co-evolutionary programming approach (Cook, Colton, and Gow 2015a; 2015b). The ANGELINA system has been particularly influential in games communities, for instance, as the first automated system to enter a game jam (Cook and Colton 2014). Other important projects which address game generation include early work of Nelson and Mateas on generating games in the style of Nintendo's WarioWare series (Nelson and Mateas 2007). The Game-O-Matic project[2] enables people to automatically generate short, simple games by defining relationships, as described in (Treanor et al. 2012), and an approach to puzzle generation with simulated physics models was investigated in (Shaker et al. 2013). Further aspects of AI for game design are discussed in (Riedl and Zook 2013). Taking a higher level view, Liapis, Yannakakis, and Togelius (2014) describe the mutual benefits for Computational Creativity research with application to video game design.

This paper is structured as follows. First we describe how games are specified in the TekaTekiTech system, and

[1] "Teka-teki" is Javanese for "puzzle".

[2] `http://cartoonist.soe.ucsc.edu`

how a specification translates to a playable game. Next we give some illustrative examples of games generated by the system, in two categories: first, abstract games designed with minimal human input; second, games where a human designer intervened to create games with meaning. Next we frame the TekaTekiTech system in the wider context of Computational Creativity, discussing our goals for the project in both scientific and philosophical terms. Finally we outline future work on the project.

The TekaTekiTech system

The system has two GUI components: an interface for designing games, and an interface for playing them. See Figure 1. A game in the system is defined by a set of *elements* such as: input event handlers; collision detectors and handlers; object spawners, movers and killers; game progress trackers and displayers; and theme managers for graphical and audio assets. Many of the element types take parameters. Figure 1 (a) shows an example of a game element being edited in the designer GUI. The element is of type `NumPiecesDraggedProgressMechanism`, and takes two integer parameters: `failurePenalty` and `maxNumPiecesDragged`. This element implements a game mechanic where at most `maxNumPiecesDragged` can be touched in one game: if more than this number of pieces are touched, the player loses `failurePenalty` points.

TekaTekiTech currently defines a large library of element types, each of which is a Java class; the system can easily be extended by adding more element types. The elements are building blocks of middling complexity, more complex than e.g. arithmetic operators but less complex than e.g. movement controllers for game characters. Excluding boilerplate code, the behaviour of a single element is defined by 5–20 lines of code on average. In this way TekaTekiTech has superficial similarities with PyVGDL (Schaul 2014), but there are also many differences in design. PyVGDL is intended as a human-readable game specification language for AI competitions, and includes some high-level constructs to this end; in TekaTekiTech the emphasis is much more on automated generation and mutation of games, which would only be made more difficult by adding complexity to the specification system.

All element types implement a `respondToTick` method, which is called once per game update for every currently active element in sequence. There is also a shared pool of events, and elements may add or remove events from the pool. For example one element may add information on a collision event to the pool, and another element may then handle the event. Certain elements can also modify the list of active game elements whilst the game is in progress.

The game definition itself is simply a list of element types and parameters. The game specification is a flat list of elements. (The hierarchy shown in Figure 1 (a) reflects the hierarchy of Java packages defining the element types, and is provided for user convenience; the underlying data structure being edited is flat.) This flat structure is in contrast to some other game description languages where the specification is a hierarchy of elements (Schaul 2014) or a program in a declarative programming language (Love et al. 2006). This makes it easy to generate TekaTekiTech games at random or by simple evolutionary approaches, for instance. The current version of the editor GUI allows elements to be added and deleted, and their parameters to be adjusted by sliders. Once the game has been edited, it can be played with a single menu command or shortcut key, allowing rapid iteration of editing and testing. This simple interface is enough to allow simple editing of random games, and even design of games from scratch. It is easy to envisage how more advanced editing tools could be provided to allow for local search, better parameter visualisations etc. This is a subject for future work.

At the beginning of the game, elements are instantiated according to the specification. On each frame of the game, all elements are updated. Gameplay emerges from complex interactions between the components. It is important to make a distinction between game *elements* and on-screen game *objects*: elements manage various aspects of gameplay, which may include the creation, update and deletion of on-screen objects, but elements are not themselves objects in the game world.

We now describe some of the games currently implemented in the system. These games were designed by a (very simple) creative agent in collaboration with a human designer: the games were initially generated randomly, before being curated, tweaked and themed by hand. Only minor tweaks were made to the games to make them more playable, for example adjusting the numbers or movement speeds of pieces to tune the difficulty (some generated games were far too difficult, others were far too easy). Importantly, no changes were made that substantially changed the core mechanics of the games.

Abstract games

The first phase of development focussed on abstract games, where the on-screen objects are simple circles and rectangles. All of the abstract games use the same basic on-screen objects: a rectangular board split into cells, and a number of circular pieces. In *abstract game 6* (Figure 2 (a)), the board rotates counterclockwise whilst the pieces orbit clockwise. When a piece is clicked, all pieces which overlap the board stick to the board. The aim of the game is to make six pieces stick to the board. However if a piece is moved by dragging, all unstuck pieces shrink to nothing; also, if the player touches more than one piece, the game is lost. One strategy for the game is to observe that the pieces sometimes overlap the corners of the board as they orbit; when this happens, click and release any piece. As long as the same piece is clicked every time, this leads to a win.

In *abstract game 7* (Figure 2 (b)), the board rotates and the pieces move in straight lines in random directions. When pieces collide, they change directions at random. When a piece is clicked, it stops moving. The aim is to get 10 pieces onto the board, however the game ends in a loss if any piece moves off the screen. Despite its simplicity, this game is very challenging. One strategy operates in two phases: first, rapidly click many pieces to create a barrier around the edge

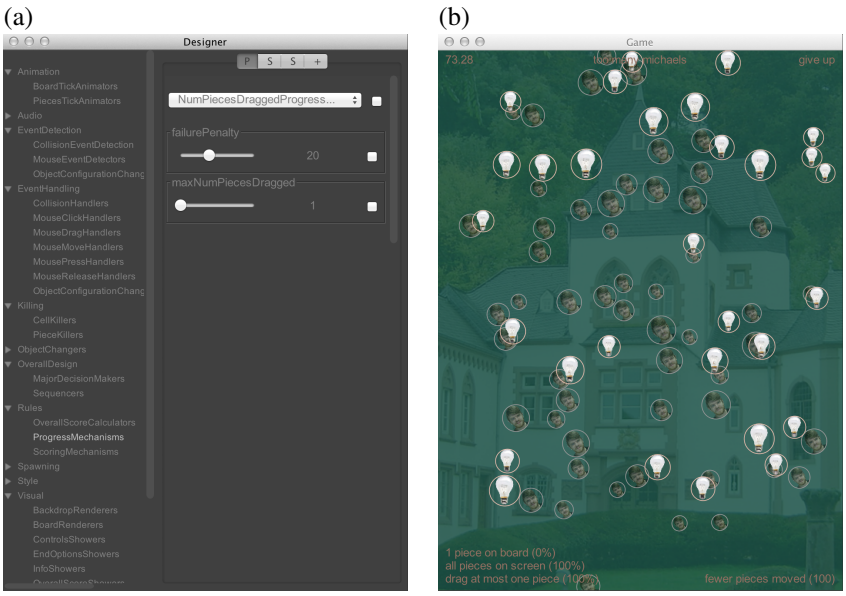

Figure 1: Screenshots of the TekaTekiTech system, working with the game Too Many Michaels. (a) The game designer GUI. (b) The gameplay GUI.

of the screen; second, engineer collisions between pieces until enough of them overlap the board. It took many hours of play to discover this strategy; other strategies exist, but are even more difficult to find.

In TekaTekiTech, most games use a random number generator for initial placement of pieces. However the seed for the generator is fixed, so the placement is the same every time the game is played. This allows the player to learn the placements of pieces over multiple plays of the game, and develop a strategy accordingly. It may be the case that some games are only solvable under certain random seeds. For example, abstract game 6 is winnable because the initial placements ensure that every piece passes over a corner of the board at some point in the game. A different seed may not have this property, and so would render the game unsolvable. It would be possible for an observant player to notice this and hence prove that the game cannot be won.

Games with meaning

The second phase of development was to move beyond abstract games and towards games with meaning. "Meaning" in this context is achieved by adding graphical assets for the on-screen objects and background in a way that, when combined with the mechanics of the game, tells some kind of story.

Too Many Michaels features a number of static lightbulb sprites (representing ideas) and a number of Michael sprites moving in circles centred in the middle of the screen. The player can move a lightbulb by dragging, and use this lightbulb to push the other bulbs around. The aim is to ensure that, when the dragged lightbulb is released, one and only one lightbulb sprite is overlapping a Michael sprite. In other words, the aim is to ensure that Michael gets only one idea, rather than a larger overwhelming number of them. See Figure 1 (b).

In *Let It Snow*, blue circles orbit the middle of the screen whilst snowflakes fall in the background. The score increases whenever a snowflake appears, and the game ends once a target score is reached. See Figure 2 (c). Dragging a snowflake causes all snowflakes on the screen to shrink and disappear. Hence the best strategy for this game is simply to "let it snow", i.e. to watch the game play out without interacting. Of course this is not spelled out to the player, so realising this becomes a game objective in itself.

Discussion

Our aims for the TekaTekiTech systems are numerous. Firstly, we are interested in automated programming as a methodology for the building of creative systems. Approaches where software writes software, or re-writes its own code, have a number of advantages, not least with respect to projecting autonomy onto software from a public understanding perspective. That is, it is fair criticism to say that software is only an extension of the person/team who wrote it, therefore any interesting behaviour which may resemble creativity that the software might exhibit can be attributed to the author. This point of view is fairly commonplace because software is programmed directly by a person, and there is a general lack of understanding that AI techniques such as machine learning and evolutionary programming generate and alter code indirectly, hence not explicitly instructed by a programmer. By talking about automated software writing projects, where software writes and re-writes its own code or that of other programs, we believe that it would be more difficult for people to argue that the original programmer — who is now one step removed —

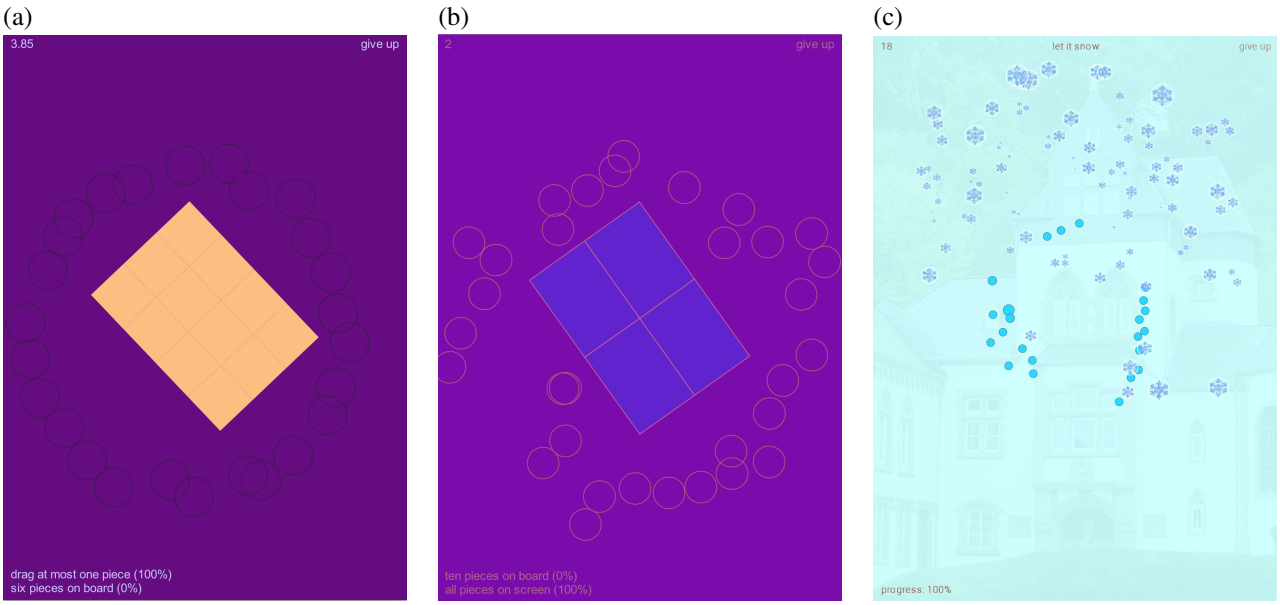

Figure 2: Screenshots from (a) abstract game 6; (b) abstract game 7; (c) Let It Snow.

should gain credit for the behaviours of the software. With the project here, we are interested in adding to the usual set of automated programming techniques such as machine learning and evolutionary approaches, as previously mentioned.

Another aim for the TekaTekiTech program is to use the output games and the fact that they were automatically generated as a vehicle with which to explore an analogy between puzzle/problem solving in an entertainment setting and in a pure mathematics research setting. When playing a commercially produced casual puzzle game, there is a certain inevitability in the solving of each puzzle presented. Even if a game level/puzzle is very difficult, players tend to know that it is worth persevering, as it would be largely unthinkable that an *unsolvable* puzzle had entered into the game. Notable exceptions to this are games with a randomised generative element, such as card games like FreeCell Solitaire (Heineman 2009) and numerical games like the CountDown Numbers Game (Colton 2014), where the random shuffle of cards or generation of numbers can lead to an unsolvable puzzle. These are very much exceptions to the rule in commercial puzzle gaming, and it would likely be seen as a bug in a game if an unsolvable level was shipped with the game. The same is true of certain types of mathematical problems in educational and research settings. Imagine, for instance, a maths problem in an exam which was not solvable — there would be outrage amongst the students that they were forced to waste valuable time on such a question. The same is true for mathematical tutorial questions, but it is emphatically not true about problems arising in pure mathematics research.

In such a setting, *open conjectures* come with the caveat that what is thought to be a truth about a mathematical object such as a number, graph or group, could actually be completely untrue. For instance, the rock of Fermat's Last Theorem (Singh 1997) against which thousands of mathematicians before Andrew Wiles crashed could definitely have been false, and a counterexample could have been found with more powerful calculations. Indeed, there have been several cases where a famous centuries-old conjecture was unexpectedly disproved by the discovery of a counterexample[3]. When working on such open problems, it is commonplace to switch between finding a proof and finding a counterexample, and such doubt adds to the difficulty, but also the enjoyment of research-level pure mathematics, i.e., perseverance in the light of potential failure is richly rewarded with high satisfaction in success and bitterly digested when an open conjecture becomes a false theorem. Of course, with open conjectures, the rich rewards are not just attributed to personal satisfaction related to coming through adversity. In addition, open problems are by definition not known to be true or false by any other human being. Hence, by being the first to prove or disprove an open conjecture, a certain notoriety and satisfaction is gained at being the first to know some new — and in certain cases important — fact about the world.

In general, puzzle solving in a ludic, entertainment setting has much in common with mathematics problems in an educational setting, but is missing the two elements vital in the enjoyment of research mathematics, namely the tension between not knowing whether something is true/false (in the gaming analogy: a level being solvable/unsolvable) and being the first person in the world to know something (in the gaming analogy: being the first to complete a level). As many of the mini-games produced by TekaTekiTech are unsolvable, but neither the software nor any person knows

[3] http://math.stackexchange.com/questions/514

this in advance of playing, and as the game space is so huge that it's unlikely that a randomly sampled game would be given to two people simultaneously, we have the opportunity to see whether the joys of pure mathematics research transfer over to that of mini-game playing. Within this context, and of particular importance for Computational Creativity, we would like to test the hypothesis that human-free generation of such puzzles adds to the enjoyment. There is a long-standing debate in the philosophy between humanists and Platonists, whereby the former point out that mathematical results come from human minds and are therefore inventions like the vacuum cleaner, but with the latter indicating that certain mathematical concepts and results such as prime numbers are clearly universal, and are merely discovered by people, like Columbus discovered the American continent. Software generating puzzles seems somehow more akin to the Platonic view, and so a person solving such puzzles might gain the joy of discovering something new. On the other hand, if a person wrote a mini-game, we might argue that it is their responsibility to determine whether it is solvable or not, and not engage with it. This is a subtle point that we hope to explore with further philosophical thought and experimentally.

One difference between mathematical problems and TekaTekiTech games is that the former can be solved with pure mental effort, whilst the latter also require some degree of physical skill to achieve a solution. In some puzzles it may be easy to see what must be done, but difficult to make the precise inputs required. The constraints are often soft. Such puzzles are less like pure mathematics and more like research in the physical sciences. Adapting the analogy, the difference between playing a designed game and a discovered game is akin to the difference between reproducing a set experiment as a student and devising an entirely new experiment as a researcher. Both require skilful use of apparatus as well as mental skill, but only the latter carries the uncertainty of trying something that may be impossible and the potential satisfaction of succeeding where no-one has before.

A third aim for this work is to explore further the notion of discovering game mechanics as a meta-level game mechanic. Each new game presents a new set of rules, so that the player cannot know how to play a game — part of the enjoyment comes from figuring out the rules of the game. This presents a unique challenge, which can be fun and frustrating in equal measure. Discovery of game mechanics as a game mechanic in itself is a feature of some indie games such as Antichamber and The Binding of Isaac. However where it often manifests simply as a deliberate lack of tutorials and hand-holding in a game with many other mechanics, we intend to push TekaTekiTech as a meta-game in which the main meta-mechanic is the discovery of game mechanics. This raises the question of how mechanic discovery can be encouraged beyond simply withholding information from players, e.g. whether certain mechanics are more or less discoverable than others.

Future work

Good game design is a feedback loop: all human game designers know the vital importance of prototyping, playtesting and refinement. To truly automate the game design process, it is necessary to automate playtesting. Playtesting in TekaTekiTech can be considered an instance of general videogame playing, as the playing agent cannot be programmed in advance with knowledge of the game. It is also desirable to be able to playtest and iterate rapidly, so playing techniques such as deep reinforcement learning (Mnih et al. 2015), which require extensive training to achieve strong play, are not suitable. State-of-the-art techniques for general game playing without offline learning tend to be based on Monte Carlo Tree Search (MCTS) (Browne et al. 2012). In videogames, where a decision is required once per game tick (as opposed to turn-based games where decisions are allowed to take several seconds), MCTS is often augmented with macro-actions (Powley, Whitehouse, and Cowling 2012) and/or open loop search (Perez et al. 2015). We will use these and other techniques to create an agent that can play TekaTekiTech games competently. The aim is not necessarily to create a world-champion level player, merely one that is smart enough to recognise when a game is too easy. The most interesting games may turn out to be the ones which the AI agent can almost, but not quite, solve; and informing the player of this may add an enjoyable element of human-versus-machine competition.

Let It Snow, described above, is an example of a "game" which on the surface looks rather uninteresting, but becomes interesting with suitable interpretation and the added gameplay dimension of mechanic discovery. In the context of the curated collection of games we have assembled so far, it is interesting because it is unusual. However in the search space this type of game is relatively common. Indeed there are abundant examples of even less playable games, for example those where the player instantly wins or loses. It is necessary, and fairly easy, to detect and filter such games.

Despite the rich history of metaprogramming in computer science, automated code generation remains relatively uncharted territory in computational creativity. TekaTekiTech is engineered from the start with automated programming in mind. While approaches such as that in the Mechanic Miner sub-system (Cook et al. 2013) uses direct code manipulation for very similar aims to ours, namely game mechanic generation, the ANGELINA system which this forms part of has not been developed specifically with automated programming in mind. By splitting games into 43 distinct elements which can be instantiated in numerous ways, parameterised and chained together for combinatorial complexity, we have reduced the amount of operational code in the elements to on average a small number of lines of code (around 5–20). This means that the code is now eminently amenable to automated programming, and our first attempt at this will be a *cut, paste, tweak* and *compile* approach. That is, TekaTekiTech will take its existing code for a particular instantiation of a particular game element, copy and paste it as a new instantiation and then perform rudimentary alterations, attempting compilation and trying out the new element in existing games, testing for any differences in game-

play that arise. Because the operational code is so small, the dependencies of alterations can be somewhat predicted in advance, and the code alteration can be more carefully controlled. If, for instance, a piece of copied/pasted/tweaked code was hundreds of lines long, there are more likely to be butterfly effects, where small changes in the code make for large changes in the gameplay. In future work, we plan to test the efficacy of numerous automated code generation techniques.

We have anecdotal evidence that the system generates compelling games. We will use a framework such as curation analysis (Colton and Wiggins 2012) to assess the effectiveness of the system. Whilst TekaTekiTech is a promising vehicle for exploring the above mentioned scientific and philosophical issues, our ultimate aim is to bring it to market as a commercial game for mobile devices. Puzzle games are popular on such devices, so a product which promises a practically infinite variety of such games should be extremely compelling. We hope that players' discovery and possibly co-designing of new games will promote a strong sense of creative ownership, and we will give players a way to share games on social media, for example to challenge their friends to solve a particularly difficult game. Games in which players are left to discover the mechanics often breed the strongest communities, with some players spending as much time discussing and cataloguing the game on forums and wikis as they spend playing it. We hope and believe that TekaTekiTech has the potential to inspire this kind of dedication amongst players.

Acknowledgments

This work has been funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities). We are very grateful to the participants of the recent Dagstuhl Symposium on AI and Games, for their very useful feedback on the system. We also thank the anonymous reviewers for their insightful comments on this paper.

References

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Colton, S., and Wiggins, G. A. 2012. Computational creativity: The final frontier? In *Proceedings of the European Conference on Artificial Intelligence*, 21–26.

Colton, S. 2014. Countdown numbers game: Solved, analysed, extended. In *Proceedings of the AISB symposium on AI and Games*.

Cook, M., and Colton, S. 2014. Ludus ex machina: Building a 3d game designer that competes alongside humans. In *Proceedings of the Fifth International Conference on Computational Creativity*.

Cook, M.; Colton, S.; Raad, A.; and Gow, J. 2013. Mechanic miner: Reflection-driven game mechanic discovery and level design. In *Proceedings of the EvoGames Workshop*.

Cook, M.; Colton, S.; and Gow, J. 2015a. The ANGELINA videogame design system, part I. *IEEE Transactions on Computational Intelligence and AI in Games (to appear)*.

Cook, M.; Colton, S.; and Gow, J. 2015b. The ANGELINA videogame design system, part II. *IEEE Transactions on Computational Intelligence and AI in Games (to appear)*.

Heineman, G. T. 2009. Algorithm to solve freecell solitaire games. `http://broadcast.oreilly.com/2009/01/january-column-graph-algorithm.html`.

Liapis, A.; Yannakakis, G.; and Togelius, J. 2014. Computational game creativity. In *Proceedings of the Fifth International Conference on Computational Creativity*.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2006. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford University.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Nelson, M., and Mateas, M. 2007. Towards automated game design. In *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence on AI and Human-Oriented Computing*.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games (to appear)*.

Powley, E. J.; Whitehouse, D.; and Cowling, P. I. 2012. Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem. In *Proceedings of IEEE Conference on Computational Intelligence in Games*, 234–241.

Riedl, M., and Zook, A. 2013. AI as game producer. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*.

Schaul, T. 2014. An Extensible Description Language for Video Games. *IEEE Transactions on Computational Intelligence and AI in Games* 6(4):325–331.

Shaker, M.; Sarhan, M.; Al Naameh, O.; Shaker, N.; and Togelius, J. 2013. Automatic generation and analysis of physics-based puzzle games. *IEEE Transactions on Computational Intelligence and AI in Games*.

Singh, S. 1997. *Fermat's Last Theorem*. Fourth Estate.

Smith, A. M.; Butler, E.; and Popović, Z. 2013. Quantifying over Play: Constraining Undesirable Solutions in Puzzle Design. In *Proceedings of the 8th International Conference on the Foundations of Digital Games (FDG 2013)*, 221–228.

Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. The micro-rhetorics of Game-O-Matic. In *Proceedings of the Procedural Content Generation Workshop*.