Automated Tweaking of Levels for Casual Creation of Mobile Games

Edward J. Powley, Swen Gaudl, Simon Colton, Mark J. Nelson, Rob Saunders and Michael Cook
The MetaMakers Institute, Games Academy, Falmouth University, UK

metamakers.falmouth.ac.uk

Abstract

Casual creator software lowers the technical barriers to creative expression. Although casual creation of visual art, music, text and game levels is well established, few casual creators allow users to create entire games: despite many tools that aim to make the process easier, development of a game from start to finish still requires no small amount of technical ability. We are developing an iOS app called Gamika which seeks to change this, mainly through the use of AI and computational creativity techniques to remove some of the technical and creative burden from the user. In this paper we describe an initial step towards this: a Gamika component that takes a level designed by the user, and tweaks its parameters to improve its playability. The AI techniques used are straightforward: rule-based automated playtesting, random search, and decision trees learning. While there is room for improvement, as a proof of concept for this kind of mixed-initiative creation, the system already shows great promise.

Introduction

Liapis, Yannakakis, and Togelius (2014) describe automated game design as a *"killer application"* for Computational Creativity research, as it combines many disparate creative disciplines into a cohesive whole. Casual games, i.e. games with relatively shallow learning curves and relatively simple game mechanics which demand only a relatively short time investment for an engaging return, have broadened the popularity of gaming as a pastime and have contributed to a shift in the overall demographic of those who play games regularly (Juul 2009). Similarly, *casual creator* software (Compton and Mateas 2015), which emphasises the enjoyment of creation over the necessity of completing a task, has broadened the popularity of digital creation and begun to blur the boundary between consumers and producers of creative artefacts. However, the creative act of game design is not yet fully supported by casual creation on mobile devices.

A desire to further democratise game design has led us to the building of the Gamika iOS application. We believe that this is the first tool which enables entirely new casual games (containing multiple levels with novel aesthetics, game mechanics and player interactions) to be designed entirely on a mobile phone without programming. Gamika empowers designers to experiment with new game mechanics and untried gameplay. This naturally increases the requirement for extensive tuning and playtesting of game levels, which can be tedious. To combat this, we have endeavoured to make tuning and testing of levels an entertaining experience, and to automate parts of the process where appropriate. We have sought to make this automation enjoyable, e.g. by allowing the user to watch the tuning process as it happens, rather than merely presenting them with a progress bar.

The work here represents a proof of concept for on-device casual game creation, including the invention of new game mechanics, without programming. Our contribution is the whole pipeline and the AI functionality embedded in Gamika which supports casual co-creation, rather than a focus on studying and optimising one particular aspect.

In this paper we discuss our approach to provide support for level design through automated parameter adjustment. We begin by surveying existing mobile apps which allow users to create games or game content. We then describe the Gamika system, and present as a running example a particular game that has been designed within the app by one of the present authors. The design of new levels for this game presented us with a challenge, as levels often require fine-tuning of parameters to yield a satisfying play experience. We present the results of our initial experimentation into automated tweaking of parameters, using random search, learned classifiers (decision trees) and a simple automated playtester. We also discuss the nature of the search space, and the difficulties we have encountered in trying to apply more intelligent search methods. We conclude by looking forwards to our goals and plans for the Gamika project.

Game creation on mobile devices

The term *casual creator* (Compton and Mateas 2015) describes a piece of software which allows users to quickly and easily create artefacts such as musical compositions, filtered digital photos, abstract artworks, stories, and poems. Some games include features such as character creators or level editors which could be classed as casual creators. However, there is a lack of such software that gives aspiring game designers fine-grained control over all aspects of a game, particularly the underlying rules and mechanics of the game.

There are many software packages for novice game designers to experiment and learn the craft, such as Scratch (scratch.mit.edu), Stencyl (stencyl.com) and AgentCubes (agentcubesonline.com). They can then move on to

professional-grade tools such as GameMaker: Studio (yoyogames.com), Unity (unity3d.com) or Unreal (unrealengine.com). Many of these allow the designer to deploy their game to mobile devices, however all require that the development takes place on a desktop or laptop computer. Also all require programming (using visual programming environments and/or text-based programming languages) for all but the most trivial games. This significantly raises the barrier to entry for non-technical users. Practically, all casual games on the market today are made by professionals or skilled hobbyists, and are created using these tools.

Apps which enable creation of games or game assets on the mobile device itself fall into three main categories:

- Apps which require programming skills. Many, such as Scratch Jr (scratchjr.org) and HopScotch (gethopscotch.com), are primarily tools for teaching programming skills to children, with game creation being a vehicle rather than an end in itself. These apps give the user a wide range of creative expression, but assume a degree of technical proficiency (or a desire to learn technical skills).
- Apps which enable only skinning existing game templates, i.e. customisation of a game's visual appearance and audio, but little or no ability to change the gameplay. Examples here include Coda Game (codarica.com) and Playr (playr.us) which offer some degree of creative freedom, but only with respect to a game's aesthetic qualities.
- Apps which enable the authoring of levels for an existing game. Examples include Createrria 2 (incuvo.com) and Sketch Nation (sketchnation.com). Minecraft: Pocket Edition (minecraft.net) can also be included in this category, with advanced features such as redstone circuitry blurring the lines between level design and programming. Such apps empower creative expression, but limited to the characters, rules and game worlds provided.

Gamika is our attempt to fill a gap in the market: a casual creator app which empowers users to create entire games on their mobile device, in particular enabling the invention of entirely new game mechanics. One of our ultimate aims is for Gamika to be able to automatically generate entire casual games, so that users can delegate as much (or as little) of the creative responsibility as they like. Several authors (Lim and Harrell 2014; Khalifi and Fayek 2015; Nielsen et al. 2015) have studied automatic generation of game levels using general-purpose game description languages such as VGDL and PuzzleScript. Nelson and Mateas (2008) formally modularise recombinable game mechanics. These allow users to define novel game variants and get automated feedback on properties such as playability. The Game-O-Matic project (Treanor et al. 2012) enables users to generate short games by just defining relationships. Zook and Riedl (2014) use planning techniques to generate new game mechanics. ANGELINA (Cook, Colton, and Gow 2016) is perhaps the closest to a true whole-game generation system that exists currently, but there is still a long way to go before automated systems reach a level where they can be taken seriously as game designers in their own right.

Gamika

Gamika is an iOS application developed in the Swift programming language and using Apple's SpriteKit 2D game development library. SpriteKit includes a physics engine based on Box2D, which forms the basis of game object interactions in Gamika. A *game* in Gamika is a list of *levels*, each of which comprises: 284 numerical parameter settings which control several aspects of gameplay; an optional vector graphic to be used as a controller; and an optional text string explaining the rules of the game to the player. By analysing classic arcade games and by attempting to design novel games within the app, we have grown parameter set organically until satisfied that they admit game levels that are sufficiently diverse, interesting and engaging.

A game level features *objects* of several classes, with each object corresponding to a rigid-body in the physics simulation. The parameters that define a game level are split into several categories:

- **Image**. The level can optionally use an abstract art image generated by the ELVIRA evolutionary art system (Colton, Cook, and Raad 2011) as an in-game object or as a background image.
- **Look**. The background image can be changed, as can the size, shape, colour and sprite images for the game objects.
- **Lights**. Spotlight effects can optionally be added to the game, and the appearance of game objects under lighting (e.g. normal mapping) can be adjusted.
- **Spawning**. Objects of each class can be spawned at a given rate, from a given position or range of positions, optionally with an upper limit for the number of objects of each class allowed on screen at one time.
- **Movement**. The physical properties of the objects, such as restitution, mass and linear damping, can be changed. Force-fields can be added, to attract or repel objects with respect to a particular point, or apply a force in a given direction. Joints such as pins, springs and sliders can be added to objects.
- **Collisions**. For each pair of object class, several actions can occur when objects collide. For example they may bounce, pass through, stick, be destroyed, or change class. When objects stick together they form clusters; it is possible to set a maximum cluster size so that larger clusters are destroyed. Walls can be added to the edges of the screen.
- **Control**. The player's taps and swipes can cause game objects to move, rotate, change direction, stick in place, spawn, be destroyed, or be attracted or repelled.
- **Counters**. The designer can define counters (e.g. score, health, lives) by which the player's progress is measured. Counters can be incremented or decremented by in-game events such as collisions, spawning or destruction of objects, objects leaving the screen, clusters being formed, and objects entering scoring regions.
- **Endings**. The game can optionally be set to end after a certain time limit, or when a certain counter value is reached. What constitutes a win or loss, and high scores are recorded can also be specified.

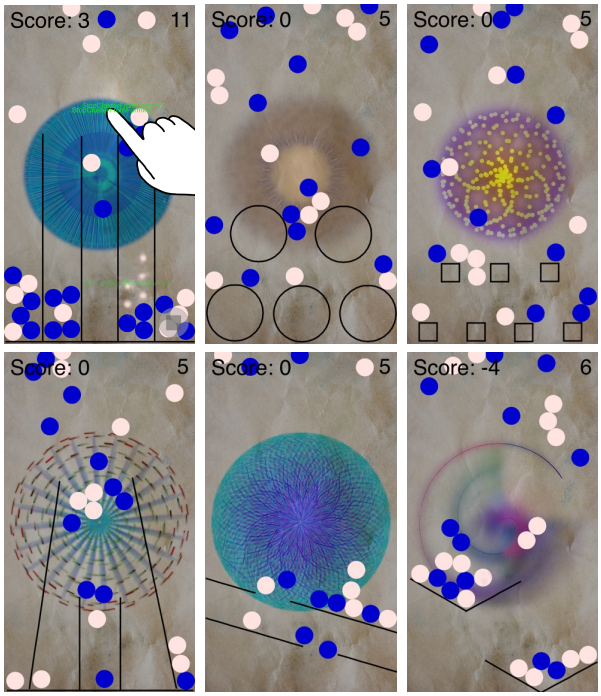

Figure 1: Screenshots of Let It Snow levels 1 to 6.

Gamika maps a space of casual games to a space of numerical parameters. Other attempts to map out a space of games for manual or automatic exploration include VGDL (Schaul 2013) and PuzzleScript (puzzlescript.net). These systems map games to hierarchical code structures, whereas Gamika uses a space of numerical vectors. Similar representations have also been used, for example, by (LeBaron, Mitchell, and Ventura 2015) for tile properties in puzzle games. In Gamika, any sequence of 284 numbers in the correct ranges is a syntactically valid (but not necessarily playable) game level, making the representation amenable to simple local search and evolutionary methods.

Gamika is also set apart by its use of simulated physics. VGDL and PuzzleScript define explicit movement rules for in-game objects: "when the player presses the A key, move the player sprite one space to the left, unless there is a wall in the way". Gamika specifies only the physical properties of the objects and the environment: "when the player taps the screen, add a force that attracts the controller object towards the tap position". This level of indirection changes how the space is navigated: designing a game in Gamika is less like an exercise in software engineering, and more like a random walk whose course can be changed completely by a serendipitous discovery.

Let It Snow

This paper focusses on one particular game, titled *Let It Snow*, designed within Gamika by one of the present authors (Colton). Figure 1 shows screenshots of six levels. Each level features a drawn controller formed of black lines/shapes. The abstract art image in the background is for decoration only. White and blue balls are spawned at the top of the screen, at a rate of 3 each per second, and fall towards the bottom, bouncing off the drawn controllers as they settle. If there are 20 balls of a single colour on screen, no new balls of that colour are spawned until the number drops below 20. (The values of 3 balls spawned per second and maximum 20 balls on screen are parameters which may vary from level to level and may differ for blue and white balls within the same level.) Balls bounce off balls of the other colour and off the controller. Balls of the same colour stick together. When a cluster forms of four or more balls of the same colour, all balls in the cluster explode. The player gains one point for each white ball that explodes, and loses one point for each blue that explodes. The controller can be "jiggled" slightly by dragging on the screen, but the player's main interaction with the game is to tap blue balls. Tapping a blue ball causes it to explode; tapping a white ball does nothing. Exploding a blue ball loses one point, so should be done sparingly, however it is sometimes necessary in order to clear the way for a cluster of whites to spawn and thus give a net gain. The game ends when the player reaches a predetermined score threshold (either 50 or 100 points, depending on the level). There is no time limit, but the player's position on the high score table is determined by how quickly they reach the target score, and achieving lower than a certain time allows the user to advance to the next level.

Let It Snow is a challenging, fast-moving game, rather more dependent on quick thinking and reactions than many casual colour-matching games. Novice players tend to fail to get control of the game and scores can plummet. Sometimes the arrangement of the balls means that no new balls are spawned but no clusters form, hence the game comes to a halt. In these *quiescent* moments the game takes on a puzzle element, as the player has time to ponder their next move. The designer found that a good playing strategy is to concentrate on getting these quiescent moments, then carefully keeping control of the situation through selective destroying of blue balls. For example in level 1, a skilful player will get blues locked in singletons, pairs and triplets at the bottom of the five columns. Then only white balls spawn and land on those exposed above the locked blues in such a way that they continuously form groups of four and thus allow more whites to spawn. At this stage the game has the look of snowing (hence the game's name), and players can sit back and watch the score increase rapidly. However, the player may occasionally need to step in if the whites fall in an unlucky arrangement and fail to form a cluster, and an expert player can often be more proactive to try and bring their time down further. An expert player can generally reach 100 points on level 1 within 60 to 90 seconds, with novices usually taking three minutes or more.

An automated player for Let It Snow

Gamika provides a configurable automated playtester to aid designers in testing and fine-tuning game levels. Given that Gamika allows a wide variety of games to be created, it initially seems attractive to playtest them using a general-purpose approach such as Monte Carlo Tree Search (MCTS) (Browne et al. 2012). MCTS has been shown to perform well in the General Video Game AI Competi-

tions (Perez et al. 2015) and for physics-based games (Perez et al. 2013). Game tree search (though not MCTS) is used as a playtester by the Mechanic Miner component of ANGELINA (Cook et al. 2013).

However, when designing new levels for an existing game, the generality of MCTS can be its downfall: we probably want to reject levels where the winning strategy deviates too far from that for the rest of the game, but it is difficult to prevent MCTS from automatically adjusting its strategy to the level at hand. The measure of fitness for a new level is not whether it is playable in general, but whether it is playable with respect to some predefined strategy. In addition, one of our design goals is for the playtester's decision making process to be transparent to the user, so that designing the strategies used and watching them in action becomes part of the enjoyment of using the app. For these reasons, we opted for a simple rule-based player whose rules are parameterised and exposed through the user interface.

The player "ticks" 15 times per second, and on each tick analyses the current state of the game. Certain patterns of balls trigger certain actions. The player taps a blue ball that is about to form a cluster of 4 or more, or a blue ball that is blocking a cluster of 4 or more whites from forming. If the game has become quiescent (with no ball having a non-zero velocity), it taps a blue ball at random. If a period of time has passed without any whites being able to spawn, it jiggles the controller in a small random direction. These rules were informed by the designer's examination of his own strategy. We have found this playtester to play at a level competitive with intermediate to expert human players, albeit not always in a human-like way. For example the automated player taps the blue ball at the last possible moment to prevent a cluster from forming, whereas a human player would anticipate the situation further in advance and tap the blue ball earlier.

Search-assisted level tweaking

In Let It Snow, the main component of a level is the drawn controller. However many controller designs fail to yield a playable level, especially if they are significantly different from the vertical column design of level 1. It is often necessary to tune the parameters of the level to improve its playability, and this (in a cycle of tweaking and playtesting) can often be the most time-consuming part of level design. Indeed, this precise situation arose when the designer of Let It Snow came to extend the game past the initial level.

Our ultimate aim is to perform the tweaking automatically: the designer draws a controller, and then the app takes care of tweaking the level. The *modus operandus* we anticipate for multi-level game design is as follows: the user designs the core mechanics of the game, and in tandem designs an initial level (or perhaps more than one). Based on this initial level, the user designs an automated player by choosing from a number of preset tactics. The app performs some initial analysis on the game. The user then designs further levels, perhaps by drawing a controller or positioning objects. For each new level, the app performs a search, guided by its initial analysis, to refine the game parameters. Ultimately, the user is the curator for the refined levels: they may be presented with several alternative refinements to choose from, or they may choose to fine-tune a generated level by hand. This process should run entirely on the user's mobile device, either while the user waits, or overnight while the device would normally be idle and connected to its charger.

"Fail-fast" random search

A human designer will not waste much time playtesting an obviously bad level: often only a few seconds are needed to see that the current parameters are no good. We seek to emulate this fail-fast approach to playtesting in Gamika's search mechanism. To this end, we define a test set, based on a single automated playtest, that indicate a level is "bad":

1. The score reaches -30 or less at any point;
2. The level is won in less than 20 seconds;
3. The time reaches 80 seconds without a win;
4. The playthrough featured fewer than two quiescent moments (defined as a continuous stretch of time where no ball has a non-zero velocity);
5. The player tapped more than 3.5 times per second on average; or
6. The average distance between consecutive taps was more than 200 pixels.

These tests can be edited by the designer within the app.

For Let It Snow, the designer chose a set of eight parameters to adjust: ball size, fall speed, spawning rate and number allowed on screen, each for both blue and white balls. These parameters have the largest effect on gameplay without deviating too far from the core game mechanics. The parameters are allowed to vary within $\pm 30\%$ of their initial values.

As a baseline, we implemented an uninformed random search. Each *trial* begins by choosing values for the eight parameters from a uniform random distribution. The autoplaytester then plays the game a maximum of three times. The games are played at 4 times normal speed and visualised on screen. If at any stage the level fails one of the tests listed above, the trial ends immediately and a new trial begins. If all three playtests end without failure, the search ends and the level is presented to the user.

Decision trees

As discussed below, uninformed random search is slow. As a first step towards a more informed search, we trained classifiers to recognise those games that are likely to fail the first playthrough (due to failing one of the six tests above). This "two-tiered" approach, of statically filtering candidate levels before dynamically playtesting, is similar to that of (Williams-King et al. 2012). To train the classifiers, for each of of the six levels, we ran the uninformed random search, recorded the parameter settings for each trial, and whether the trial succeeded or failed the first playtest. Testing whether we could train a classifier to cut down search for a set of fine-tuned parameters for level 2, we split the data into two sets. The first data set comprised 249 instances for success in tuning level 2, and we sampled 249 failure instances at random. The second data set comprised instances for success in tuning any of levels 1, 3, 4, 5 or 6 (i.e., with

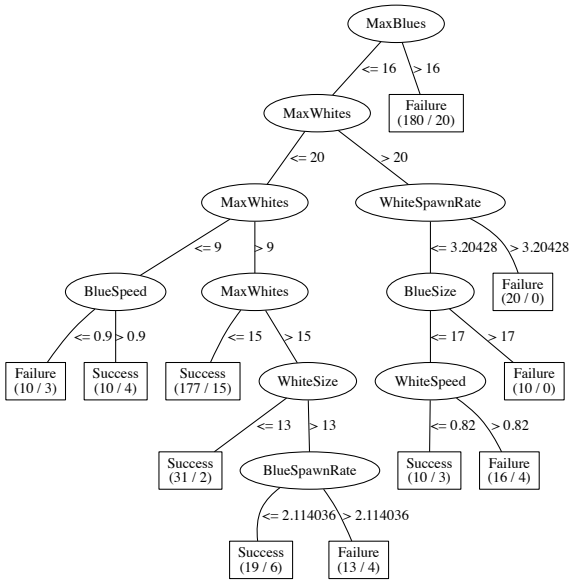

Figure 2: Decision tree trained on Let It Snow level 2. A leaf node labelled (a/b) corresponds to a correctly classified instances and b incorrectly classified instances.

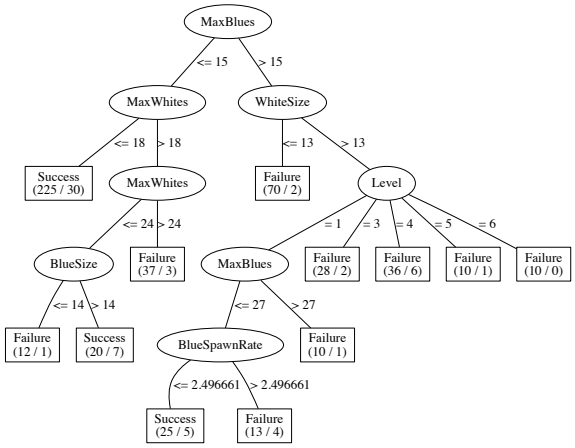

Figure 3: Decision tree trained on Let It Snow levels 1, 3, 4, 5 and 6. See caption for Figure 2.

no knowledge of level 2), and we sampled 249 successes and 249 failures for this data set. These were used to generate two decision trees using the C4.5 algorithm (Quinlan 1993), as implemented by the J48 classifier in Weka version 3.8.0 (Hall et al. 2009).

The generated trees are shown in Figures 2 and 3. The classification accuracies, according to 10-fold cross-validation, are 82.46% and 83.06% respectively. There are some similarities between the trees, both emphasise the importance of the parameters for maximum number of balls on-screen, and impose similar (though not identical) constraints on them. The level number was included in the training data for Figure 3, but has relatively little influence on the decision tree, with only one node testing it (in that case, testing whether the level number is 1).

Rather than taking the traditional rejection sampling approach, we use these decision trees "in reverse" to generate level instances, as follows: First, choose a "success" leaf node in the tree using roulette wheel sampling, with the probability of choosing a leaf node proportional to its number of correctly classified instances (the first number in the leaf nodes in Figures 2 and 3). Then traverse the tree from this leaf node to the root, recording the constraints in the edges along the way. These constraints give a narrowed set of ranges for the parameters; sample uniformly within these ranges. The sampled instance is guaranteed to be classified as a success by the decision tree.

Results

We compared three random search variants: one uninformed, one using the decision tree trained on Let It Snow level 2 (Figure 2) and one using the decision tree trained on the other five levels (Figure 3). In each case the search was seeking to adjust the parameters for Let It Snow level 2. We ran 18 instances of each search variant for a maximum of 24 hours. The instances were run in parallel on a desktop PC, but it is also possible to run a single instance on a mobile device. Each instance ended once it had found a level that passed three playtests as described above.

Table 1 shows the wall-clock time and number of trials taken for the search instances to succeed. For uninformed search, most instances took several hours to complete, and two of the 18 failed to complete within 24 hours. In contrast, for the decision tree trained on level 2, one-third of the instances completed in less than an hour, and only one took longer than 9 hours. The decision tree trained on the other levels was significantly faster than uninformed search but slower than the level-specific search.

Table 2 shows the number and proportion of trials which fail on the first, second and third playtest, and those which succeed. For the uninformed search, the vast majority of trials end in rejection after the first play, with only a handful making it through to the third play and either succeeding or failing there. The searches informed by the decision trees still reject a large proportion of games on the first play, but significantly fewer than the uninformed search.

Figure 4 (a) shows the distribution of playtest durations (measured using in-game seconds) for the three searches. The searches using decision trees have proportionally more trials reaching 80 seconds, and proportionally fewer rejected in less than 20 seconds. Thus, the mean playtest duration is shorter for the uninformed search, thus, it can complete more playtests in less time, however this is not necessarily a benefit as those quick tests are generally failures.

Figure 4 (b) shows the distribution of trial failure reasons in the three searches. Compared to uninformed search, the searches using decision trees reject far fewer levels based on the score reaching -30 or the game lasting less than 20 seconds. This suggests that the decision trees are particularly good at rejecting levels which would fail for this reason. In

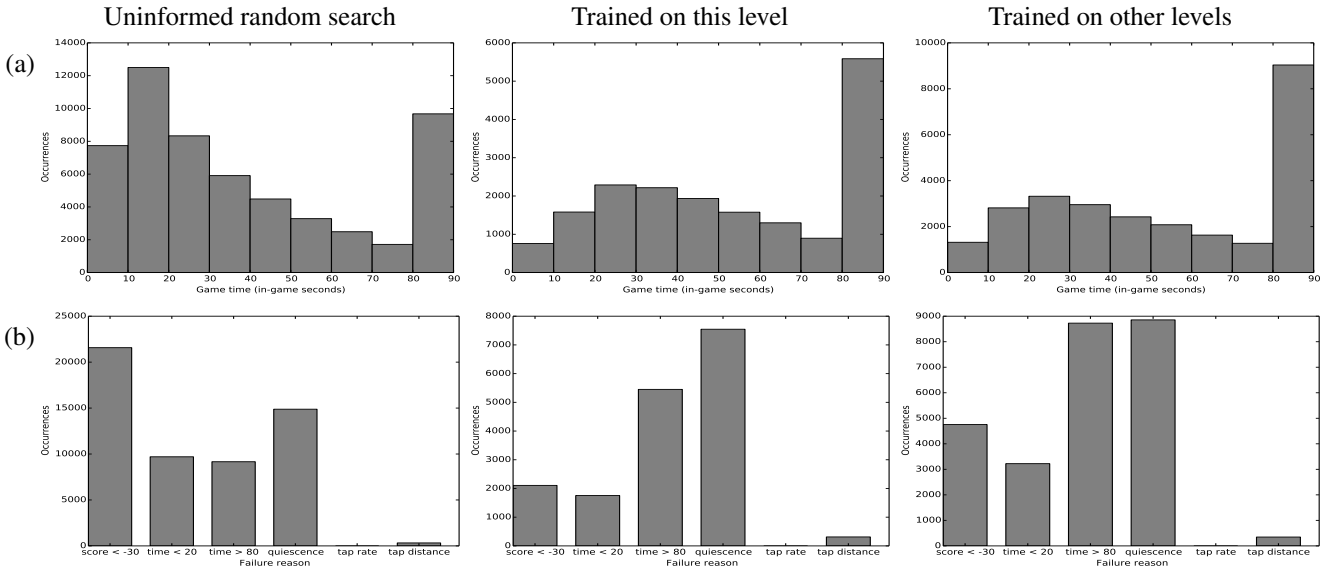

Figure 4: Distribution of (a) play times and (b) trial failure reasons for the three random search variants.

Uninformed random search		Decision tree trained on this level		Decision tree trained on other levels	
Time	Trials	Time	Trials	Time	Trials
0:36:39	243	00:02:24	13	00:08:59	42
1:19:58	478	00:11:29	46	00:47:03	209
1:23:50	489	00:16:19	70	00:52:53	231
4:31:31	1615	00:24:09	121	01:40:39	452
5:09:52	1884	00:24:26	116	01:46:36	458
7:43:46	2786	00:30:45	143	02:39:09	715
8:12:19	2977	01:06:32	272	02:41:26	718
9:26:04	3481	01:31:03	399	03:30:34	935
10:23:26	3816	02:35:13	693	03:42:38	963
10:36:10	3777	02:36:01	690	04:52:19	1272
10:40:37	3884	03:01:42	783	06:22:28	1680
12:08:55	4407	03:54:08	1049	07:16:06	1890
13:23:55	4895	04:55:34	1306	08:11:59	2184
15:05:55	5604	06:22:43	1681	08:45:30	2296
20:36:11	7467	07:04:56	1856	12:01:31	3124
20:38:31	7845	08:22:12	2207	12:02:42	3143
—	—	08:58:33	2362	12:29:54	3309
—	—	16:28:38	4330	12:32:13	3239

Table 1: Wall-clock times (in hh:mm:ss) and number of trials for 18 instances of each random search variant to find a game level that passes all three playtests within 24 hours.

Outcome	Number of trials	Proportion of all trials	Proportion for this test
Uninformed random search			
Fail after 1 play	55225	99.242%	99.242%
Fail after 2 plays	366	0.658%	86.730%
Fail after 3 plays	40	0.072%	71.429%
Success	16	0.029%	28.571%
Decision tree trained on this level			
Fail after 1 play	16331	95.019%	95.019%
Fail after 2 plays	762	4.434%	89.019%
Fail after 3 plays	76	0.442%	80.851%
Success	18	0.105%	19.149%
Decision tree trained on other levels			
Fail after 1 play	25072	96.713%	96.713%
Fail after 2 plays	768	2.963%	90.141%
Fail after 3 plays	66	0.255%	78.571%
Success	18	0.069%	21.429%

Table 2: Outcomes of trials for each random search variant.

all cases, hardly any games are rejected on the basis of tap distance, and none at all on tap rate. This suggests that these tests, which the designer felt were important measures of the difficulty of a level, were in fact not so important. In real-world usage, such information can influence the designer's choice of tests for future uses of the search functionality.

When trained on a specific level, the search finds a suitable level in a median time of 2 hours 30 minutes, however, simply collecting enough data to train the classifier can take longer than this. The median time for a classifier trained on the game but not on the specific level is around 4 hours. When designing many levels for the same game, or a new level for an existing game where the classifier has already been trained, this is likely to be the best tradeoff.

To assess the results of the search methods, the designer performed a *curation analysis* (Colton and Wiggins 2012) on the generated games. For each search variant, he played the first five generated levels in the order in which they were produced (as listed in Table 1). Each was played 10 times, after which the designer decided whether or not the level was good enough to be added to the canonical set of Let It Snow levels, recording the reasons for his decision. For the uninformed search, the second level was the first to be deemed good enough, with three of the five passing the test. For the search using the decision tree trained on level 2, the second

level was deemed good enough and four of five passed. For the search trained on the other levels, the first was deemed good enough and three of five passed. The purpose of this analysis is not to compare the three search variants against each other, as the sample size is too small to draw any meaningful conclusions. However the analysis does show that all three variants produce a reasonable proportion of playable levels, so the user can reasonably expect to obtain a satisfactory result after two or three attempts.

Of the levels that the designer rejected, the main failure was that the level was too easy to get into a snowing state, and too passive thereafter. Of the games rated as very good by the designer, some had the feel of level 1 (which the designer had tuned by hand to be enjoyable), whereas others placed more emphasis on the game's slow-moving puzzle elements. The levels generated by the informed search were noticeably distinct from each other, with no noticeable narrowing in the space of games produced by the informed searches compared to the uninformed search.

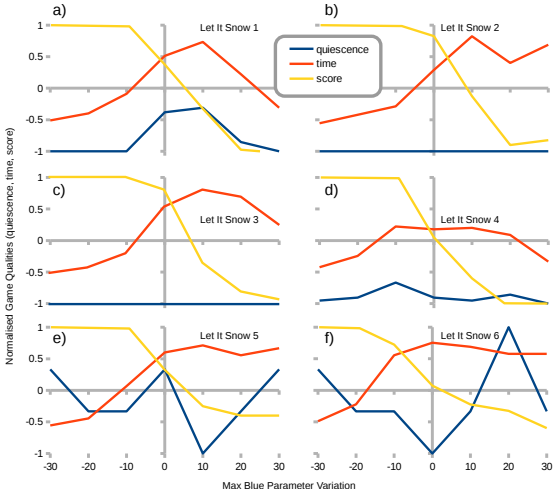

Figure 5: Analysis of the maximum number of blue balls parameter plotted for all six Let It Snow levels independently.

Understanding and traversing the game space

Random search is a baseline, so it should be possible to do better. Playtesting time is dominating the computational overhead of almost any search method, so there is scope to use more sophisticated techniques than random search whilst staying within the confines of the mobile device.

Many games in Gamika have a stochastic element. In Let It Snow, balls are always spawned at the top of the screen but their x coordinate is chosen at random. Furthermore, the physics simulation involving dozens of objects bouncing off each other exhibits chaotic phenomena such as sensitive dependence on initial conditions. This makes any fitness measure based on playtesting highly noisy: it is not uncommon for our automated player's time to complete a given level to vary by 20 seconds or even more. This causes problems when trying to apply search methods such as hillclimbing: either the search is led astray by noise in the fitness landscape, or the need to perform several trials and take an average negates the speed benefit.

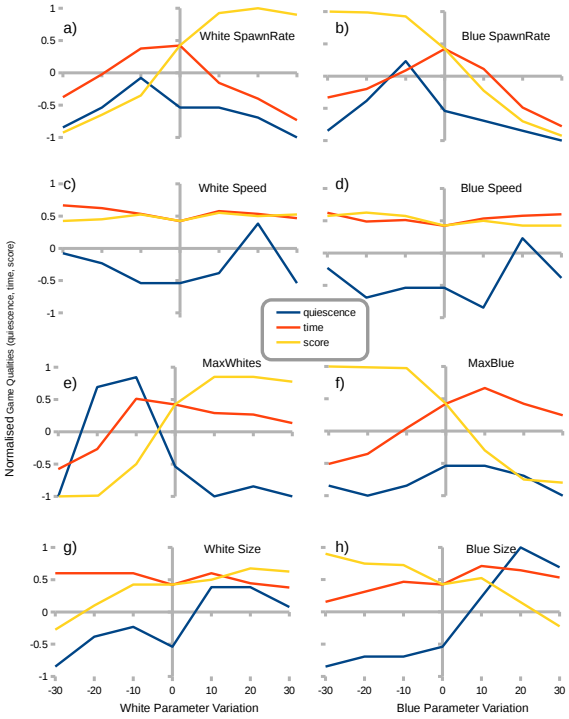

Figure 6: Analysis of all eight parameters.Each graphs shows a parameter sweep averaged over all six levels.

To better understand the effect of the eight chosen parameters on the playability of game levels, we performed a parameter sweep. Each parameter in turn was varied from -30% to $+30\%$ of its initial value in steps of 10%, with the other seven parameters fixed at their original value. For each parameter setting, we ran 10 games with the automated playtester and recorded the final score, the game duration, and the number of quiescent moments.

Figure 5 shows the results for each of the six levels for the "maximum number of blues" parameter. Some commonalities are apparent, for example increasing the number of blues results in a roughly sigmoidal decrease in score. However there are also noticeable differences, particularly in quiescence. A similar story is presented when examining the individual levels' graphs for the other seven parameters (not presented here). In contrast to our initial assumption that all parameters are somewhat level independent, our results suggest that the effect of some parameters on playability is indeed level dependent. Figure 6 shows results averaged across all six levels for each of the eight parameters. Graphs such as these could be displayed in Gamika to allow the designer to visualise trends in parameter changes. Indeed they have already proven useful to the designer of Let It Snow, confirming some of his intuitions regarding the parameter space and forcing him to reconsider others. A full parameter sweep takes around 2 hours to compute on a mobile device, however this could be parallelised as a cloud service to provide faster feedback.

Conclusion

Our aim with the Gamika project is to provide a tool with which non-technical users can create novel, interesting and engaging casual games, beyond mere user-generated content or visual customisation, entirely on their smartphone or tablet. Game design is challenging, both technically and creatively, so we believe that the key to achieving this aim is to use computational creativity to support the user: providing guidance, automation, and ultimately co-creation. This paper describes an approach using rule-base automated play, decision tree learning and statistical analysis to support the designer in creating and adjusting new games and levels.

As a proof of concept, we believe the current version of Gamika shows promise. However there are several areas in which it can be improved:

- Further expanding the space of designable games, for example allowing configurable non-player characters.
- Improving the user interface, to ensure that the user is not bewildered by the array of parameters available. Compton and Mateas's (2015) design patterns are a valuable source of guidance here, as is the literature on HCI in general. In particular we plan to emphasise the more tactile aspects, such as freehand drawing and drag-and-drop interfaces.
- Improving the automated playtester, to allow it to play a wider variety of games, and possibly to allow it to learn by observing the user play. Here the emphasis is to ensure that designing the playtester is as easy and enjoyable as designing the game.
- Implementing better search techniques such as hillclimbing, evolutionary and constraint solving approaches. The search methods presented in this paper are encouraging as a baseline, but too slow to run on a mobile device.

Lovell and Fahey (2012) write of the *Starbucks test* for casual games: the ability to give the player a meaningful experience in the time it takes the barista to make their coffee. With Gamika we hope to achieve something similar but more ambitious: to allow users to make meaningful progress towards designing an entirely new game or level in the space of a few minutes. The Starbucks test is one of the factors which has widened the demographic of people who play games on a regular basis; we hope that Gamika will similarly bring the joy of game design to a wider audience.

Acknowledgments

This work has been funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities). We thank the anonymous reviewers for their useful comments.

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. 2012. Computational Creativity: The final frontier? In *Proceedings of the 20th European Conference on Artificial Intelligence*.

Colton, S.; Cook, M.; and Raad, A. 2011. Ludic considerations of tablet-based evo-art. In *Proceedings of the EvoMusArt Workshop*.

Compton, K., and Mateas, M. 2015. Casual creators. In *Proceedings of ICCC*.

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. 2016. The ANGELINA videogame design system, parts i and ii. *IEEE Transactions on Computational Intelligence and AI in Games*.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. 2009. The WEKA data mining software: An update. *SIGKDD Explorations* 11(1).

Juul, J. 2009. *A Casual Revolution: Reinventing Video Games and their Players*. MIT Press.

Khalifi, A., and Fayek, M. 2015. Automatic puzzle level generation: A general approach using a description language. In *Proceedings of the First Workshop on Computational Creativity and Games*.

LeBaron, D. M.; Mitchell, L. A.; and Ventura, D. 2015. Intelligent content generation via abstraction, evolution and reinforcement. In *Proceedings of the AIIDE Workshop on Experimental AI in Games*, 36–41.

Liapis, A.; Yannakakis, G.; and Togelius, J. 2014. Computational game creativity. In *Proceedings of ICCC*.

Lim, C.-U., and Harrell, F. 2014. An approach to general videogame evaluation and automatic generation using a description language. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*.

Lovell, N., and Fahey, R. 2012. *Design rules for free-to-play games*. GamesBrief.

Nelson, M. J., and Mateas, M. 2008. Recombinable game mechanics for automated design support. In *Proc. of AIIDE*.

Nielsen, T.; Barros, G.; Togelius, J.; and Nelson, M. 2015. Towards generating arcade game rules with VGDL. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*.

Perez, D.; Powley, E. J.; Whitehouse, D.; Rohlfshagen, P.; Samothrakis, S.; Cowling, P. I.; and Lucas, S. M. 2013. Solving the physical travelling salesman problem: Tree search and macro-actions. *IEEE Transactions on Computational Intelligence and AI in Games*.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couetoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 general game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*.

Quinlan, R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Schaul, T. 2013. A video game description language for model-based or interactive learning. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*.

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*.

Williams-King, D.; Denzinger, J.; Aycock, J.; and Stephenson, B. 2012. The gold standard: Automatically generating puzzle game levels. In *Proceedings of AIIDE*.

Zook, A., and Riedl, M. O. 2014. Automatic game design via mechanic generation. In *Proceedings of AAAI Conference on Artificial Intelligence*.