Semi-automated Level Design via Auto-Playtesting for Handheld Casual Game Creation

Edward J. Powley, Simon Colton, Swen Gaudl, Rob Saunders and Mark J. Nelson

***Abstract*—We provide a proof of principle that novel and engaging mobile casual games with new aesthetics, game mechanics and player interactions can be designed and tested directly on the device for which they are intended. We describe the Gamika iOS application which includes generative art assets; a design interface enabling the making of physics-based casual games containing multiple levels with aspects ranging from Froggerlike to Asteroids-like and beyond; a configurable automated playtester which can give feedback on the playability of levels; and an automated fine-tuning engine which searches for level parameterisations that enable the game to pass a battery of tests, as evaluated by the auto-playtester. Each aspect of the implementation represents a baseline with much room for improvement, and we present some experimental results and describe how these will guide the future directions for Gamika.**

I. INTRODUCTION

Mobile gaming is an important part of game culture and has become an everyday activity for a large fraction of our society. Within mobile gaming, casual games, categorised by relatively shallow learning curves, comparatively simple game mechanics and relatively short time investment for an engaging return, are extremely popular. Indeed, such games have broadened the popularity of gaming, contributing towards a recent shift in the demographic of game players, i.e., to a more gender-balanced state with a higher average age than a decade ago [1]. Unlike music, photography, writing, abstract art and many other areas where handheld *casual creator apps* [2] have enabled consumers to become creative producers, the creative act of game-making is not yet fully supported on mobile devices. In section II, we partially categorise handheld applications that empower game creation and highlight limitations of existing tools.

A desire to reduce these limitations and further democratise game design has led us to build the Gamika iOS application. We believe it to be the first tool enabling entirely new casual games (containing multiple levels with novel aesthetics, game mechanics and player interactions) to be designed on a mobile phone without requiring coding. One of our main contributions here is a breakdown of a subset of physics-based casual games into a set of components so that games can be built by making choices for each part. In section III, we describe this breakdown, and provide a description of the Gamika software, focusing on how it enables the construction of game levels through a copy-and-tweak methodology, involving generative art assets, a mutation mechanism, drawing functionality, and fine-grained tuning of the game components. Gamika, along with some other mobile apps, enables the production of clones of existing games, which are safe in the sense that the gameplay has been pre-tested. However, Gamika also empowers designers to produce entirely new games with new game mechanics and untried gameplay. This increases the need to extensively tune and playtest game levels, a potentially tedious task that could reduce users' enjoyment of the app. Since we'd like the app to be fun to design as well as play games in, we have endeavoured to make tuning and testing of levels an entertaining experience. In section IV, we introduce a puzzle/reaction game called Let it Snow, made with Gamika, as a running example game for which playtesting and tuning is required for each level. To aid designers in rapidly producing levels, we have implemented an automated playtester in Gamika, which can be watched while playing and is configurable, as described in section V. To further support designers in producing game levels, we have implemented an automated tuning facility, which can search for values of user-specified parameters which fix various failing aspects of a level design. This functionality is described via a case study in section VI. Here, a designer first designs an original level and then plays it to find values for the parameterisation of an imperfect playtester. Logs of both human and computer gameplay, in addition to a parameter sweep, highlight aspects of the game which aid the designer in making new levels. The designer then specifies a test suite that a game level must pass in order to be considered as part of a game design. Finally, the ten produces five new levels of Let it Snow by simply drawing one element of it in Gamika, and letting the software fine-tune various parameters to find a design which passes the tests. The work here represents a proof of the principle that ondevice casual game creation, beyond producing clones and not requiring coding, is possible. 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. Each part of the Gamika implementation is sub-optimal and can be improved via better user interaction design, faster search techniques and more sophisticated parameterisations, which will be informed by the results of the case study, as described in section VII.

II. BACKGROUND

Compton and Mateas [2] introduce the term *casual creator* to describe a piece of software with which users can quickly and easily create artefacts such as musical compositions, artistic imagery such as filtered digital photos, abstract artworks and graphic designs, and texts such as stories and poems. We

are interested in casual creators that work directly on handheld devices, rather than those which merely enable deployment to such devices, and in particular those which allow for the creation of digital games or game levels.

While there are many environments in which novice game designers can learn the craft, such as GameMaker: Studio, Scratch and Stencyl, these require PC-based development. Of the environments that enable on-device game/level creation, the following is a partial characterisation:

- Apps which require learning programming skills. Some, such as Scratch Jr or HopScotch, are designed to introduce children to coding.
- Apps which enable only the skinning of existing game templates, which allows for some level of creativity, but not the entire game making experience. Examples here include Coda Game and Playr.
- Apps which enable the authoring of fairly complex game levels for an existing game. An example here is Createrria 2, where levels of a side-scrolling platform game can be created on-device. These empower creative expression, limited to the provided characters, rules and game worlds.

Of particular note here is Sketch Nation, where the final two categories above are combined, i.e., users can create complex levels within a number of templates.

We defined a space of casual games by factoring Gamika games into a set of numerical parameters (elaborated in the next section). The idea of defining a space of games has similarities to systems such as VGDL [3] and PuzzleScript. With these systems, games are mapped into a space of hierarchical code structures, whereas Gamika uses a space of numerical vectors. Gamika also differs in its use of simulated physics. That is, while those systems define explicit movement rules for in-game objects, Gamika specifies only the physical properties of the objects and the environment, from which movement emerges. This reliance on emergence changes how the space is navigated: on one hand, it reduces the ease of finding specific designs that users may have in mind, but on the other, it increases the chances of the parameters combining in unanticipated and serendipitous ways.

One of our ultimate aims is for Gamika to generate entire casual games automatically, and the work presented on automatically fine-tuning game levels is the first step towards this. Automatic generation of game levels has been looked at for both VGDL [4] and PuzzleScript [5] [6], in addition to a variety of systems generating Super Mario Bros. levels [7]. Nelson and Mateas [8] formally modularise recombinable game mechanics, so that users can define novel game variants and get automated feedback on properties such as playability, via automated theorem proving. Cook et al.'s ANGELINA system is very influential in automated game generation [9]. There has also been work on defining generative spaces of games in terms of the games' semantic and narrative content, mapping sprites to a relatively fixed set of mechanics [10][11]; here we focus on the space of mechanics rather than theme or meaning, but in future work Gamika may branch out from abstractly themed games to include such elements.

As described in section V, Gamika provides a configurable automated playtester to aid designers in making game levels. For such playtesting, it would initially seem natural to choose a method that is capable of playing any general game (or at least any game expressible in the system). The results of the General Video Game AI (GVGAI) Competition [12] suggest that game tree search approaches, particularly Monte Carlo Tree Search (MCTS), are strong in this area. As well as the grid-based games that have been the focus of the GVGAI competitions thus far, MCTS has been demonstrated to work well for physics-based games [13]. In the domain of game generation, the Mechanic Miner system by Cook et al. [14] uses breadth-first search as a playtester to evaluate generated game mechanics. Reinforcement learning methods such as deep Q learning have also been demonstrated to work well for general videogame playing [15].

One of our design goals is for the playtester's decision-making process to be transparent to the user so that they can design AI-bots to play their game level, and enjoy watching the bot play. Game tree search is conceptually simple, but rarely yields a satisfying explanation as to why the AI player chose a particular action. Reinforcement learning is even less comprehensible to the non-expert. For our purposes, easily explained tactics and strategies are preferable to a trained "black box". Another design goal is the ability to generate or tweak a game with respect to a fixed strategy. The success of search and learning based approaches in general game playing is precisely because they adapt to the game at hand, which puts them at odds with this aim. Thus, as described below, we opted for a simple rule-based player, whose rules are parameterised and exposed through the user interface. Whilst the playtester is not particularly sophisticated in computational intelligence terms, it is well-suited to our aims with the Gamika project.

III. A BREAKDOWN OF PHYSICS-BASED CASUAL GAMES

The Gamika tool is an iOS application developed in the Swift programming language using the SpriteKit game development library's built-in 2D physics engine, a modified version of the well-known Box2D engine. Each Gamika game is an ordered list of game levels, where a level is a triplet of an optional text explaining the rules for players; an optional drawing represented as a vector graphic; and a list of numerical values for a set of 284 parameters which define how a set of game objects look, move and interact with each other and with the player's touches. Starting by analysing a number of classic arcade games like Frogger and Asteroids, and supplementing these with analyses of novel games, we have organically grown the set of parameters. Parameters were added until we were satisfied that the game levels they can define are sufficiently diverse, interesting and engaging.

There are three classes of physics object in a level: multiple *friend objects*, multiple *foe* objects, and the single *controller*. The naming of objects as friends/foes allows designers to attach meaning to game objects, to more easily manage the large number of game parameters, but the user is at liberty to ignore this. All objects have a rigid physics body and a collision shape. One option for the controller is for it to be a decorative abstract art image generated by the ELVIRA

evolutionary art system [16] and the genomes and a set of thumbnails of 1,000 supplied images. The images cover different styles, giving a wide range of choices, and each genome can be mutated or more carefully varied, so designers have a good chance of expressing an aesthetic of interest to them. The first set of numerical game parameters define the mathematical functions, blurring regime and post hoc transformations which dictate how the art asset is generated, as per [16]. Alternatively, the controller can be a hand-drawn shape created within the design interface, which can optionally be combined with an art image, which opens up many more aesthetic possibilities. The contour of the controller image is traced to determine its physical shape, hence the choice of the controller is not purely aesthetic, but can also be an important factor in gameplay. The numerical parameters control the following aspects:

- The **properties** of the friends/foes. These control the object size, shape, colour, sprite image, how the calculation of their boundary is performed, and some physical properties such as their restitution (bounciness), mass and damping.
- The **lighting effects** applied to the background and game objects. These control effects such as spotlights, ambient light, the calculation of a normal map for the background and controller image, and the lit appearance of the friend/foes.
- The **spawning** regime for the friends/foes. These control the spawning positions within time-varying ranges, spawn frequencies, total number of each object allowed, and some spatial constraints for the spawning, such as minimum/maximum distances from each other and being fixed on a grid.
- The **movement** of friend and foe objects both at the start of the level and during the game. These control the force fields acting on the objects via directions and strengths, with parameters for noise, friction and angular/linear drag on objects, speed limits, whether objects can rotate or not, and how joints such as pins, springs and sliders act on the objects.
- the **collision** between friends, foes and the controller. These control whether objects stick, bounce, explode and/or change types on collisions, and timings for these, which screen walls are active and how bouncy they and the controller are, as well as how clusters form and when they explode.
- the **user interface** which move the controller and affect the friends/foes. Tapping, dragging and swiping actions are caught, and the controller can be attached by springs, pins and horizontal/vertical sliders, and can be subject to movement and/or rotation by player touches. Player taps can also explode, halt, reverse or change the type of the friends/foes, and taps on the background can spawn more objects.
- the **progress calculation** The progress calculations which alter three counters: score, health and lives (naming suggestions the designer can choose to ignore). Each calculation adds up to five measures prescribed by events on friends/foes. Events include collisions, explosions, spawning, staying on screen, clusters being formed, and objects entering scoring regions.
- The **end-game criteria** which dictate how the progress calculations and/or game duration terminate the game. These control what constitutes a win or loss, how the overall score is calculated and whether high-scores are recorded.

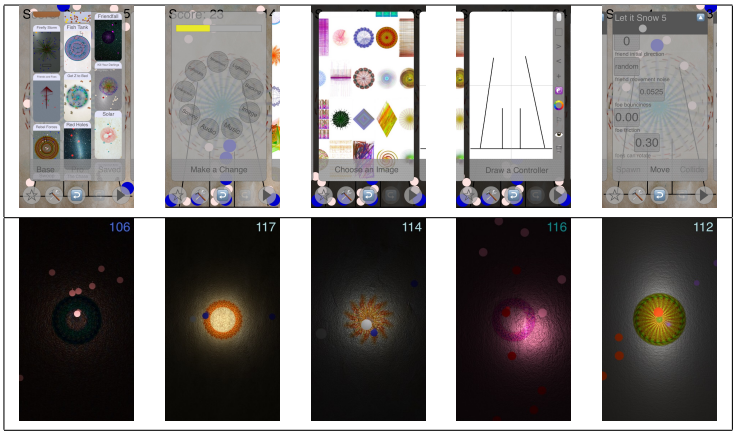

Fig. 1: (a) *basecamp*, mutation, art asset, drawing and parameters screens from the Gamika design interface (b) Five levels in the *Pendulands* game.

Many games can be described in terms of an object with physical properties and spawning rules. Thus, they can be expressed in Gamika. As an illustration, a facsimile of the classic arcade game Frogger can be described as follows. Friends are spawned at the left and foes at the right of the screen and are attracted towards the opposite side by a force-field. The controller is drawn (possibly as a frog) at the bottom of the screen, and can be moved in the four cardinal directions by the player swiping the screen. If the controller collides with a friend or a foe, the game is lost. If the controller reaches the top of the screen, the game is won. Likewise, a simple Asteroids version has the controller (ship) moving towards where the player taps. Friends (bullets) are spawned continuously at the controller's position and move in the direction in which the controller points. At the start, foes (asteroids) are spawned randomly on screen and move subject to a noise field. Friends and foes are destroyed upon collision. If the controller collides with a foe, the game is over.

The design interface includes a *basecamp* screen, from which the user can load one of several preset games, including new original games and homages to classic ones. The main mode of use for the app is that the user loads a *basecamp* game and modifies it, from a simple re-skinning to a complete change of game mechanics. We decided not to provide the option of creating a new game from scratch, to avoid the discouraging effect of the blank canvas. However, it is important to note that no part of the game design is concealed from the user, so any game that can be expressed in Gamika can be created by modifying any existing game.

The *basecamp* screen is the first showing in figure 1. The design interface also includes a mutation screen, shown second in figure 1. One advantage to having games decomposed into

numerical parameters is that they can easily be mutated. The mutation screen has nine buttons arranged in a dial, with each button representing a different aspect of a game level, such as movement, collisions, lighting, etc. The chosen button and the degree by which the dial is rotated dictates which design aspects are varied, and by how much, with bespoke schemes used when the mutation is initiated. Figure 1 also shows the art asset selection screen where the chosen image is generated from an underlying genome as per [16], and the drawing screen, which has a number of editing facilities. Finally, figure 1 shows one of the screens where parameters can be altered directly via a slider interface.

Figure 1 shows five levels from a game called *Pendulands*. Here, the designer (second author) had the idea of having balls spawning from the sides and being attracted to the centre, with any pair of balls exploding if they collide. Players have to move the controller (a circular abstract art image) in order to be underneath the balls for long enough for them to stick to the controller. Once stuck, these collected balls are then under the player's control and need to be protected. The designer used the mutation screen to explore player interaction and lighting design. Two mutations led to important aspects of the game: (i) the player interaction mutation added a spring to the game controller, so that when the player lets go, it springs back to the centre, usually causing collected balls to explode against uncollected ones, and setting progress back (ii) the lighting mutation reduced the ambient lighting significantly, and added a spotlight to the position of the player's finger.

Given that dragging the controller directly causes the player's finger to obscure it, the best way to play the game is to drag the controller off-centre. Dragging off-centre means that the controller is always in the shadows, as the only light in the game is at the player's finger, which is kept in contact with the screen at all times to avoid the controller springing back. The designer built on this aspect of gameplay with a grungy aesthetic, achieved via the use of normal maps. Once the first level of *Pendulands* was made, fifteen more levels were produced relatively quickly, sometimes with only 10 minutes of effort. Each level has similar elements, including a circular controller, dark/grungy aesthetic and the need to catch five balls on the controller to complete the level. However, each level includes a different game mechanic, achieved by tweaking the nature of the balls, where they spawn, what happens when they collide (bounce, stick or explode), how fast they are and how they move. *Pendulands* is one of around 30 games that have been developed with Gamika so far.

IV. RUNNING EXAMPLE: LET IT SNOW

For the remainder of the paper, we will use a running example of a particular game called *Let it Snow* to describe how advanced functionality in Gamika provided automated assistance to the designer (second author) when making the game. In *Let it Snow* levels, see figure IV, the controller is a drawn item, with the art asset being used purely as a backdrop. The player can drag the controller (which we call "jiggling" it). Blue (rain/foe) and white (snow/friend) balls are spawned in random positions at the top of the screen and fall towards the bottom. Balls are spawned at a rate of three white and three blue per second until 20 of each colour are on-screen. Balls bounce off those of the opposite colour, the controller and the screen walls. When two balls of the same colour collide, they stick together. If a cluster of four or more balls of the same colour forms, the cluster explodes. An exploding ball causes a new ball of the same colour to spawn at the top of the screen.

Players gain one point for each white ball that explodes and lose one point for each blue ball that explodes. Players can tap blue balls, which causes the tapped ball to explode, tapping white balls has no effect. While tapping a blue ball will lose the player a point, it may cause a cluster of whites to form and explode which will score more than the loss. Jiggling the controller can unlock occasional stalemates. The aim of the game is to reach a score of 100 points in a time which beats the player's current personal best. If the level is completed within 60 seconds, the player can move on to the next level.

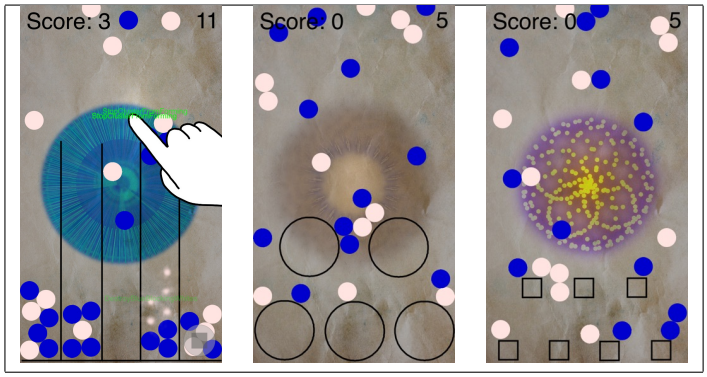

Fig. 2: Screenshots from Let It Snow

Let it Snow is a difficult, fast moving, reaction game, and it takes some practice to become proficient at it. Novice players tend to fail to get control of the game and scores can plummet. An interesting aspect of the gameplay is that, from time to time, an arrangement of balls arises where no more clusters are formed; hence the game comes to a halt as no new balls are spawned. In these quiescent moments, the game takes on a puzzle character, as the player can take their time to decide which blue ball to destroy in order to gain the most points through whites forming clusters. The designer discovered that a good strategy for players is to concentrate on getting to these quiescent moments, then carefully keeping control of the situation through selectively destroying blue balls. A strategy to make the game reach quiescent moments is to react to any potential cluster of four blues and stop it forming by destroying one of them. If the player is successful in this, eventually the blues and whites become distributed over the five columns of the controller, locked out of clusters by balls of the opposite colour.

A winning strategy is to get all the blue balls locked in singletons, pairs and triplets at the bottom of the screen. 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, as only whites are being spawned, the game has the look of snowing (hence the game's name), and players can sit back

and watch the score increase rapidly as it snows, although expert players may use a more proactive tactic. An expert player can usually reach 100 points within 60 to 90 seconds, with novices often taking more than three minutes to complete level 1. *Let it Snow* is quite an addictive and engaging casual game, but presented a number of difficulties for the designing of new levels. We explore how automation has helped address some of these difficulties in section VI, after we describe how automated playtesters can be configured to play the game.

V. Automatic Playtesting

To support designers in making casual games, we have implemented a configurable automated playtester which can be parameterised on the device to describe an AI-bot to play game levels. Currently, the exposed playtester design parameters are not as extensive as those for level design and are rather focused on playing *Let it Snow*; we intend to grow the parameter set in future work. The automated playtester "ticks" at a frequency of once per game frame (normally 60 per second), and analyses the state of the game. The designer chooses a number of *tactics* for the bot to employ, each formed of a condition or a pattern to check for, and an in-game action to execute if the pattern is found. The tactics are arranged in priority order, i.e., the $(i+1)$th tactic takes effect only if the ith tactic does not. The following are the tactics used in a *Let it Snow* playtester, which through design and experimentation we have found to be a human-competitive player for the game.

- **Stop blue clusters from forming.** If two blue balls are close to one another, and they are not already in a cluster, and if the balls colliding would form a cluster of four or more, then tap the faster moving ball. "Close to one another" means that either the two balls are currently within a threshold distance of each other, or they are predicted to be before the next AI tick.
- **Pop blues blocking whites.** If a blue ball is touching two or more white balls which are in distinct clusters, and those clusters contain four or more balls in total, and the blue ball has met this criterion continuously for 1 second (a value which is changeable), then tap the blue ball.
- **Tap if quiescent.** If 2 seconds (a value which is changeable) have passed with all balls having zero velocity, then tap a blue ball that is touching a large number of white balls and is itself either unclustered or in one of the smallest clusters.
- **Jiggle if whites are stuck.** If 5 seconds have passed without a white ball being able to spawn, shake the board in a small random direction.
- **Do nothing.** If no above conditions are met, do nothing.

The "perfect" version of the playtester exhibits super-human accuracy and reaction speeds. We also consider an "imperfect" playtester, with a limit imposed on the number of actions per second and random noise added to its tap positions. We analysed logs from three games played by the first author, and identified his maximum number of taps per 1-second window was 3 and the average distance from the centre of a tapped ball was 10 pixels. Thus, we restrict the imperfect player to 3 taps per second and add random noise with magnitude $2 \times 10 = 20$ to its taps. Additional experiments using a small random time delay in bot actions proved to be far too detrimental to the bot's performance: as implemented currently, the "stop blue clusters from forming" tactic results in the bot waiting until the last possible moment to tap the incoming blue, so even a small delay results in a high miss rate. A tactic that tried to anticipate blue clusters further in advance would likely solve this problem. We acknowledge that the above is a basic attempt at mimicking the limitations of human players, but it does serve to dull the automated player's super-human qualities. Creating playtesters that more convincingly try to model human players such as [17], and possibly the user in particular, is a subject for future work.

As per our desire to make interacting with all aspects of Gamika enjoyable, we have visualised the playtester so the designer can see exactly how the bot is playing the game, and enjoy the experience. The first screenshot in figure IV shows the auto-playtester's simulated hand interacting with the game. We have also added a slider to the design interface which forces the auto-playtester to play at 1, 2, 4 or 8 times normal speed. The speed up is achieved by increasing the physics world speed in SpriteKit which unfortunately means that the playtester is exposed to proportionally fewer ticks, and the bot design had to be altered to cope with this. At higher speeds, therefore, the playtester's ability to play the game is reduced, but we have found that at four times normal speed, it still plays at human-comparable levels. The speed up allows designers to more quickly understand game levels through playtesting, and speeds up the automated fine tuning, as described below.

VI. Case Study

While the designer of both *Pendulands*, described above, and of *Let it Snow* is the same (the second author), the design process has been quite different. For *Pendulands*, the initial level took around 2 hours to perfect, and subsequent levels took much less time – for some, it involved making some changes to the game mechanic and aesthetics, then playing the game around 10 times to test whether it was difficult enough, but not too difficult. *Let it Snow* is perhaps a more interesting game, with simple rules yet high difficulty, and both a puzzle and reaction element, and potentially addictive qualities, which may drive players to want to improve their high score.

However, the designer found making levels for *Let it Snow* much more difficult than for *Pendulands*. In particular, the random nature of the spawning of the balls introduced quite a difficult design problem. That is, the designer found that the game is heavily, but not entirely, luck-based, and observed situations where an extremely lucky run of balls allows a novice player to achieve an expert-level score with hardly any interaction at all. Hence, it was difficult to tell whether it was possible to get better through practice, which is rather a pre-requisite for an enjoyable game of this type. It was for this reason that the designer configured the auto-playtester as per section V, and used it to analyse level 1, as described in the following subsection. Making new levels was also a challenge,

and the designer relied on automated fine-tuning of 8 game parameters, guided by the auto-playtester, as described in the second subsection below.

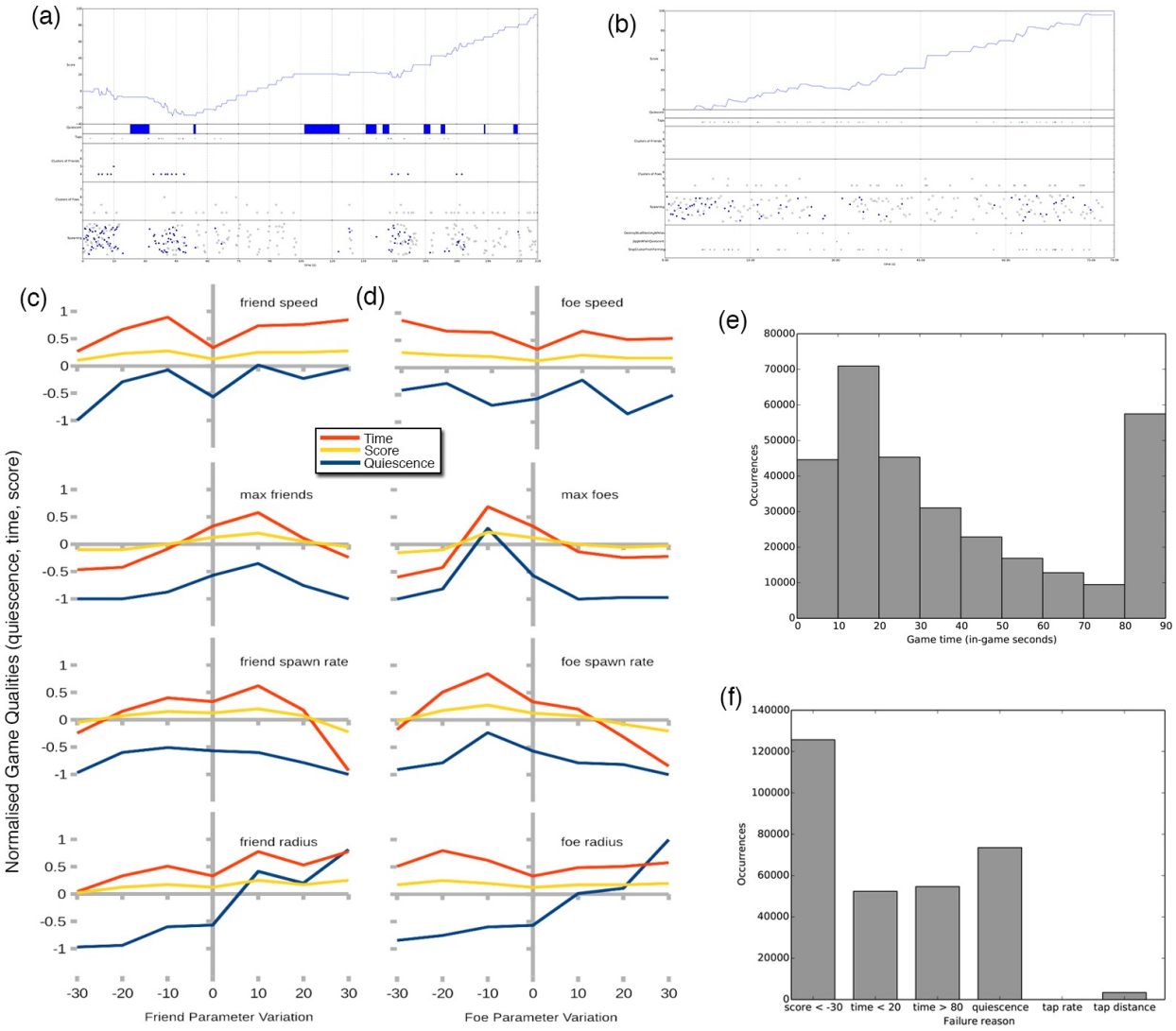

Fig. 3: (a) Log of human playing Let it Snow level 1 (b) Log of "perfect" auto-playtester playing Let it Snow level 1. (c)/(d) Analysis of the change in three normalised game qualities (quiescence, time and score) as four parameters for friend/foes are varied (e) Discretised game play time for the tuning sessions (f) Occurrences of each test failure in the tuning sessions

A. *Automated Playtesting Analyses*

Let it Snow level 1 is on the verge of being too affected by randomness to provide viable gameplay. Indeed, the designer struggled to understand whether the use of tactics helped get better scores, as it is possible to play quite skillfully using various tactics, yet on occasion still perform poorly due to bad luck. Hence, the designer first used Gamika to produce game logs of both a human player (author 1) and the "perfect" auto-playtester. These logs are presented in figures VI(a) and (b) respectively. We see that in the human log, the score (top blue graph) dips during the early stage of the game, as the player struggles to get control of the level. Control is initially achieved through a quiescent moment, and these are achieved throughout the game. This was a trend seen in multiple logs and confirmed the designer's hunch that aiming for quiescence early on is a good tactic.

The auto-playtester logs show that the bot was able to complete the level with perfect application of the tactics described in section V above, i.e., without requiring the quiescent moments to reflect on the state of the game. Hence, the bot can play successfully in a different way to the human player, and it was encouraging that it played to a high standard (indeed, it completed level 1 in around 60 seconds on the second playing). The designer gained the biggest insight into the game when they compared on-screen the "imperfect" bot against the "perfect" one: double checking the logs confirmed that more skilful play does indeed lead to better completion times.

The designer decided that new levels would be created with just a change of drawn game controller and backdrop image, as in figure IV. The designer found with level 2, the

way in which the balls bounced off the new controller (i.e., circles in level 2 rather than vertical lines in level 1), and the way the balls are collected in the gaps differently made the level feel sufficiently novel compared to the previous one. However, after designing level 2, he found that the game was even more difficult than level 1 because quiescent moments never happened. After some experimentation, the designer realised that to achieve quiescent moments, the number of balls allowed on-screen at any one time needed to be reduced. To the designer's taste, this reduced the fun of the game, as it was less dynamic. On experimenting with other ball properties, namely the size, speed and spawning rate, the designer found that these properties could also be used to improve the game design.

The designer suspected that the relationship between the ball properties and aspects of the level such as completion time was non-linear. To investigate this, they returned to level 1 and used Gamika to perform a parameter sweep of the level. This was achieved by systematically altering the eight parameters (size, speed, spawning rate and maximum allowed on-screen for both ball types) within the range -30% to +30% around their original value, using a 10% step-size and employing ten independent trials for each variation of a parameter. This resulted in a total of $((8 \times 6) + 1) \times 10 = 490$ runs of each altered level, which took around 6 hours of processing on the device. For each of the parameters, Gamika calculated the mean value for the number of quiescent moments μ (with a quiescent moment defined as a continuous stretch of time where no ball has a non-zero velocity), the score the player achieved s and the time until the level finished, δ. The raw values were normalised as follows: s within the range of $[-30, 50]$ points, δ within the range of $[0, 90]$ and μ within the range of $[0, \mu_{max}]$, and mapped to the interval $[-1, 1]$.

Figures VI(c) and (d) show the results of the parameter sweep. As suspected, the non-linear nature in which the game design affects scoring, completion time and the number of quiescent moments is highlighted. The designer found that the parameter sweep was very informative and will no doubt help with future level design. In particular, the analysis highlighted certain sweet spots which can be exploited to fix faulty levels, e.g., if levels are too easy, increasing the foe speed by 10% would increase the average game completion time. One exception to non-linearity is the approximate positive correlation of ball size with the number of quiescent moments. This was surprising to the designer, as they had expected the opposite, as larger balls tend to have more collisions, hence more explosions and spawning and a more dynamic, less quiescent level. However, on investigation, it appears that smaller blue balls are less likely to get stuck in non-exploding clusters, as they have more room to move around in, which leads to clusters forming more often and less quiescence.

B. Automated Fine Tuning of New Levels

We want to enable people to make experimental games with Gamika, without having to spend an inhibitively long time designing and playing them. To do so, we intend for the app to have intelligent search methods that can take a nearly-finished level and fine-tune various parameters so that the altered level passes a series of user-defined tests. In a baseline experiment to investigate the potential for this, the designer created levels 2 to 6 of *Let it Snow* as per figure IV. The levels were altered to end after 50 points, rather than the original 100 points, to improve testing speed, and a timeout of 90 seconds was applied. Using Gamika, the designer defined a test suite for new levels as follows: a level fails if (a) at any time, the score reaches -30 or below, or (b) the bot completes a level in less than 20 seconds or more than 80 seconds, or (c) the game contained fewer than two quiescent moments, or (d) the tap rate was more than 3.5 per second, or (e) the average distance between consecutive taps was more than 200 pixels.

TABLE I: Curation analysis for *Let It Snow* Levels 2 to 6

Level	trial	mod	Time	Prop.	Tot.
2	478	2	1h 19m	3/5	16
3	408	1	1h 04m	2/5	12
4	400	2	1h 00m	3/5	14
5	7531	3	17h 52m	2/5	6
6	402	1	0h 50m	3/5	11
av.	1843	1.6	4h 25m	2.4/5	11.8
av2	422	1.5	1h 3m	2.75/5	13.25

The designer specified that the size, speed, spawning rate and number-allowed parameters for both friends and foes should be the subject of random variation. These were chosen as the parameters that have the largest effect on gameplay, without having such a large effect as to make the game no longer recognisable as *Let it Snow*. For a given newly-designed level, Gamika randomly varies each of the eight parameters within -30% to +30% of its value for level 1. A *trial* involves a tweaked level being played three times at four times the normal speed by the "perfect" auto-playtester. If any of the three playthroughs fails any of the above tests, the whole trial is discarded. If, however, all three playthroughs pass the tests, then the altered level is saved to the app, and a new alteration is sought, which continues until the user stops the process. We have found that these tests discard levels for being too easy or difficult, and also levels that cater somewhat to the super-human abilities of the playtester, which can be relentlessly accurate, where a person probably cannot. This testing set up is quite efficient, as the "perfect" bot is the fastest at completing a level, and the order of the tests means that many broken levels fail very fast, often within a few seconds.

We undertook the random approach to see whether successful tweaks are possible automatically and to set a baseline against which more intelligent search methods will be assessed in future. Not surprisingly, we found that the random approach was not particularly efficient, rejecting tens of thousands of tweaked games on average before finding one that passed the tests for three plays. Note that we ran these experiments overnight in parallel on a simulated desktop version of Gamika. Figure VI(e) portrays the proportion of games which finished in certain time bands. Note that those finishing between 80 and 90 seconds were those which timed out. We see that a large proportion of the games ended quickly, in 20 seconds or less (or 5 seconds at $4\times$ physics speed). This is

explained in figure VI(f), where the breakdown of the reasons why levels failed is given: the test to see whether a game is so difficult that the very able auto-playtester gets to a -30 score is very effective, and this quickly rules out bad levels. The high proportion of games that fail due to lack of quiescence confirmed the designer's view that producing a level to achieve such quiescent states was difficult. While the tap distance test was used to discard some trials, the tap rate was not.

To assess whether the random variation and auto-testing scheme works, the designer undertook a *curation analysis* (as introduced in [18]) of the tweaked *Let it Snow* levels. In particular, for each level, he played the first five output variations in the order in which they were produced. Each variation was played 10 times, and the designer decided whether the level was good enough to be added to the game, or should be discarded, recording the reasons for either decision. The *curation analysis* results are presented in table VI-B. For level 2, the second (Mod)ified level shown to the designer, after 478 (Trial)s and a (Time) of 1 hour 19 minutes, was deemed the first one that was good enough. The (Prop)ortion of the first five modified levels shown to the designer which were deemed good was $\frac{3}{5}$, and these came from a (Tot)al crop of 16 generated over a 48 hour period. We see that the other levels have similar results, with level 5 being an exception. Here, only six levels passed the battery of tests in 48 hours and the first good one came after 17 hours 52 minutes. This level is clearly an outlier and suggests the controller drawn in this case admits a much smaller space of playable levels.

Ignoring this outlier, the average waiting time for a good level was around 1 hour, which, while clearly still too long to wait, is encouraging for a random search. The line marked av2 in table VI-B provides averages for the four levels excluding level 5.

Of the levels that the designer saw but rejected, the main failure was that the level was too easy, encouraging too passive a playing style because the snowing moments happened too easily and the player was not actively able to improve their completion time. Some of the games were rated as very good by the designer. Sometimes this was because they had the feel of level 1, but for others, it was because they were quite novel, e.g., the second modified version of level 6 was relatively slow moving and had a greater emphasis on the puzzle element of the game, with very interesting quiescent moments.

VII. Conclusions and Future Work

With the work presented here, we believe that we have demonstrated in principle that novel, interesting and engaging casual games which are truly more than levels of an existing game world, or skinnings of templates can be produced on a hand-held device without the requirement for coding. With the minor case study of the *Pendulands* game and the major case study of *Let it Snow*, we have shown that the Gamika app has much potential to help democratise game design so that broader sections of society can make digital games. *Let it Snow* is the kind of difficult to design game that requires automated assistance in producing levels. We showed that an automated playtester could be designed in Gamika that could play *Let it Snow* levels to a (super) human standard and that getting the AI-bot to play the levels on the device was (anecdotally) entertaining, and could elucidate valuable insights into game levels. We further showed that a random search for tweaks of game parameters, with levels tested by the playtester, could be used in semi-automated game design, where Gamika fine-tunes game mechanics to find suitable modifications of levels. The results from this approach for four of five new levels of *Let it Snow* were encouraging. While we have shown in principle that the approach is viable, there is much work on all aspects which will need to be carried out before Gamika can be released commercially, which is our intention. In particular, we are currently undertaking the following improvements to Gamika:

* Further expanding the space of games available through the app, by implementing more intelligent behaviours in the friends and foes, possibly via the on-device specification of fitness functions and behaviour trees. * Improving the user interface to the game design parameters, employing more drag-and-drop functionality and expanding what can be defined through the drawing interface. * Making the automated playtester much more generic so that it can be configured to play a wide range of games. We also plan to experiment with an approach to training the playtester through play coupled with structured answering of questions, which will hopefully be entertaining, in a pedagogic way. * Implementing more intelligent fine-tuning search techniques. The random approach has provided a baseline, but it takes far too long to find viable solutions. Hence, we have started work on a hill-climbing method and we will also investigate evolutionary and constraint solving approaches.

We agree with Liapis et al. [19] that videogame design is a killer application for Computational Creativity research [18], and we are very interested in Gamika becoming a creative game designer, much like the ANGELINA system [9]. Moreover, we believe that Computational Creativity is ready to have an impact on gaming culture, and we hope to help bring this about through the Gamika app, which will co-create games with designers from all backgrounds, complimenting in many people the joy of game playing with the joy of game design.

Acknowledgements

This work is funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities). We are grateful to the many people who tested Gamika and provided valuable feedback.

References

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

[2] K. Compton and M. Mateas, "Casual creators," in *Proceedings of the Sixth International Conference on Computational Creativity*, 2015.

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

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

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

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

[7] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of level generators in the mario ai framework," in *Proceedings of the Foundations of Digital Games Conference*, 2014.

[8] M. J. Nelson and M. Mateas, "Recombinable game mechanics for automated design support," in *Proc. of the AIIDE Conference*, 2008.

[9] M. Cook, S. Colton, and J. Gow, "The ANGELINA videogame design system, parts i and ii," *IEEE Trans. Comp. Intell. AI Games*, 2016.

[10] M. J. Nelson and M. Mateas, "An interactive game-design assistant," in *Proceedings of the 13th International Conference on Intelligent User Interfaces*, 2008, pp. 90–98.

[11] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, "The micro-rhetorics of game-o-matic," in *Proceedings of the Procedural Content Generation Workshop*, 2012.

[12] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general game playing competition," *IEEE Trans. Comp. Intell. AI Games*, 2015.

[13] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Travelling Salesman Problem: Tree Search and Macro-Actions," *IEEE Trans. Comp. Intell. AI Games*, 2013.

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

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[16] S. Colton, M. Cook, and A. Raad, "Ludic considerations of tablet-based evo-art," in *Proceedings of the EvoMusArt Workshop*, 2011.

[17] S. E. Gaudl, J. C. Osborn, and J. J. Bryson, *Learning from Play: Facilitating Character Design Through Genetic Programming and Human Mimicry*. Cham: Springer International Publishing, 2015, pp. 292–297. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23485-4_30

[18] S. Colton and G. Wiggins, "Computational Creativity: The final frontier?" in *Proceedings of the 20th European Conference on Artificial Intelligence*, 2012.

[19] A. Liapis, G. Yannakakis, and J. Togelius, "Computational game creativity," in *Proceedings of the Fifth International Conference on Computational Creativity*, 2014.