A Parameter-Space Design Methodology for Casual Creators

Simon Colton, Mark J. Nelson, Edward J. Powley, Swen E. Gaudl
Rob Saunders, Blanca Pérez Ferrer, Peter Ivey and Michael Cook
The MetaMakers Institute, Falmouth University, UK

`www.metamakersinstitute.com`

Abstract

Casual creators are creativity support tools intended to be fun and easy to use for exploratory creation. We have built casual creators called fluidic game designers, which support exploratory game design directly on mobile devices. These work by encoding games in a parameterised design space and enabling player/designers to create new games by varying the parameters in a design interface and seeing changes in their design instantly. A question largely unanswered by existing work is how to choose a suitable parameter space. We describe a methodology for specifying and implementing parameterised design spaces for casual creators, a context that requires balancing a large and expressive space against a manageable and fun user interface. This methodology was derived through investigating and generalising how the parameter spaces for three fluidic games were conceived. It suggests an iterative process whereby parameters are sourced in seven different ways, within a dynamic of incremental expansion and contraction of the parameter spaces.

Introduction

A *casual creator* is design software for creativity support that is oriented towards exploration, by definition enjoyable to use and with a low barrier to entry. Compton and Mateas (2015) describe the act of making with casual creators as an: "intrinsically pleasurable activity, rather than an extrinsically-motivated way to accomplish tasks". Hence casual creators contrast with standard design software such as Adobe's Creative Suite, which typically emphasise large feature sets, often with a complex interface, oriented towards professional productivity. We have been building casual creators that we call *fluidic game* designers, which support exploratory videogame design on mobile devices (smartphones and tablets) in minutes and hours rather than days and weeks, and without a requirement for programming (Nelson et al. 2017a; 2017b; Gaudl et al. 2017).

Our motivation is to bring design of mobile games to the kind of large and diverse audience that characterises mobile-game playing, making casual creators that are more akin to casual games than to professional game-design tools. Fluidic games are designed to reduce the context gap between play and design, by allowing games to be designed on the same devices that they're played on, enabling the user to rapidly alternate between those two modes of game player and game designer, enjoying themselves in both roles.

A casual creator can be seen as an accessible, enjoyable *design space explorer* – a class of design tools that visualise and support navigation of a space of design possibilities supported by the tool, dubbed the *design space* (Woodbury and Burrow 2006a; 2006b). For fluidic games, we use *parametric design* as the technical approach for representing the design possibilities that we support (Woodbury 2010). Parametric design represents design possibilities through an enumerated set of parameters, where each parameter is a possible design choice. Thus the design space is constructed explicitly: N parameters produce an N-dimensional design space, and each design is one point in this space, i.e. one choice of the N parameters. Although computer-supported parametric design is well studied, to our knowledge there is little practical advice on how to construct the actual parameterised design spaces, especially for our purpose of casual creation (versus a context such as engineering optimisation).

Parametric design was pioneered in computer graphics and architecture, where parameters often fall out directly from a choice of "universal" representation, such as a mathematical representation of 3D surfaces. The parameter space then is dictated by this choice of surface representation – for example, choosing a Coons-style (Coons 1967) or NURBS-style (Piegl 1991) surface brings with it a set of parameters. This is also sometimes the case in evolutionary art, where a general representation such as a Compositional Pattern-Producing Network (CPPN) can be used as the basis for design, with parameters falling directly out of the way CPPNs are constructed, as described in (Stanley 2007).

We're sceptical that a useful parameter space for games, especially one for end-user design, can take the form of a similar universal representation. Some parameters will be dictated by underlying technology, such as a physics engine or lighting model, but to cover a meaningful space of casual games requires parameters for characters, player interactions, collisions, dynamics and pacing, visuals, music and audio, scoring, progression and win conditions. Game description languages have identified and formalised many such elements (Schaul 2014; Browne and Maire 2010), but do not in themselves explain how to design a usable parameterised representation. Our main contribution here is to propose a methodology for doing so.

We outline the methods we have used to iteratively build parametric design spaces for fluidic games. The specification of each design space is driven by the twin goals of capturing a meaningful space of games that offers a degree of variety and satisfaction, while providing a simple, enjoyable and comprehensible user interface for design. These two goals are in tension, since expanding creative possibilities requires more parameters and more choices, which produces more complex user interfaces. To address this, our methodology consists of seven different ways to derive parameters within a dynamic of incremental expansion and contraction of parametric spaces in close interplay with the design of user interfaces to navigate them.

To illustrate how and why we arrived at this methodology, we first summarise the development of *Gamika Technologies*, a parameterised game engine. We initially built it by deriving 284 parameters and implementing a user interface called *Cillr* to navigate the parameter space. We describe the parameters and the drawbacks to *Cillr* as an application for developing casual games. Following this, we describe three fluidic games, namely *No Second Chance*, *Wevva* and *Blasta*, which were built by tightly coupling the design of a more focused parameter space with a casual-creator design interface. For each, we describe the derivation of the parameter space and the design of its corresponding casual creator interface, the kinds of games afforded, and some experiments and playtests. This enables us to subsequently present a generalised methodology for parameter-space design of casual creators.

Gamika Technologies

Gamika began as an iOS app for evolutionary art, based on work in (Colton, Cook, and Raad 2011). An initial set of parameters were derived and exposed in a UI to control the art generator. These were then extended to enable the design of *digital fascinators*, i.e., small visual toys designed to hold a player's attention for a few minutes (Pérez Ferrer et al. 2016). We built several prototypes to turn the abstract art pieces into interactive toys, One prototype was a "whack-a-mole" style game, where players tap particles atop the image before they escape. Another was a safecracking game, where the image is split into concentric rings which the player must re-align. A third was a clone of the classic game Breakout, with bricks made from the art.

The fourth prototype, which we called Friends & Foes, emerged as the most promising. This used a feature in the iOS SpriteKit API: converting an image into an object with realistic simulated 2D rigid-body physics based on its contours. In Friends & Foes, the art image is pinned to the centre of the screen, with the player controlling its rotation. Green and red balls (the eponymous "friends" and "foes") spawn at the edges of the screen and are pulled towards the centre. The aim of the game is to have more friends than foes on screen when the timer reaches zero, by using the art image to bat away the foes, but not the friends. Friends & Foes had promise as a game: it requires equal parts tactics and dexterity, and has "hero to zero" moments: one careless move can result in the loss of all the friends accumulated so far. It also depends very strongly on the abstract art image, so multiple levels with varying difficulties were possible. For instance, a rectangular image gives a very different playing experience to a spiral shaped one, which is different again to a smooth circular shape (which makes the game all but unplayable).

Given this promise, in line with producing a casual creator app, we exposed a number of hard-coded values in the software for Friends & Foes as changeable parameters in a user interface called Cillr (see below). In particular, we exposed: the ratio of friends to foes, their spawning rates, speeds and bounciness, and the time limit. These parameters could be tuned differently for friends and foes, so for example the game was made easier by making the foes bouncier (hence easier to bat away) than the friends. To turn this prototype into a game development engine, we continued to identify and expose more parameters, enabling increasingly varied version of Friends & Foes to be made on-device.

One straightforward extension was to allow friends/foes to stick in place after having been in contact with the image for a certain length of time (another parameter). This gives more sense of progress to the game: once a friend or foe is stuck, it cannot be batted away. Interestingly the game with sticking is still recognisably Friends & Foes, but is also recognisably different. This simple change expanded Friends & Foes into a space of two types of games: one with sticking and one without. There are also interesting points at the edges of the parameter space, e.g., if the sticking time is set to zero, balls stick to the image instantly, and the whole strategy of the game changes: foes can no longer be batted away with the image itself, so the player must build a barrier of stuck friends and use that to bat the foes away instead.

To expand the space of games further, we looked at what other aspects of the game could be parameterised, starting with scoring and game ending conditions. In the original game, friends on screen are worth 1 point and foes worth -1, but we extended this to enable scoring when balls stick or are batted away. Also, originally the game ended when the timer ticked down to zero, but we added a parameter to enable the game to end when a certain score was reached, or when a certain number of balls are stuck. We also added conditions for winning or losing based on timing, number of stuck friends/foes and scores. We continued to challenge our assumptions about what aspects of Friends & Foes should be hard-coded and what should be parameterised, e.g., when two balls collide, they were hard-coded to bounce off each other, but we changed that to enable them to stick to each other, and to explode, with a new parameter. Moreover, we enabled collisions to feed into the scoring system, e.g., when a cluster of a certain size forms all balls in the cluster explode, gaining or losing points (reminiscent of the ubiquitous match-3 genre of games). Points could also be awarded or deducted for batting balls off screen, and we added a parameter for the scoring zones.

At this stage, we determined that all of the games in the expanded space were still too recognisable as Friends & Foes, and the casual creator felt like it afforded only versions/levels/variations of this. To greatly increase the range of games, we employed two main tactics. Firstly, we identified some *inspiring examples* as described in (Colton, Pease, and Ritchie 2001), namely some classic casual arcade games that we felt should be in the space of games achievable with Gamika. These included Frogger, Space Invaders and Asteroids. We then determined which parameters would be needed in order for the space of games to include these examples. Secondly, we looked at the underlying physics engine and lighting rendering system available in iOS through the SpriteKit interface, and exposed parameters which were afforded by the methods and fields there.

We also added parameters in ad-hoc ways, including: a systematic search through code to see if any hard-coded values could be extracted; engaging in what-if ideation to imagine different game mechanics; and adding a parameter in order to turn a bug into a feature, or to balance the usage of a different parameter. We describe these sources in general terms when presenting the design methodology below. Note that, in exposing more parameters, we often came up against certain computational limits, so we sometimes restricted the parameter ranges to make it less likely that users could design games with unacceptably low frame rates.

The Set of Parameters in Gamika

After much development, when the parameter-exposing exercise was completed, we had identified and exposed 284 parameters to the user interface for game design. With these parameters, the space of games afforded was sufficiently large to cover radically different games such as four-in-a-row puzzles, bullet hell games and clones of our inspiring examples. Note that one of the computational constraints we imposed was having only two sets of physics objects with a fixed maximum number of each allowed. This did allow us, however, to continue to describe on-screen objects as friends and foes, with the term 'controller' describing the evolutionary art object which the player controls.

The 284 parameters identified for Gamika can be grouped into several categories as follows.

• **Properties** of friends/foes, including their size; shape; colour; sprite image; bounciness, mass and damping.

• **Lighting effects** applied to the background and game objects, controlling: spotlights; ambient light; the calculation of normal maps; and the lit appearance of the friend/foes.

• **Spawning** regimes for the friends/foes. These set: the spawning positions within time-varying ranges; spawn frequencies; total number of each object allowed on-screen; and spatial constraints, such as min/max distances from each other when spawned and spawning on a grid.

• **Movements** of friend/foes, both initially and during a game, via forcefield directions and strengths with parameters for: noise; friction and angular/linear drag; speed limits; whether objects can rotate or not; and how joints such as pins, springs and sliders act on the objects.

• **Collisions** between friends/foes and controller: whether pairs of friend/foe stick, bounce, explode and/or change types on collisions, and timings for these; which screen sides have walls, and how bouncy they and the controller are; and how clusters of friends/foes form and explode.

• **Player interactions** with the controller and friends/foes: tapping, dragging and swiping actions; how the controller is attached by springs, pins and sliders, and can be subject to movement and/or rotation by player touches; tapping to explode, halt, reverse or change the type of the friends/foes; taps on the background spawning more objects.

• **Progress calculations** altering three counters: score, health and lives, via calculations which add up to five measures prescribed by events on friends/foes, which include collisions; explosions; spawning; staying on screen; clusters forming; and objects entering scoring regions.

• **End-game criteria** which dictate how progress calculations and/or game duration terminate the game, which set: what constitutes a win or loss; how the final overall score is calculated; and whether high-scores are recorded.

The Cillr Design App

As mentioned above, *Cillr* was the initial game design interface to Gamika Technologies, an iOS application that enables users to navigate the entire space of games by setting the values for the 284 parameters described above. It was supplemented with additional design screens to (a) save/load games (b) randomly mutate aspects of the game genomes (represented simply as a list of the parameter values) (c) make drawings which get turned into live physics objects, and (d) browse thumbnails of available evolutionary art pieces. Four design screens from Cillr are shown in figure 1, along with some screenshots of example games.

Each parameter in Cillr is assigned a slider in the UI, and these are distributed over screens – which can be scrolled between – with a manageable number on each. In particular, the sliders are grouped into categories with related functionalities to make them more discoverable (the spawning-related sliders are collated, the collision-related sliders likewise, etc.). Even with these groupings, however, we found that having 284 parameters was unwieldy, as even expert users had trouble simply finding the right parameter to change, and it wasn't always clear why a game hadn't changed as expected. We experimented with enabling users to navigate the game space in an evolutionary way, with a screen performing focused random mutations of games. Producing a random game variant, then trying to figure out what it is, can be a fun interaction loop. However, we found that the proportion of playable games produced in this manner was too low to consider it for end-user consumption.

As an initial baseline, *Cillr* is usable, at least by experts. We have used the interface to produce clones of classic games like Frogger, Asteroids and Space Invaders, as well as a variety of novel casual games. Often, we have found that such novel games *emerge* during design, often in response to unforeseen physical interactions of objects. We investigated such emergence with a narrated set of design sessions, as reported in (Colton et al. 2016). Moreover, in a preliminary user test with game-design undergraduate students to test whether *Cillr* could be used as-is, we found them somewhat frustrated by the experience of using it to make games. Interface complexity was one issue, but more importantly, the difficulty of understanding the high-dimensional design space made it hard for these initial testers to grasp what they wanted to do in the app, and how they would begin to do it. Therefore, rather than focusing on improving *Cillr*'s interface, we resigned it to an in-house tool. For public release,

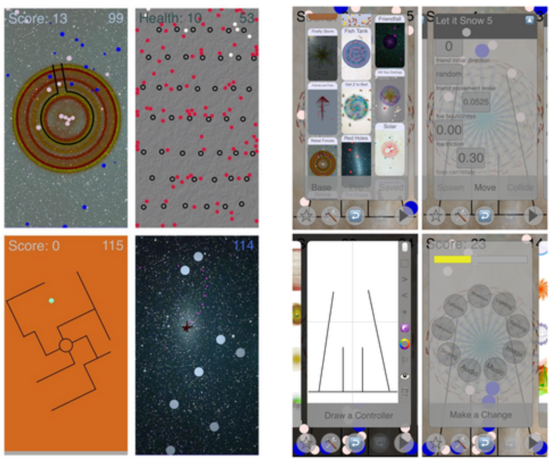
Figure 1: Four Gamika games (left) designed with the *Cillr* app (UI on right). *Cillr* design screens clockwise from top left: List of editable saved games, a screen of movement-related sliders, brainstorming wheel to randomise subsets of parameters, and drawing interface to edit controllers.

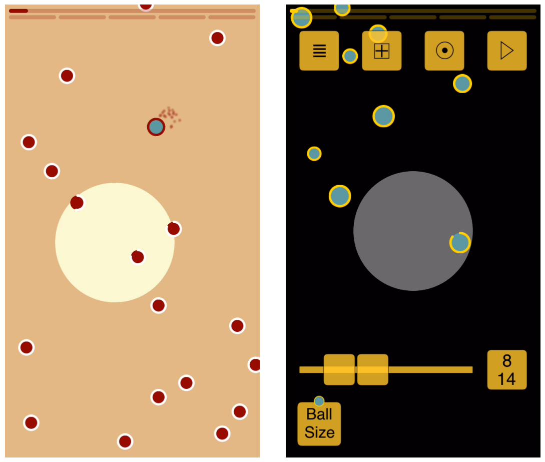

Figure 2: *No Second Chance* gameplay (left) and example design screen (right) for changing the ball sizes.

we have focused on producing design tools for more cohesive subspaces of parameterised games, as described next.

Fluidic Game Design Apps

With the term *fluidic game*, we mean a casual game for which it is very quick and easy to change any aspect of the design, to improve it, make it easier or more difficult, produce a new level/variation or generally explore possibilities and have fun being creative. As such, an individual fluidic game is itself a design tool, enabling players to search for games, rather than a fixed game in the space. We wanted to blur the line between designing and playing a game, so that it becomes natural for people to change games as they play them, hopefully demystifying and popularising game design, similarly to how level design in games such as Little Big Planet or scene building in Minecraft provide gateways to game design. Hence we give each fluidic game its own user interface, rather than sharing a global one – so that the UI feels like an integral part of the game. In the subsections below, we describe three *fluidic game design apps*, which are collections of parametric fluidic games in a single iOS application, with associated administrative screens, e.g., for collating and sharing games, changing global settings, etc.

In order for it to be as much fun to make fluidic games as to play them, we designed their user interfaces as casual creators, often sacrificing parameters which would increase the space of games to maintain a fun design experience. Despite all being 2D physics-based games, the Gamika space is heterogeneous, with some games puzzle-like, others meditative, and others fast action. Sometimes, changing a parameter will design a slight variant of a game, but sometimes it completely changes the game genre, or could break it. We decided that a fluidic game, by contrast, should encompass a smaller parametric design space that (a) can be navigated more deliberately, with understandable relationships between parameter changes and changes in gameplay behaviour (b) retains emergent properties, so that unexpected games can be found, and (c) minimises or eradicates the possibility of producing a broken or slow game.

Having identified a design subspace, it then becomes important to understand the structure of the space well enough to build a casual creator interface that enables designers to control the space's salient features. In the first two fluidic game design apps described below, the parameters shaping the design space are subsets of those of Gamika, previously described in some detail in (Nelson et al. 2017a). In the third app, new parameters have been derived. In each case, the procedure for defining the space of games is based on first cutting down a large set of parameters to a rather small subset, then expanding the set again around the core of a specific type of game. When the expansion happens, this is in close interplay with an interface specific to the genre of game covered by the space. This is followed again by another round of contraction based on user testing with the interface, if this shows that the interface is too complex.

No Second Chance

Using *Cillr*, we designed a game of patience and concentration called Pendulands, where balls (friends/foes) move in a pendulum-type motion and annihilate each other on collision. Players catch balls by hovering under them with a large round target (controller) until they stick. By varying parameters, we discovered that many quite different Pendulands variants could be created. We decided on some fixed elements defining this subspace of Gamika games: the player always controls the target by dragging, and must catch five balls on the target within five minutes. *No Second Chance* is an app built around this space of games. The name comes from a meta-game mechanic: players can share games which are deleted if the receiver doesn't beat the game on the first try (in five minutes). This emphasises the "disposable" nature of games in a generative space, and the challenging nature of figuring out how each one works on first encounter.

The game design screen (see figure 2) is laid out as a hierarchical menu, with submenus allowing visual style and a variety of physics parameters to be changed. As the control and scoring mechanisms are fixed for *No Second Chance* games, new ones are made by varying the nature of movements, collisions and spawning, in addition to visual aesthetics and a soundtrack. Within these constraints, very different types of challenging games can be created. In particular, we found that games requiring gameplay with various levels of *skill* (chasing after balls), *ingenuity* (working out what is going on) and *patience* (waiting for exactly the right moment) were possible. To demonstrate the variety of games that can be produced (and to provide an initial challenge), the app comes with 100 preset games designed using this interface.

To supplement the design interface, we added a 'generate' button, which creates a new game via an evolutionary process. In particular, groups of related parameter settings (e.g., all those for lighting, or movements) from two of the preset games are crossed over and the resulting offspring is mutated. The resulting games are filtered using heuristics to reject clearly bad candidates, and the first four candidates that pass the filter are auto-playtested at super-speed on the device in a split-screen view. As we want games to be playable but not too easy, the app chooses the game that the playtester was able to catch the most balls on, without being able to catch all five. No Second Chance has been tested successfully for game design in an after-school club for 12 year olds, and in rapid game jams (Gaudl et al. 2018).

Wevva

Again using *Cillr*, we made a relatively addictive four-in-a-row game called *Let It Snow*, where snow and rain pour down from the top of the screen (as white and blue balls respectively). When four or more white balls cluster together, they explode and the player gains a point for each, which are then replaced by new ones spawned at the top. Likewise with blue balls, except that the player loses points for them. Players can interact with the game by tapping blue balls to explode them, losing one point in doing so. A grid structure collects the balls into bins, and the best way to play the game involves trapping the blue balls in groups of twos and threes at the bottom, while the whites are exposed above and are continually refreshed through cluster explosions. Occasionally, when all blues are trapped in small clusters, only whites spawn, which looks like snowing (hence the game's name) and is a particularly pleasing moment.

From this single game as a starting point, we first expanded to a larger but still relatively small design space of similar winter-themed puzzle games. These were collected into an app called *Snowfull*, with a fluidic interface allowing designers to change the rain and snow cluster sizes, what happens when the player taps, and the win conditions. A major interface difference from *No Second Chance* was that, given the reduced parameter space, we were able to show a visual overview of all selected parameter values on a single screen, used as (a) an instruction screen for each game, explaining the rules, win conditions, and interaction methods, and (b) the starting point for editing the game.

We conducted a series of 1 to 2 hour rapid game jams using *Snowfull*, with groups from Girlguiding Cornwall (Gaudl et al. 2018). They were able to design games, but gave mixed feedback: despite the casual look-and-feel, the design is oriented towards challenging puzzle-like games,

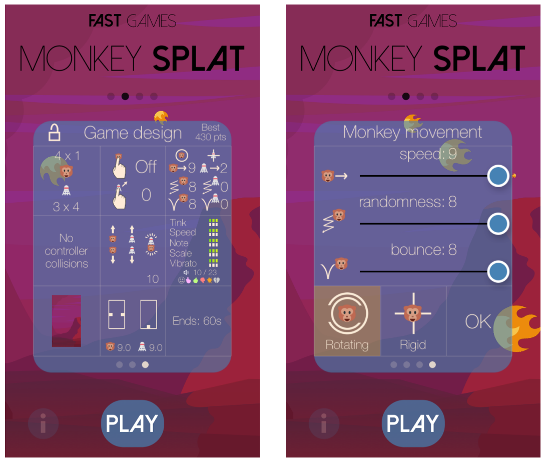

Figure 3: *Wevva*, showing the overview design screen and a sub-design screen for character movements.

requiring finding a carefully balanced interplay of mechanics that both provide challenge and allow for strategies to overcome those challenges. We found that a large proportion of the participants wanted to make much more casual games, and also felt constrained by wintry theme and the relatively small number of parameters available to change, with game mechanics limited to clustering interaction.

Based on feedback like this, *Snowfull* gradually evolved into a fluidic game called *Wevva*, described in (Powley et al. 2017). While it has many more parameters for game design than *Snowfull*, it is in some ways more causal to design with, and games are certainly more casual to play, although it is still possible to make them challenging. Among other changes, we returned to the idea from *No Second Chance* of the player controlling an in-game object (their avatar of sorts), and also added a number of ways to change sprites, backgrounds and music/audio.

Although we significantly expanded the parameter space, we decided it was important that all selected parameters would be shown in a single-panel visual overview, as portrayed in figure 3 (left). Going row-by-row from top left, the nine grid squares show: character options; what tapping does; physics parameters for character movements; effects of collisions between the avatar and other characters; effects of collisions between non-player characters; music parameters; player avatar look/position, interaction and background; spawning and scoring zones; and win/loss conditions. The constraint that the value of every changeable parameter needed to be represented in this screen was a continual presence in discussions of which parameters to add/remove to expand/contract the space.

Not counting audio design, which is a more advanced topic (as volumes and tempos for five tracks can be set), there are 33 design screens enabling the setting of 47 parameters. We have found that this number seems manageable for most users, and the UI enables rapid progression and a comprehensible, satisfying, design experience. This is achieved through a sensible collection of parameters on screens through which there is a logical progression which rarely gets too far from the home screen. The variety of

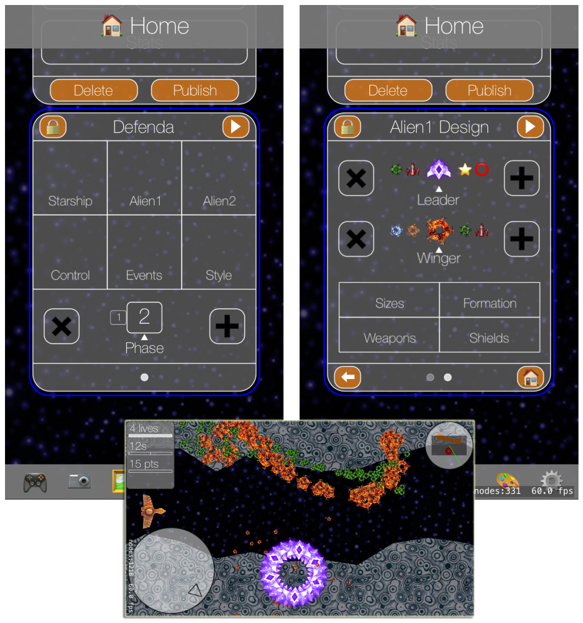

Figure 4: *Blasta*, showing the home design screen, a sub-design screen for alien characters, and a game.

games is evidenced by Wevva shipping (on the iOS app store[1]) with 28 games in four different game packs (simple, fast, skilful and tricky). In each of around 15 game jams with Wevva, we have always seen a few genuinely new game mechanics found (and hence novel games). In the most recent tests of Wevva, we found that secondary school children were able to make novel games in around 15 minutes, with zero preparation and no in-game help.

Blasta

A new fluidic game design app called Ludando is currently being developed by the first author as a commercial development for Imaginative AI Ltd. With this app, players will be able to design different types of games including ones in the *Blasta* genre, which covers certain types of shoot-em-ups and driving games. The fluidic games differ to those in Wevva in a number of ways. Firstly, multiple phases (like levels) can be defined for a single genome, so that games can have long gameplay durations, measuring in hours to complete a game. In the design screen, a new phase is constructed by copying the parameters of the previous phase, so it is easy to increase difficulty from level to level. Secondly, the designer can use their own content, namely photographs and audio files in the game. Thirdly, although *Blasta* games differ somewhat by game mechanic, this is not as emphasised as in *Wevva*, and the parameters are instead used to highly fine-tune and personalise the look and feel of standard shoot-em-up games (which have norms and expectations), rather than discovering brand new casual games.

Screenshots of the design UI and an example game are given in figure 4. We see that the home screen breaks down the game design into three main character types (starship, alien1 and alien2) and has design screens for the control mechanisms, the events and the game style. For the two alien types, the sprite and movement patterns for both leaders and wingers can be set, as can parameters for weapons and shields. For the starship, lives, shields and weapons are parameterised, and settings for how the player controls it (by dragging directly, tapping in a desired location or driving it like a car) are also exposed. Style parameters enable the user to set game aspects including backgrounds, terrain, music, animations and the heads-up display.

At the current stage of development, the app has too many parameters, and the interface is useable but somewhat daunting (as per some initial user testing). We are currently reducing the number of parameters, thus limiting the space of games, in order to improve usability of the design interface. *Blasta* was originally intended to cover stealth games, infinite runners and possibly even platform games, in addition to shoot-em-ups and driving games. However, the parameters for these extra genres have been dropped, as the space was too large and the design interface had become unwieldy (much like Gamika). It is likely that some level of homogenisation of aspects like bullets (which can be altered in detail) is still required, and we plan to give designers collections of parameters, e.g., for movement formations, rather than access to the parameters themselves. After the contracting stage, we will undertake substantial play-testing of the app, which will inform more expansion and contraction.

Constructing Parametric Design Spaces

Based on our experiences designing Gamika/Cillr and the three fluidic game design apps above, we have abstracted out a general methodology for building parametric design spaces specifically for use in casual creators. The methodology consists of two parts: (i) sourcing a set of parameters, and (ii) deciding how and when to add or remove parameters, which we suggest can be achieved iteratively through expansion and contraction cycles. Dealing first with the sourcing of parameters, looking systematically at why each parameter in Gamika and the fluidic games was introduced, we have identified the following seven sources:

1. Capture an initial example: Choose one game to implement, and identify the minimum set of parameters needed to represent that game. At the beginning of the Gamika project, we started with a simple game called *Friends & Foes*, which necessitated some obvious initial parameters, e.g., to capture spawning, speeds, etc.

2. Externalise fixed parameters: Systematically investigate values that were hard-coded when implementing the first game – e.g., constants in the source code – and turn appropriate ones into parameters of the design space.

3. Capture an inspiring example: Think of a game that seems possible to express in the current space, and if it isn't, add new parameters to expand the space until either that game, or something like it, is included. For example, in Gamika we expanded the space in an attempt to capture *Frogger*-like games. We didn't end up with precisely *Frogger*, but the exercise helped us identify a number of new parameters that made sense to add to the design space, and we ended up with Frogger-like games in the space.

[1]https://itunes.apple.com/gb/app/wevva/id1322519841

4. Pass through parameters from underlying technology: Study the fields and input parameters to methods available within the APIs available in the programming environment being used, and expose suitable ones as parameters in the design space. For example, Gamika is built on the iOS SpriteKit in which physics objects have a field called *restitution*, which we just passed through directly as *bounciness*. These are exploratory parameters: in contrast to those added to capture a specific example, they're added opportunistically to see what new possibilities they might enable.

5. Balance other parameters: If it becomes clear that having parameterised a game setting X, you really needed to be able to change Y too, then add the extra parameter. For example, in Gamika, we added the ability for objects of the same type to stick to each other, and to explode once a cluster of a certain size was formed. We then realised that we should provide similar functionality for heterogeneous clusters; i.e., clusters of both friends and foes. Identifying balancing parameters can be done systematically post-hoc.

6. Reify emergent features: When experimenting with a parameterised design space, combinations of features often produce novel emergent effects. It may be possible to make an emergent property available in a game, and parameters for turning it on/off and altering it can be added. Doing so makes them easier to use, as the user is now aware that such effects are possible and can control them directly.

7. Split parameters: An existing parameter may be employed for more than one aspect of the game design due to bundling things together in the code, and it may be possible to split out parameters for each aspect. For example, an initial implementation may have a single lighting parameters which controls the overall light level. Later, this could be split into multiple parameters, such as controlling diffuse and specular lighting separately.

The fourth source of parameters above is the one closest to much of the work on parametric design in graphics and architecture, but in our experience, it accounts for only a minority of the parameters required to specify a casual game.

Given these various ways to identify parameters, the other important feature of our proposed design methodology for parameterised design spaces is when and how to introduce or remove the parameters. In retrospect, we can characterise the development of fluidic games (as a form of casual creator) as involving a series of parameter expansions and contractions, carried out in the following four stages:

• Stage 1: unconstrained expansion. First add a large number of parameters, sourced via all seven methods above, to map out as general a space as is feasible. In our case, this is described above in how we arrived at the parameter space for Gamika. This produces an initial parameterised design space containing a large number of highly varied games, but one that is likely to be too large and heterogenous to be a good basis for a casual creator.

• Stage 2: radically cut down the parameter space to just enough to encapsulate a single example game. Choosing one promising game within a large game space enables the building of a bare-bones casual creator UI for designing variants of that game. For *No Second Chance* and *Wevva*, we cut down the parameters in Gamika to those required just for the initial games, respectively *Pendulands* and *Let it Snow*.

• Stage 3: UI-constrained expansion. Initial playtests will likely find that users feel constrained by the small design space and will want to modify more game aspects than those available in the bare-bones interface. At this point, parameters can be re-added to meet user requests, but with a strong constraint from the UI: every new parameter must at the same time fit cleanly into the UI, keeping the resulting interface fun to use, as per the notion of a casual creator. In several cases, we chose very specific UI constraints to further structure this process. For instance, in *Wevva*, the 3x3 grid structure for the design interface, and the requirement that all parameter values be readable on the top-level screen, strongly guided how we added new parameters.

• Stage 4: consolidation and polishing. Even though Stage 3 only adds parameters under a strong constraint of fitting them into the UI, it is possible that the resulting UI may still become over-complicated. It is also likely that the incremental addition process may have resulted in parts of the UI being somewhat inconsisetnt. At this point, relatively modest streamlining can be done by consolidating or linking similar parameters, rethinking arrangements, etc., in preparation for a final version of the casual creator.

Conclusions and Future Work

We have proposed a methodology for specifying parameterised design spaces for casual creators, derived from our experiences in developing three casual creator apps for broadly accessible game design. Our method includes seven sources of the parameters themselves, and a four-stage process of expansion and contraction of the parameter space – initially as a relatively unconstrained design of parameters in the abstract, and later in tight interplay with the design of a casual-creator user interface. We were surprised to find that, although there is a large literature on parameterised design, there is little written about how to specify the parameter spaces themselves, which is obviously key to the process. Therefore, we believe a methodology such as the one here is a useful contribution and may be of benefit to casual creator designers. In future work, we would like to understand how general such a methodology is when applied to other types of casual creators, and to extend the methodology with more automated methods of parameterisation.

This methodology currently focuses on discrete design decisions, such as spawn rate or collision response. In some cases, we might additionally want high-level design parameters that impact many lower-level design decisions. Such parameters are common in generative machine-learning systems. For example, Microsoft's SongSmith musical accompaniment system both data-mines parameters from a corpus and makes a virtue of the hyperparameters required by machine-learning models by exposing them as well (Simon, Morris, and Basu 2008) – for example, Markov-model transition weight was made a parameter and dubbed the "jazz factor". Similar high-level parameters could be used in fluidic games for complex assets such as music, sound effects, and visuals, for which users often want broad stylistic control. We plan to add such high-level parameters to *Blasta*.

An alternative way to support high-level design-space navigation is to allow users to link together parameters with constraints. This is perhaps the primary feature of parametric design in architectural CAD tools (Motta and Zdrahal 1996; Woodbury 2010), which allow users to specify, for example, that a component must be twice as long as tall. The user can then resize the component without having to keep linked quantities in sync. Although designing a casual-creator interface for constraint editing would be challenging, giving users a way to accumulate constraints would allow them to navigate large parameter spaces more efficiently.

As a contribution to Computational Creativity, casual creators add an extra constraint on the design of parameter spaces by requiring that human users of creativity support tools must be able to navigate the parameter space and enjoy doing so. The methodology presented here helps with this tension between supporting breadth of creative expression and designing intuitive, fun interfaces. The main way it does so is by requiring the final parameterised design space to be built up *in tandem with* a casual-creator interface for navigating it. This process, dubbed Stage 3 above, ensures that the parameter space meshes nicely with the UI for navigating it. Theory-based constraints on interface design could also be enforced at this stage, such as limits on the cognitive concurrency of design actions (Kavakli and Gero 2002).

In future work, we plan to look at the Computational Creativity literature to identify further sources for parameter extraction. We note that the first three sources above have some relationship to the descriptive IDEA model (Colton, Charnley, and Pease 2011). The IDEA model describes a computationally creative system able to capture inspiring examples, and then fine-tuning to generalise from them, as the first two stages of developing a creative system; for us here, those are the first two sources of parameters for enabling human creativity in a design space. In addition, the iterative expansion and contraction of a design space bears some resemblance to work on design-space 'sculpting' (Smith and Mateas 2011).

When engineering software for autonomous creativity, many Computational Creativity researchers, including ourselves, have specified a parameterised space of artefacts and enabled software to intelligently search the space. Hence, in addition to usage for implementing creativity support tools, we believe the methodology presented here might have broader usage across Computational Creativity research. We have experimented somewhat, but not yet fully investigated how automated techniques could take advantage of fluidic game spaces to make interesting games. We also plan to explore the possibilities for automatic use of the methodology here, i.e., getting software to create at the meta-level by automatically specifying a parametric design space for creative systems, which we believe would represent a major step forward for Computational Creativity systems.

Acknowledgements

This work is funded by EC FP7 grant 621403 (ERA Chair: Games Research Opportunities). We would like to thank the anonymous reviewers for their helpful comments.

References

Browne, C., and Maire, F. 2010. Evolutionary game design. *IEEE Trans. Comp. Intelligence and AI in Games* 2(1).

Colton, S.; Nelson, M. J.; Saunders, R.; Powley, E. J.; Gaudl, S. E.; and Cook, M. 2016. Towards a computational reading of emergence in experimental game design. In *Proceedings of the 2nd ICCC Computational Creativity and Games Workshop*.

Colton, S.; Charnley, J.; and Pease, A. 2011. Computational creativity theory: The FACE and IDEA descriptive models. In *Proc. of the 2nd International Conference on Computational Creativity*.

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

Colton, S.; Pease, A.; and Ritchie, G. 2001. The effect of input knowledge on creativity. In *Proceedings of the ICCBR'01 Workshop on Creative Systems*.

Compton, K., and Mateas, M. 2015. Casual creators. In *Proc. of the 6th International Conference on Computational Creativity*.

Coons, S. A. 1967. Surfaces for computer-aided design of space forms. Technical Report TR-41, MIT.

Gaudl, S. E.; Nelson, M. J.; Colton, S.; et al. 2017. Exploring novel game spaces with fluidic games. In *Proceedings of the AISB Symposium on AI and Games*.

Gaudl, S. E.; Nelson, M. J.; Colton, S.; et al. 2018. Rapid game jams with fluidic games: A user study and design methodology. *Entertainment Computing* 27.

Kavakli, M., and Gero, J. S. 2002. The structure of concurrent cognitive actions: A case study on novice and expert designers. *Design Studies* 23(1).

Motta, E., and Zdrahal, Z. 1996. Parametric design problem solving. In *Proc. of the Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*.

Nelson, M. J.; Colton, S.; Powley, E. J.; et al. 2017a. Mixed-initiative approaches to on-device mobile game design. In *Proc. of the CHI 2017 Workshop on Mixed-Initiative Creative Interfaces*.

Nelson, M. J.; Gaudl, S. E.; Colton, S.; et al. 2017b. Fluidic games in cultural contexts. In *Proceedings of the 7th International Conference on Computational Creativity*.

Pérez Ferrer, B.; Colton, S.; Powley, E.; et al. 2016. Gamika: Art based game design. *Art/Games* 1.

Piegl, L. 1991. On NURBS: A survey. *IEEE Computer Graphics and Applications* 11(1).

Powley, E. J.; Nelson, M. J.; Gaudl, S. E.; et al. 2017. Wevva: Democratising game design. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Schaul, T. 2014. An extensible description language for video games. *IEEE Trans. Comp. Intelligence and AI in Games* 6(4).

Simon, I.; Morris, D.; and Basu, S. 2008. MySong: Automatic accompaniment generation for vocal melodies. In *Proc. CHI*.

Smith, A. M., and Mateas, M. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Trans. Comp. Intelligence and AI in Games* 3(3).

Stanley, K. O. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines* 8(2).

Woodbury, R. F. 2010. *Elements of Parametric Design*. Routledge.

Woodbury, R. F., and Burrow, A. L. 2006a. A typology of design space explorers. *AI EDAM* 20(2).

Woodbury, R. F., and Burrow, A. L. 2006b. Whither design space? *AI EDAM* 20(2).