Multi-Faceted Evolution Of Simple Arcade Games

Michael Cook and Simon Colton,
Computational Creativity Group, Department of Computing,
Imperial College, London, UK
ccg.doc.ic.ac.uk

***Abstract*— We present a system for generating complete game designs by evolving rulesets, character layouts and terrain maps in an orchestrated way. In contrast to existing approaches to generate such game components in isolation, our ANGELINA system develops game components in unison with an appreciation for their interrelatedness. We describe this multi-faceted evolutionary approach, and give some results from a first round of experimentation.**

I. Introduction

Computational intelligence research has done much to help the games industry attack the content generation problem in recent years. As consumers expect a greater breadth and depth of content in videogames, procedural generation techniques have been used to alleviate the burden on developers and give way to more dynamic and interesting games. This research has contributed work towards the generation of level designs [1], [2], [3], [4], [5], visual content [6], [7], controllers for game entities [8] and game rulesets [9], [10], [11]. Much of the material written about this work mentions a move towards *automated game design*, where an intelligent system is responsible for the overall development of a game. However, we argue that in its current state, such research is little more than an assistant in the game design process, not a designer itself.

For a system to take on the role of a game designer, it is not sufficient for it merely to generate all of a game's components by itself. It must also understand the relationship those components have with one another, and how a change in one component affects the final game as a whole. We have developed a system, called ANGELINA[1], that designs games from scratch by evolving the constituent game components with respect to one another. ANGELINA is composed of separate evolutionary processes which design a game's ruleset, its level and the layout of the game's non-player characters (NPCs), combined with an understanding of the dependencies between these processes that allow it to influence the end result accordingly. The types of games that ANGELINA can currently generate are limited to simple arcade games, as portrayed in figure 1, but we intend to extend this to more sophisticated games in the future.

The remainder of this paper is organised as follows: in section II, we introduce ANGELINA and outline the process by which it evolves games. In section III, we present some sample games generated by ANGELINA and evaluate the quality of the output, showing that ANGELINA is capable of working alone or assisted by a human, and demonstrating the importance of the system's awareness of the finished game. In section IV, we look at existing research in automated game design, and discuss how ANGELINA fits into this context. Finally, in section V, we discuss the future for ANGELINA and the possibilities for automated design beyond this project.

[1]A Novel Game-Evolving Labrat I've Named ANGELINA

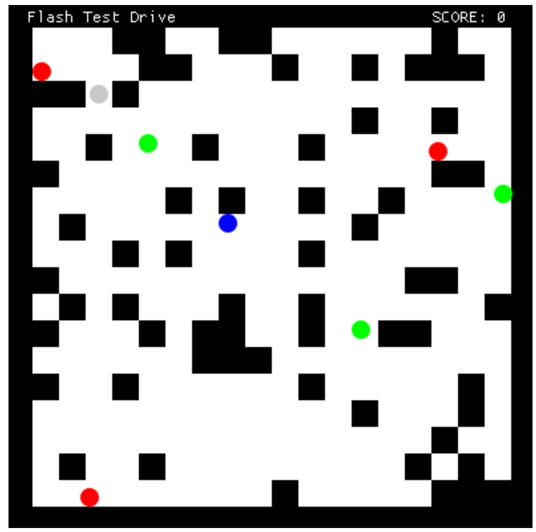


Fig. 1: A screenshot from a game designed by ANGELINA. The player is a light gray colour, and the non-player characters are coloured in red, blue and green. The score is displayed in the top right-hand corner.

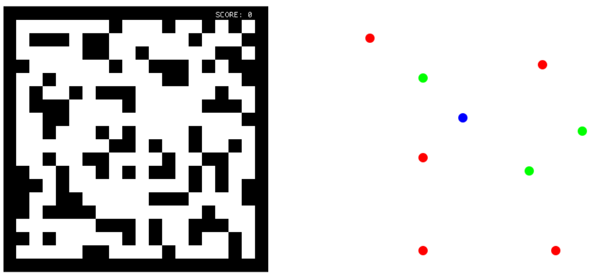
II. The ANGELINA System

A. The Anatomy of a Game

We represent a game as three flexible components - a *map* which defines passable and impassable areas in a two-dimensional grid of square tiles, where a tile is a small square of space in the game world; a *layout* which describes the arrangement of player- and non-player-characters on a similar two-dimensional grid; and a *ruleset* which describes the effects of collisions between types of game entities, as well as movement types for the NPCs, time limit and score limit for the game.

An example of a map and layout can be seen in the example game in Fig 2. The state space for these two components is the space of possible configurations of the 2D arrays representing them. For a map, black squares denote impassable terrain. NPCs can be freely placed anywhere

```
<scorelimit>44</scorelimit>
<timelimit>72</timelimit>
<redmovement>CLOCKWISE</redmovement>
<greenmovement>RANDLONG</greenmovement>
<bluemovement>RANDLONG</bluemovement>
<rules>
  <rule>PLAYER,RED,DEATH,DEATH,1</rule>
  <rule>GREEN,RED,TELEPORT,DEATH,0</rule>
  <rule>OBSTACLE,RED,NOTHING,TELEPORT,0</rule>
</rules>
```



(a) The map (b) The game layout

Fig. 2: A sample game produced by ANGELINA, with the map and layout shown visually in the subfigures, and the rules presented in XML format.

within the layout, but not on top of each other. The state space described by the rulesets is derived from those defined in [10]. In particular, circles of three different colours move around the grid defined by the map. The manner in which they move is chosen from a selection of presets based on existing arcade games, including: *random walks* which change direction at unpredictable intervals, and *wall hugging* behaviours, which only change direction when they encounter an obstacle.

When two objects collide, rules in the ruleset may fire. Rules have conditions, which are the two colours whose collisions they are triggered by (possibly including the player), and the effects applied to the two colliding objects. These effects are *death*, i.e., removing the object from play, and *teleport*, which moves the object to a randomly-chosen, non-obstructed point on the map. A rule may only apply an effect to one of the colliding objects. Each rule also has an associated score value which may be positive, negative or zero. The player's score is shown while they play the game.

These three game components can be evolved individually as if they were single artifacts, or with reference to each other, as described later. The combined game components are then converted into external formats for automatic compilation into executable games. ANGELINA is currently capable of outputting *Monkey* code[2], which can be compiled into a variety of target executables, including Flash, HTML5 and Objective-C code. The flexibility of these output formats will make it simple to distribute ANGELINA's games widely, which will help when we incorporate human-sourced playout data in future experiments.

[2]http://www.monkeycoder.co.nz/

B. Crossover and Mutation

In order to recombine the various elements of the games we implement a range of crossover styles, obtained through experimentation with each component type. For maps, we produce offspring by taking a binary approach to the two parents. For each pair of parent maps, we produce five children. One child is produced through binary AND, where a tile in the child map is an obstacle if it is an obstacle in both of its parents. We do the same for another child, through binary OR. The remaining children are produced through a scattered inheritance, where an arbitrary map tile in the child map map_{child} is defined as:

$$map_{child}[i][j] = \begin{cases} map_1[i][j] & \text{if } rnd() < 0.5 \\ map_2[i][j] & \text{if } rnd() \geq 0.5 \end{cases}$$

For NPC layouts, we use some similar approaches, employing binary operators AND and OR to produce children. We also transpose sets of coloured NPCs between the parents; for example, one child may be a copy of one of its parents, but have its set of blue NPCs switched out and replaced with the set of blue NPCs from its other parent. Which sets are crossed over is chosen randomly, and each pair of parent layouts generates ten children.

Ruleset evolution includes some single-parent mutation as well as dual-parent recombination. Single parents generate mutated offspring by choosing a random rule and randomly changing one of its parameters, such as the effect of the collision, or the coloured NPCs it is triggered by. Crossover works by swapping rules over between parents, so that the resulting child has a mixture of rules from its parents. For every eight children generated this way, we also include two randomly generated 'fresh' rulesets to avoid stagnation in the evolutionary search, which was noted in early experiments.

These evolutionary settings have been determined by some preliminary experimentation, but in future, we plan to perform systematic testing of the evolutionary parameters, to optimise the system

C. Fitness Functions For Game Components

1) Maps: Map designs are scored using two metrics which we have termed *fragmentation* and *domination*. A map's fragmentation score is equal to the number of *islands* present in the map, where an island is a set of blocks that are adjacent to one another and disconnected from both the wall and other islands. A map's domination score represents the number of tiles which *dominate* sections of the map. A tile is said to dominate two other tiles if all paths between those two tiles must pass through the dominated tile. A similar definition can be found in [5]. In the vocabulary of game design, a dominating tile represents a doorway or a corridor; a tile separating two regions of the map.

Calculating which tiles dominate others is costly to do, given that it must be performed on every tile in every map in the population at every generation. We check if a tile is dominating by calculating how many tiles are reachable from that tile in the map. We call this set of reachable tiles a *flood plain*. For a tile, t, the flood plain is defined as:

$$fldplain(t, map) = |\{t_2 \text{ such that } rchable(t, t_2, map)\}|$$

where

$$rchable(p, q, m) \iff \exists \text{a path from p to q in map m}$$

We then modify the map by blocking out t and making it inaccessible. We then perform the same calculation for a neighbour of t, t_{nb}, in the modified map map_{br}.

$$map_{br} = map[t = \texttt{obstacle}]$$

Where $map[t = \texttt{obstacle}]$ represents a remapping of map such that tile t is now inaccessible. Therefore, tile t is a dominant tile for map if the modified floodplain is more than one tile smaller than the unmodified floodplain.

$$floodplain(t, map) < floodplain(t_{nb}, map_{br}) + 1$$

Since we have only blocked off one tile in map_{br}, if the floodplain is more than one tile smaller then blocking t must also have blocked access to other tiles. In other words, t dominates at least one tile. In general, fragmentation controls the number of obstacles in a map, while domination controls the openness of the play areas. These map-generation metrics are simple, but include a range of arcade-style archetype maps varying from open arena-based games, such as *Pong* or variants of *Snake* with minimal obstacles, to the more labyrinthine maps with maze-like properties such as those used by *Pac-Man* or *Sokoban*.

2) Rulesets: We score rulesets using two distinct methods. First, we run a series of checks to filter out the most unplayable or useless rulesets. We do this by examining the rulesets directly and checking for pathological combinations of rules. For instance, a ruleset is unplayable if the only way of gaining score is for the player to die; similarly, a game should never specify a rule for a NPC type if that type is entirely absent in the game layout, and the player cannot collide with themselves, etc.

In deciding on what form these checks should take, we took inspiration from McGonigal's definition [12] of a game's components as a *goal*, a set of *obstacles* and a *feedback mechanism*. In particular, we ensure the games are directed towards a goal using score gain and provide an obstacle to the player through the loss of score or death. These games shrink the state space for rulesets, whilst retaining a core space of 'interesting' rulesets that can be explored through playouts. We also cull the lowest-scoring half of the population and take the remainder through to a series of playouts. In total we simulate eight playouts of the game, recording the score and whether or not the player died. A playout begins by fixing a behaviour for the player-character (since the playout is automated, and therefore no human player is present) by applying one of the available NPC behaviours to them, and optionally adding in certain constraints to test different aspects of the game. The types of playout are:

- *Empty*, where the player character is removed from the game and the game is executed until the time limit. This gives useful information on the game's natural entropy. For a game like *Pong*, this would result in a large negative score, while a game like *Pacman* would return a zero score.
- *Static*, where the player is included in the game but does not move. This might be considered a step up from the Empty playout. This playout represents a worst-case scenario for player behaviour.
- *Random Walker*, where the player mimics the NPC behaviour type of random walk. The player moves through the environment, turning at randomised intervals.
- *Pre-Collision*, which are identical to Random Walk playouts, except that the player is collided with all NPCs of one or more types before the play begins. These playouts are useful to compare to Random Walks as they show what the extremes of player involvement are. A game like *Pacman* yields very useful information from these playouts, as they include a perfect run (colliding with all the pills before the game begins completes the level) and a worst-case (colliding with the ghosts before the game begins, which results in the game over screen).
- *Long Play*, which are also identical to Random Walk playouts, but they are not limited by time.

This sort of information about the role of the player in the current ruleset is used to filter out broken rulesets by detecting games which the player has little or no effect on, or that are too easily won. This is done in a second round of evaluation, where each ruleset is examined in turn and the score data for their playouts analysed. We expect some or no progression between the Empty, Static and Random Walk playouts, as they represent increasing player involvement and our definition of a game is strongly geared towards the player being the factor affecting the score. It is also useful to compare pre-collision data, as if all playouts of this type tend towards a positive or negative score it can mean that a game's rulesets are either too difficult or too hard.

Note that score and time limits are not measured directly. Because they affect the length of playouts, they indirectly affect the fitness of the games overall - a time limit that is too short may prevent the player from making progress, while one that is too long may give the player too much time to gain score. Score and time limits are modified during crossover and mutation as part of the evolutionary process.

3) Layouts: Layouts are scored using sparseness and volume as metrics. This space is defined thus: a layout is sparse if the average distance between two NPCs is high. For a layout with n items in it (of any colour) the sparseness of a layout is measured as:

$$2 \times (\sum_{i=1}^{n} \sum_{j=1}^{n} dist(i,j)) \times \frac{1}{(n \times n-1)}$$

Assuming the existence of a *dist* function, measuring the Euclidean distance between two items. Volume is the measure of how many entities NPCs there are on-screen. For a map m, its volume is defined as:

$$vol(m) = |reds(m)| + |blues(m)| + |greens(m)|$$

Where *colour(m)* is the list of items in map m of that colour. The layouts also compare themselves with current map instantiations to ensure that they are producing legal, playable layouts - that is, layouts which do not place NPCs over obstacles such as walls. In this way, the layouts and the maps shape each others' development.

D. Multi-Faceted Evolution

For creative domains such as videogame design, we have developed a new framework for artefact generation that uses individual evolutionary strands in a compositional manner, to share information during the evolutionary process and use the fitness of individual components to increase the overall fitness of the finished artefact.

The basic components of ANGELINA are the evolutionary strands that define what the system has control over. These are currently the design of rulesets, maps and character layouts, but in future we will extrapolate this to components that design graphics packs, control systems, quests or other elements of design. These strands are considered to be individual, but they are placed within a larger system that allows them to interact and influence other strands.

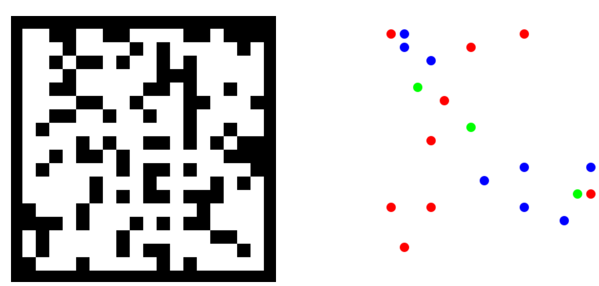
The system is represented by a central game object. The game is composed of three elements; a ruleset, a map and a character layout. These elements are initially represented by appropriately-chosen *null phenotypes*, i.e., templates that represent an empty instantiation of that object. For a ruleset, it is an empty set of rules and a static movement rule for all NPCs. For a map, it is a square arena with no obstacles inside it. For a layout, it is simply the empty layout.

Each evolutionary strand is self-contained; that is, it has its own fitness functions and understands how to create new generations of whatever it is trying to evolve. During evolution, however, fitness functions may request information from the central game object. We call this a *pull*, as the strand is retrieving information from the main object. The information that the fitness functions request may be very simple, such as the number of red NPCs in a given layout, or it may require retrieving a larger component such as the game's map, to check for legality of object placements. Initially, the strands will pull the null phenotypes from the game object. They exist to ensure there is always some information to measure fitness against, even if the information retrieved is merely a placeholder.

At regular intervals, ANGELINA requests that evolutionary strands perform a *push* to update the state of the central game artefact. Each strand copies its most fit phenotype into the game object so that information being pulled is more accurate for measuring fitness against. This is currently done after every generation, in order to keep interaction between the strands high and the information in the main object as up-to-date as possible. However, the system is flexible, and the strands could update less frequently to give more autonomy to each individual component's evolution.

The design of this system has some advantages. From a practical standpoint, it is trivial to add further components to the system without affecting the current evolution. Strands push and pull information from a central object, so adding a component that generates music or graphics will not affect existing systems, but can take advantage of their pushed information to take part in the multi-faceted process. Moreover, while the process described above has the strands operating in parallel, this is not compulsory. If one component is already designed and finished, it can be fixed in the central game component and never updated, providing accurate fitness data to other strands from the first generation. In this way, we can provide partial designs to ANGELINA and ask it to complete them for us. For instance, ANGELINA can be asked to design levels for a human-designed ruleset, by specifying the ruleset from the beginning and asking it to provide levels and layouts that exhibit high fitness alongside that design.


```
<scorelimit>44</scorelimit>
<timelimit>72</timelimit>
<redmovement>CLOCKWISE</redmovement>
<greenmovement>RANDLONG</greenmovement>
<bluemovement>RANDLONG</bluemovement>
<rules>
  <rule>PLAYER,RED,DEATH,DEATH,1</rule>
  <rule>GREEN,BLUE,TELEPORT,DEATH,0</rule>
  <rule>OBSTACLE,RED,NOTHING,TELEPORT,0</rule>
</rules>
```

(a) The game map (b) The game layout

Fig. 3: A game designed by ANGELINA running independently, where evolutionary processes are not able to inspect other components whilst evolving. The components, whilst individually fit, do not relate to each other and the resulting game is unplayable.

III. Experimentation and Results

We conducted several experiments to test the limits of ANGELINA's ability to design games both assisted and alone, and in this section we introduce each of these and examine the results arising.

A. Density Of The Game Design Space

Before we present designs produced by ANGELINA, we will give an idea of the nature of the state space that our system is exploring. We produced an initial population of 100 games, and examined each for their quality. If a game was playable, with a consistent map and layout and a goal-oriented ruleset, we classified them as Playable. If the games were not playable, they were classified with further detail on what factors were causing the lack of playability. The results of the classification exercise are given in Figure 4.

Playable	3
Illegal Map/Layout	100
Broken Rules	74
No Goal	69
No Obstacles	54

Fig. 4: Classification of 100 randomly-chosen games from the state space.

Of the 100 games, only three were playable. We define playable through the same judgements we apply in our fitness functions, along with further evaluation through human playouts. Each game was compiled and played by hand, and games which had clear goals and obstacles, and in which the player had some purpose or way to affect the system, were deemed playable. All of the games, however, had conflicting layouts and maps. Every game generated had NPC elements that were either embedded into walls or cut off from the rest of the map. We allowed a game to be declared playable if the player character was legally placed, although other game items may be placed illegally. As long as the goal remained achievable - that is, one or more of the goal items are not placed illegally - then the game as a whole is technically playable.

74 of the games examined had broken rules. These included rules that can never fire, such as a rule for a collision between two wall entities, or a rule that makes no sense given the current game design, such as a rule for green NPCs if the layout does not contain any. Further to this, 69 of the games had no way for the player to gain score, or avoid losing score, and 54 of them had no challenge for the player, either through death or the loss of score. Often, failing one of these checks was enough to stop a game being playable. However, most games failed three or more checks.

B. Independent Game Design

To investigate whether the whole is more than a sum of its parts, we first present games developed by ANGELINA with the system running without interaction or awareness between the individual evolutionary strands. What results are games that are composed of uniquely fit subcomponents, but that are not considered fit as overall games. Figure 3 shows a sample game from these runs. The game's rules, roughly, challenge the player to reach a red object. Touching the red object kills the player, and is the only way to gain score.

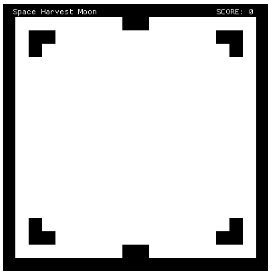
We see that each component complies with its own fitness constraints. For instance, the map in figure 3 exhibits a high domination score, creating a maze-like layout. The rulesets also conform to the loose common-sense fitness function devised for them. Despite this, the game in Figure 3 is unplayable, because, the map and layout conflict with each other by overlaying NPCs with wall tiles. This means those characters are unable to move or may be inaccessible to the player. The ruleset also includes two problems in particular - one of the rules, for collisions between `green` and `blue`, will never be used because the layout does not include any green NPCs. Another rule, the only one which allows the player to gain score, causes the player to die. This means that there is no way for the player to achieve the game's goal. Recalling our definition of a game as a goal, a series of obstacles, and a feedback mechanism, a broken ruleset such as this provides incorrect obstacles, and does not have any path to the goal for the player to pursue.

C. Flexibility And Responsiveness

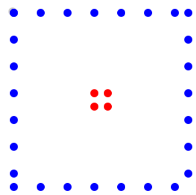
Figure 5 shows a game designed with some human input. We presented ANGELINA with a human-designed map and layout and asked it to derive a ruleset. The top images show a fairly open map with islands of obstacles that impede movement. The game ANGELINA designed exhibits rules relating to the placement of Blue non-player characters, fixes goals and obstacles for the player, and chooses movement behaviours for the various non-player character types. The player must avoid the moving blue NPCs and touch the walls to score.

We then changed the map and layout to that shown the lower images in Figure 5. This map and layout is less open than the previous one with a similar NPC arrangement. The fittest ruleset derived is much different to the previous example; the player must stay around blue objects to gain score, and avoid the fast-moving and unpredictable red objects. These objectives are more about awareness and navigation than agility, which fits with a more confined space and unpredictable opponents, since the red NPCs now teleport erratically. The module responsible for ruleset design knows, through playouts and analysis of NPC populations, that the space of 'good' games has changed because of the new level design, and the rulesets produced change appropriately. In particular, the second game designed does not use collisions with obstacles as a rule, because the higher number of obstacles would make the game too difficult (or too easy, depending on the effect of colliding with the obstacles). Instead, the proposed game takes advantage of the more maze-like approach by developing a game with an emphasis on chase and evasion. ANGELINA's responsiveness to change in one element here has resulted in a dramatically different

```
<scorelimit>55</scorelimit>
<timelimit>91</timelimit>
<redmovement>CLOCKWISE</redmovement>
<greenmovement>RANDSHORT</greenmovement>
<bluemovement>RANDLONG</bluemovement>
<rules>
  <rule>PLAYER,OBSTACLE,TELEPORT,NOTHING,1</rule>
  <rule>BLUE,PLAYER,TELEPORT,TELEPORT,-1</rule>
</rules>
```

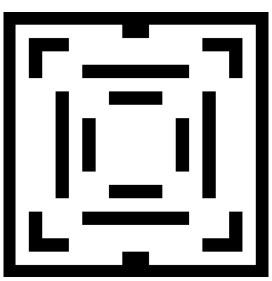


(a) The game map

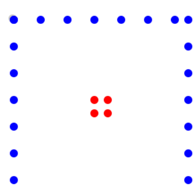


(b) The game layout

```
<scorelimit>89</scorelimit>
<timelimit>21</timelimit>
<redmovement>RANDLONG</redmovement>
<greenmovement>CLOCKWISE</greenmovement>
<bluemovement>CLOCKWISE</bluemovement>
<rules>
  <rule>PLAYER,BLUE,NOTHING,NOTHING,1</rule>
  <rule>PLAYER,RED,DEATH,DEATH,0</rule>
  <rule>PLAYER,NONE,DEATH,TELEPORT,-1</rule>
  <rule>OBSTACLE,RED,NOTHING,TELEPORT,-1</rule>
</rules>
```



(c) The game map



(d) The game layout

Fig. 5: Two games designed by ANGELINA, showing a response to the change in level design.

result that shows what we might describe as the beginnings of understanding being exhibited by the software.

D. *Free Design*

Figure 6 shows two games designed by ANGELINA without constraints. The first game is a close relative of *Pacman* - the player is encouraged to collide with blue objects (which are destroyed, and give the player score) and avoid red objects (contact with them causes death). The map designed is not as symmetrical as Pacman's, but is open enough for the player to manoeuvre around. This is a common arcade archetype that ANGELINA discovered within the state space independently.

The second game, shown in figure 6, is an arcade game equivalent of the 'steady hand game', where the player must avoid touching walls and pick up all NPCs of a certain colour. This map is designed as more confined, with less free space to manoeuvre. Its layout is more spread out and contains only one type of NPC. These two games show how ANGELINA explores and refines the search space open to it, and creates games that make sense *as a whole*, with the various subcomponents complementing each other. The Steady Hand Game is an interesting contrast to the Pacman clone, as it is not a rediscovery of a famous arcade game, yet is recognisable as a coherent game.

IV. Related Work

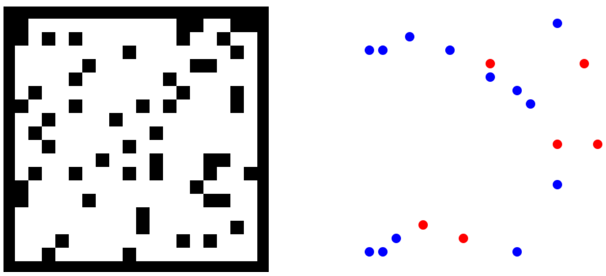
A. *Automated Game Design*

There is relatively little research which considers the problem of automated game generation as being any more than a series of linear content generation problems. Togelius et al. [10] explored the possibility of ruleset generation, using neural networks to evaluate their rulesets according to a learning-based model of fun similar to that described in *A Theory Of Fun* [13]. The rulesets generated by this system define two-dimensional arcade games where coloured shapes are controlled around a map. Although the work made some useful conclusions about its choice of model for game quality, the scope of the project was limited to ruleset design and did not attempt to generate other game components, such as levels or control schemes. However, Togelius' use of an independent playout module to assess rulesets can be seen as a move towards a more aware game designer, as it allowed the system to reflect on its own work as it progressed.

Smith and Mateas discuss automated game design in their presentation of *Variations Forever* [9], a research project into the procedural generation of 'minigames'. The authors use *answer set programming* ([14], [15]), a powerful form of logic programming, to represent a large state space of possible rulesets, and then constrain the space to explore interesting subspaces of rules. This work only generated sets of rules, but the work is particularly notable because of the discussion of automated game design found within. In particular, the authors propose further work to develop their system with an automated playtester that could evaluate rulesets and use a knowledge base of player behaviour to suggest alterations to game designs. However, the discussion does not extend beyond the realm of designing simple rulesets, in a 'generate and test' paradigm. In contrast, we aim to produce a system capable of understanding the connectedness of *multiple* game components, which means moving beyond rulesets alone.

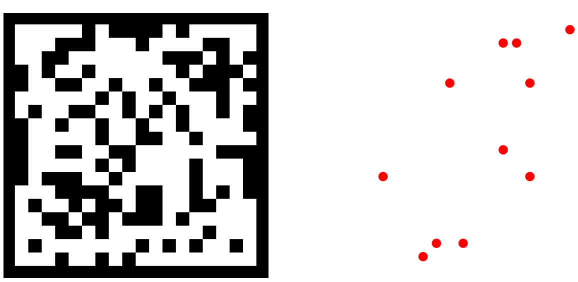
Browne and Maire [16] explore the problem of automatic game design for the world of boardgames, a medium with

```
<scorelimit>56</scorelimit>
<timelimit>28</timelimit>
<redmovement>CLOCKWISE</redmovement>
<greenmovement>STATIC</greenmovement>
<bluemovement>RANDLONG</bluemovement>
<rules>
  <rule>PLAYER,RED,DEATH,DEATH,-1</rule>
  <rule>PLAYER,BLUE,NOTHING,DEATH,1</rule>
</rules>
```



(a) The game map (b) The game layout

```
<scorelimit>24</scorelimit>
<timelimit>96</timelimit>
<redmovement>STATIC</redmovement>
<greenmovement>RANDLONG</greenmovement>
<bluemovement>RANDSHORT</bluemovement>
<rules>
  <rule>OBSTACLE,PLAYER,NOTHING,DEATH,0</rule>
  <rule>PLAYER,RED,NOTHING,TELEPORT,1</rule>
  <rule>RED,OBSTACLE,TELEPORT,NOTHING,0</rule>
</rules>
```



(c) The game map (d) The game layout

Fig. 6: Top: A Pacman-style game where the player must pick up green objects while avoiding red NPCs. Bottom: A 'steady-hand' game designed in an unconstrained mode. This game was not in the archetypal arcade game design space, but is an interesting invention by the system.

many analogues to arcade videogames. The work here closely tackles the problem of automated game design by developing not only rulesets, but board shapes and starting layouts also. The work led to the design of many boardgames, some of which achieved commercial success[3]. Although similar in structure, videogames have many components that boardgames lack. Most notably, the majority of videogames are played in real-time, which makes the task of playing out, and therefore evaluating, candidate games much more difficult due to a larger state space of game executions. Similarly, videogames typically derive their challenge from the presence of computer-controlled non-player characters. Assessing the challenge represented by these characters is harder than considering the challenge represented by binary rules, which in turn makes games harder to evaluate.

B. Evolutionary Computation

Evolutionary computation has been used in many areas of content design for videogames both in research and so-called 'real-world' videogame projects[4]. Often, these projects employ a straightforward, linear Darwinian evolution method where a population of candidate artifacts are evaluated against a single fitness function and then high-scoring artifacts are crossed over to generate a new, fitter population.

Two relevant techniques which have emerged out of research into evolutionary computation are *multi-objective evolution* (MOE) and *co-evolution*. MOE differs from standard, linear evolution by attempting to optimise more than one fitness function for a single evolutionary strand. These fitness functions typically conflict in some way, and forms of Pareto optimality are often employed to find maximally-satisfying artifacts through evolutionary search. An example of such a system, along with a succinct explanation of its operation, can be found in [17].

Co-evolution refers to a naturally-occuring evolutionary system where two organisms evolve in response to changes in each other. This is often found in relationships between insects and plants or larger animals, comparable to symbiotic relationships. Programmatically, co-evolution has been used to develop two or more artifacts together with one another [18], [19]. Two standard evolutionary strands work alone, and in order to assess their fitness, they are composed into a single artifact and evaluated.

Standard evolution and other genetic programming techniques are well-applied to the world of content generation ([3], [8]). However, these techniques work largely in isolation and have no way of interacting with larger systems, and there is no sense of a guided design process.

Co-evolution is the technique closest to ANGELINA's underlying methods, but while co-evolution is primarily concerned with change in one object *causing* change in another, we are more interested in the evolutionary pressure applied by changes in multiple components. To continue

[3]The game Yavalath was designed by Browne's Ludi system and went on to be published by independent boardgame manufacturer Nestor Games.

[4]The canonical example here is possibly Elite or, more recently, Dwarf Fortress - http://www.bay12games.com/dwarves/

the biological analogy, while co-evolution is concerned with close relationships between two species, we are more interested in the ecosystem as a whole, and how the evolution of one strand can change the objectives of other strands.

V. Further Work

While the current space of possible games is broadly inherited from [10], it is not expressive enough to further test and demonstrate the capabilities of an autonomous game designer. The core of ANGELINA will be expanded to include a modular rule system and a search space which contains a larger variety of game types. Many arcade games are developed using a very small cache of rules. For instance, *Frogger* is a visually distinct game from *Pacman*, but only includes one new rule (a win-condition for being within a defined area of the map). A larger state space would improve ANGELINA's chances of finding new game types. The inclusion of more unusual rules, such as the ability for NPCs to change their type from one colour to another, would hopefully allow exploration of a less well-known state space and uncover arcade designs that are not so well known, but enjoyable to play.

We believe that our approach to the evolutionary design of multi-faceted artefacts, through self-aware evolutionary strands that interact with each other, is applicable to many complex design tasks besides the design of videogames. We are interested in generalising the techniques we have begun to explore here, and developing a framework for creative systems which concern themselves with creating complex, interconnected artefacts. We are interested in applying these techniques to tasks such as the editorial design of magazines and newspapers, where space restrictions and aesthetic considerations conflict in interesting ways, and the construction of narratives, where there is conflict between the strands of narrative, the histories of characters, and the simulation of the world in which the plot is set. We hope that multi-faceted evolution's approach to complex artefact generation will provide a new way of considering these problems.

One advantage of generalising such a framework would be the simpler construction of tools that use evolutionary techniques to assist humans in creative tasks. As we demonstrated in section 4, ANGELINA can develop entire games from scratch, but can also work from partially-specified starting points as well. We can envisage a computer-aided design tool that uses multi-faceted evolution to fill in the blanks in a creative design that a person has begun, be it a videogame design or otherwise.

VI. Conclusions

We have presented a new approach to evolving arcade-style game designs through evolution, by maintaining individual evolutionary processes that can inspect and affect the fitness functions of other strands as they evolve. We have demonstrated this approach through the ANGELINA system, a game designer that produces simple arcade games from scratch by combining rulesets, maps and object layouts that it has designed itself. We demonstrated how this forms a single process, through multi-faceted evolution.

We showed that individually fit game components cannot be blindly composed together, and showed how ANGELINA's method for evolving these components in tandem with one another is effective and flexible. We discussed examples of these games and showed how ANGELINA was able to rediscover old arcade game archetypes, as well as reveal game designs that were novel and unexpected. Finally, we outlined our plans for a generalisation of the techniques in this paper to design better games, and other creative artifacts.

Some of the games compiled by ANGELINA have been put online, including some outlined in this paper[5]. We hand selected these from a dozen runs of ANGELINA as being fun or offering an interesting challenge. We hope to expand these in the future, as ANGELINA's capabilities broaden.

References

[1] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. HagelbŁck, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[2] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *EvoGAMES*, 2010.

[3] D. Ashlock, "Automatic generation of game elements via evolution," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[4] G. Smith, M. Treanor, J. Whitehead, and M. Mateas, "Rhythm-based level generation for 2d platformers," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, 2009.

[5] D. Ashlock, C. Lee, and C. McGuinness, "Search based procedural generation of maze-like levels," in *IEEE Transactions on Computational Intelligence and AI in Games*, 2011.

[6] A. Martin, A. Lim, S. Colton, and C. Browne, "Evolving 3d buildings for the prototype video game subversion," in *EvoGAMES*, 2010.

[7] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proceedings of the 5th international conference on Computational Intelligence and Games*, ser. CIG'09, 2009.

[8] C. u Lim, R. Baumgarten, and S. Colton, "Evolving behaviour trees for the commercial game defcon," in *EvoGAMES*, 2010.

[9] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2010.

[10] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2008.

[11] M. J. Nelson and M. Mateas, "Towards automated game design," in *Artificial Intelligence and Human-Oriented Computing*. Springer, 2007, pp. 626–637.

[12] J. McGonigal, *Reality is Broken*. Jonathan Cape, 2010.

[13] R. Koster, *A Theory Of Fun For Game Design*. Paraglyph Press, 2004.

[14] Y. Babovich, E. Erdem, and V. Lifschitz, "Fages' theorem and answer set programming," in *In Proc. NMR-2000*, 2000.

[15] F. Fages, "Consistency of clark's completion and existence of stable models," 1994.

[16] C. Browne and F. Maire, "Evolutionary game design," in *IEEE Transactions on Computational Intelligence in AI and Games*, 2010.

[17] P. Saetrom and M. Hetland, "Multiobjective evolution of temporal rules," in *8th Scandinavian Conference on Artificial Intelligence*, 2003.

[18] C. Nachenberg, "Computer virus-antivirus coevolution," *Commun. ACM*, vol. 40, January 1997.

[19] J. R. Koza, "Evolution and co-evolution of computer programs to control independently-acting agents," in *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*. Cambridge, MA, USA: MIT Press, 1990.

[5] http://www.gamesbyangelina.org