RESCALABLE, REPLAYABLE MAPS GENERATED WITH EVOLVED CELLULAR AUTOMATA*

Daniel Ashlock, Laura Bickley

Department of Mathematics and Statistics, University of Guelph
50 Stone Road East, Guelph, Ontario N1G 2W1, Canada

(Received February 29, 2016)

A fashion-based cellular automata is one whose updating rule follows the form of an ecological competition model. The rule for the automata is specified by a square matrix with entries quantifying the influence each state has on each other when they both occur within a neighborhood. Because they preserve areas containing a single cell state, these rules are well able to specify automata that rapidly transform a random initial condition into a map appearing as a collection of caverns. Because the automata acts in a purely local fashion, it is valuable for generating collections of maps with similar look-and-feel, but different details, enabling automatic content generation and replayability in video games. This study extends an earlier study, examining new fitness functions and studying reusability, scalability, and the impact of parameter tuning for this type of cellular automata for automatically designing level maps. A representation for evolutionary computation is morphable if convex combinations of instances of the representation are instances of the representation. The fashion-based rules, being specified by real values matrices, are morphable. The ability to produce new, more complex maps by exploiting morphability is also explored.

DOI:10.5506/APhysPolBSupp.9.13

1. Introduction

This study builds on earlier work using a cellular automata to design level maps resembling a network of caverns [1] as well as an initial parameter study on the type of rules used in this study [2]. These level maps are used in video games. This study examines additional fitness functions for the system for automatic content generation (ACG) developed in the earlier

* Presented at the Summer Solstice 2015 International Conference on Discrete Models of Complex Systems, Toronto, Ontario, Canada, June 17–19, 2015.

studies as well as examining the *morphability* of the evolved rules. Morphing consists of transforming one rule into another using a continuous transformation parametrized by the spatial coordinates of the map. A requirement for morphing is that convex combinations of CA rules also be CA rules, a property that the evolvable representation used in this study has.

Cellular automata instantiate discrete models of computation. A cellular automaton has three components:

1. A collection of cells divided into neighborhoods of each cell;
2. A set of states that cells can take on;
3. A rule that maps the set of possible cell states of a neighborhood to a new state for the cell with which the neighborhood is associated.

In practice, CA are a type of discrete dynamical systems that exhibit self-organizing behavior. When a cell population is updated according to local transition rules, it can form complex patterns. The updating may be synchronous, as it is in this study, or asynchronous. CA are potentially valuable models for complex natural systems that contain large numbers of identical components experiencing local interactions [3, 4]. This paper applies them to create cavern-like level maps for games. Examples of the technique are shown in figure 1. The automata in this study are called *fashion-based* cellular automata because the updating rule may be thought of as following the current fashion within each neighborhood.

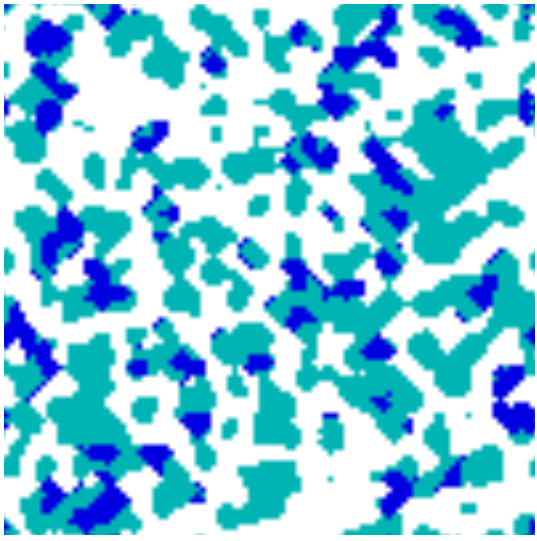


Fig. 1. An example of an evolved cavern-like level map.

2. Fashion-based cellular automata

We begin by defining fashion-based cellular automata. The automata in this study have a cell space consisting of $N \times N$ or cells for $N \in \{51, 101, 201\}$. The smaller size is used when testing variations of the evolutionary algorithm, the larger are used to generate maps for use of display. The left and right and the top and bottom sides of the cell space are considered to be adjacent making the cell space toroidal. The use of a toroidal grid of cells means that the resulting pictures will tile correctly without edge artifacts if tiling is required. The neighborhoods used by the automata are the von Neumann neighborhoods with four neighbors of each cell. The number of cell states used is $n = 6$, a number chosen in an earlier study [2]. The minimum possible number of useful states is 2, a larger number of states permits a larger and more flexible set of rules. Preliminary experimentation showed that numbers of states beyond six did not yield materially different results. The automata rule is parametrized by an $n \times n$ real matrix M, indexed by cell states, with entry $M_{i,j}$ giving the score a cell with state i gets if it has a neighbor in state j.

The automata is updated synchronously. The updating rule computes the score of each cell, based on its state and the states of its neighbors. The pairwise score, according to the entries of the matrix M, for a state and each of its neighbors is summed to compute each cell's score. Each cell then either stays in the same state, if its score is at least as high as those of its neighbors, or adopts the state of its highest scoring neighbor. This is thought of as "following the fashion" of the neighborhood. Fashion-based automata leave homogeneous regions homogeneous, a property that makes the space of rules rich with automata that generate cavern-like levels.

When rendering the automata as pictures, the states are translated into colors as follows; 0 white, 1 blue, 2 green, 3 cyan, 4 yellow, 5 magenta.

2.1. Representation

With $n = 6$ cell states, the rule is given by a 6×6 real matrix that is specified by 36 real parameters. This is encoded by a vector of 36 real numbers in the interval [0, 2] obtained by concatenating the rows of the matrix. Since scoring is relative, the range of values is not a critical parameter. The evolutionary algorithm uses two-point crossover, the default for our evolution software. Point mutation is performed by modifying one of the numbers by adding a uniformly distributed random variable in the interval $[-\epsilon, \epsilon]$ with $\epsilon = 0.1$. If a value leaves the interval [0,2] as the result of mutation, a new value is generated uniformly at random in the range [0, 2]. As the entries of the vector are the rows of the matrix, contiguous groups of 6 entries specify the scores one cell state obtains when matched against others.

In order to evolve cavern-like maps, cell state zero is interpreted as empty space and all other cell states are interpreted as full. The presence of multiple "full" cell types leaves open the possibility of evolving maps with complex terrain as in [5].

2.2. The evolutionary algorithm

Those unfamiliar with evolutionary algorithms may find a useful reference in [6]. The base evolutionary algorithm used to evolve rules operates on a population of 360 rules with size seven tournament selection as the model of evolution. This model of evolution incrementally updates the population by selecting a *tournament* from the population, sorting it by fitness, and then replacing the two least fit members of the tournament with copies of the most fit. These copies are subjected to crossover and mutation and the fitness of the resulting rules assessed. The algorithm is run for 10,000 such population updatings, called *mating events*, with summary statistics saved every 100 mating events. Mutation consists of between 1 and MNM point mutations with the number of point mutations selected uniformly at random. The symbol stands for *maximum number of mutations*. The default value used is MNM = 7. A parameter study varies the population size, number of mating events used, the MNM parameter, and the tournament size from the values used in the base algorithm is performed.

2.3. Fitness function

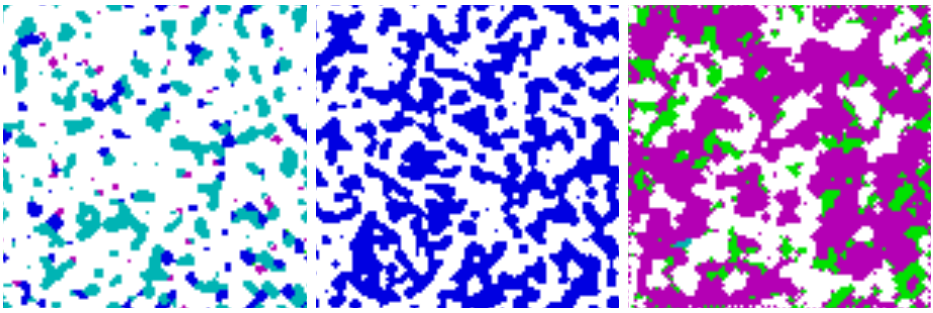
The fitness function used in the first study to judge the quality of an automata rule M is

$$f(M) = \frac{Z}{1 + |1 - 2U|}\,, \tag{1}$$

where Z is the number of cells in state zero in the connected empty component of the map including the middle cell of the map and U is the fraction of empty cells. If the center cell is not in state zero, this function yields a fitness of zero. This function rewards a large connected network of open spaces and a map that is 50% full. We will briefly examine the impact of changing this function to

$$f(M) = \frac{Z}{1 + \left|1 - \frac{U}{\alpha}\right|}\,, \tag{2}$$

where α is the desired fraction of full states, to change the density of the resulting map. Note that low values of α make it easy to have a large connected component of empty space, and so yield less challenging fitness functions. Using the software from [2] with different values of alpha, it is possible to obtain maps like those shown in figure 2. These are minor new results using small modifications of the original fitness function.



$\alpha = 0.25$ $\alpha = 0.5$ $\alpha = 0.75$

Fig. 2. From left to right these maps were generated by rules that had target densities of full cells of $\alpha = 0.25$, 0.5, and 0.75.

This study examines a fitness function, new for cellular automata-based maps, shown to be useful for decomposing the level map design problem [7]. This fitness function requires that openings to the cavern system be present in a manner that permits the evolution of *tiles* that can be linked to form larger maps. For this study, an opening in the middle of each side of the map is required and it is required that there be an open passage joining each pair of openings. If either of these constraints are violated, the automata rule receives a fiat fitness of zero. Otherwise the fitness of the rule is the average of the six distances between pairs of openings, computed with simple dynamic programming.

3. The parameter study

A total of 161 experiments were performed in which various parameters were modified to check their impact on fitness. These parameters were tournament size, number of mating events, population size, and MNM. The population size was changed in two batches. For the first batch of population size experiments, population size was set to 60, 360, 370, 400, 500, 1000, 2000, 3000, 4000, 5000, and 10000. Then, for each of these values, the MNM value was changed from 1 to 12, resulting in 120 experiments being run. For the second batch of population size experiments, population size was assigned values of 50, 100, 200, and 300. Then, for each of these values, the MNM value was set to 1, 4, 7, 10 and 12 resulting in 20 experiments being run.

For the next batch of experiments, tournament size and mating events were changed. For tournament size, the values tested were 5, 7, 8, 10, 15, 20, 25 and 30 resulting in 8 experiments being run. For the mating events, the values that were tested were 5000, 10000, 11000, 12000, 13000, 14000, 15000, 20000, 30000, 40000, 50000, 100000, and 1000000 resulting in 13 experiments being run. A partial visualization of these results is given in figure 3.

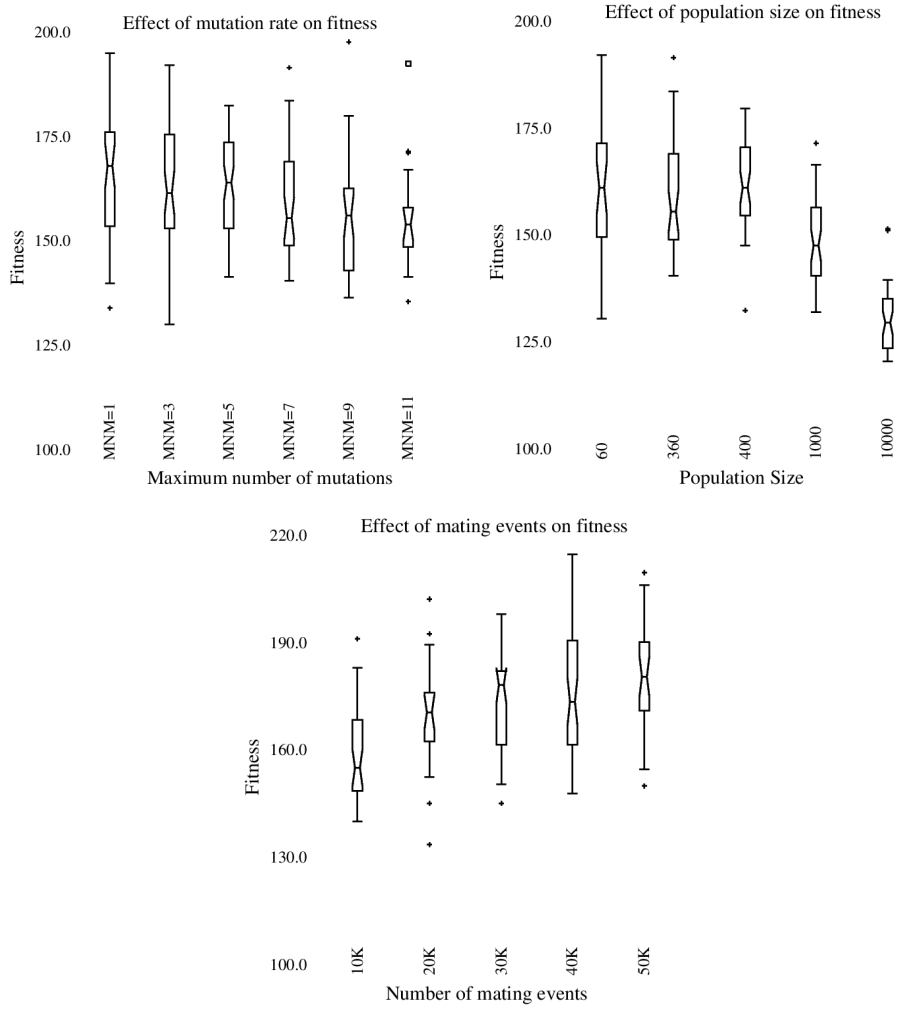


Fig. 3. Shown are inflected box plots for the impact of mutation rate (top left), population size (top right), and duration of evolution (bottom) on the final fitness of the population.

The parameter study shows that a small population size and a low mutation rate are both favored. The provision of additional mating events, extending the time of evolution, had a small beneficial effect that seems to top out at 30,000 mating events. These results will inform future experiments.

4. Reusability and rescalability

The purely local character of the updating of cellular automata means that changing the initial conditions can yield similar but distinct states of the automata. Figure 4 shows how this property can be exploited. An automata evolved to maximize average pairwise distances between openings was re-rendered eight times with different initial conditions generated uniformly at random. Similarly, making large maps requires only the use of a larger sized array of cells as demonstrated in figure 5. The matrix that specifies the rule of this automata is

$$\boldsymbol{M} = \begin{bmatrix} 0.910946 & 0.607819 & 0.613347 & 0.941688 & 0.793816 & 0.866123 \\ 0.820984 & 0.920274 & 0.78518 & 0.696141 & 0.700188 & 0.19004 \\ 0.854375 & 0.197165 & 0.451753 & 0.318452 & 0.404226 & 0.567101 \\ 0.0454479 & 0.0755465 & 0.5242 & 0.324387 & 0.357319 & 0.309198 \\ 0.389031 & 0.64076 & 0.325492 & 0.775472 & 0.924891 & 0.671959 \\ 0.198149 & 0.502077 & 0.886719 & 0.63963 & 0.487097 & 0.710947 \end{bmatrix}. \tag{3}$$

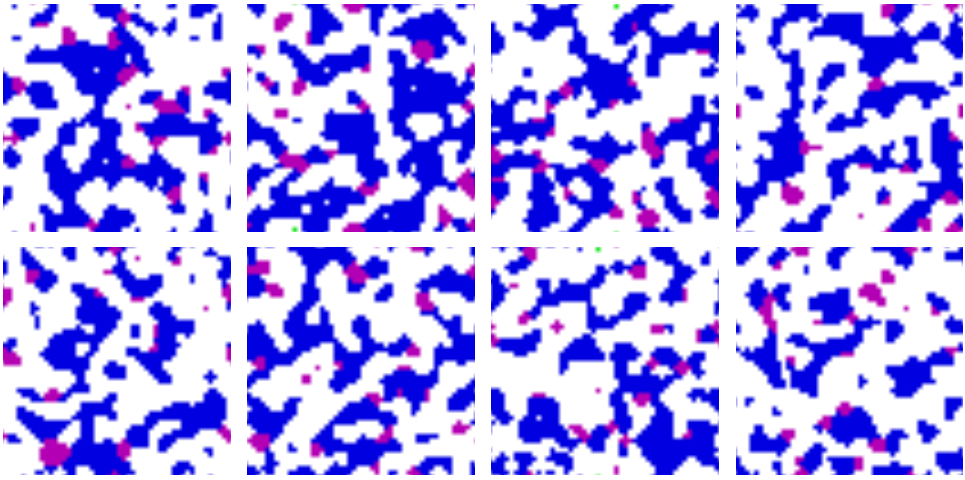


Fig. 4. During evolution of the automata rules a single, fixed set of initial conditions was used. The eight renderings of automata shown above are the result of generating new, random initial conditions. This demonstrates the ability of the rules to be reused to generate multiple maps.

These results show that maps with similar characteristics but very different details can be generated using a single rule. Scalability of these rules also comes for free, again because of the locality of the action of the rule. If a much larger cell space is used then a much larger map is obtained. We

note that with random initial conditions, it is sometimes necessary to move the entrance a few pixels or to use a repair operator of the sort described in Section 5.2.

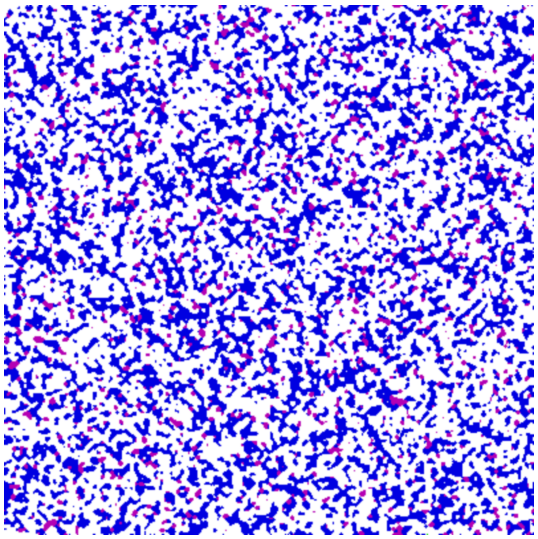


Fig. 5. This is an example of a 400×400 cell rendering of a map built with the same rule used in figure 4. It demonstrates the rescalability of automata-based maps.

5. Conclusions and future directions

Specifying evolvable rules with a real-valued competition matrix is a novel method of specifying rules. Thus far, this study and the one it extends have used this representation to create cavern-like maps, a narrowly focused goal related to video games. There are a large number of other potential applications including ecological modeling, the domain that inspired the application. Many of the advantages of using a cellular automata to generate maps, rescalability and reusability, come from the local nature of cellular automata. In the remainder of this section, we will look at properties that arise directly from the real valued nature of the encoding.

5.1. Exploiting morphability

The local nature of automata updating, quite useful when replayability and scalability are issues, nevertheless leads to a problem with relative uniformity of appearance. In this section, we will find a technique for avoiding the relative uniformity of appearance.

As noted in the introduction, a representation for evolution is *morphable* if convex combinations of instances of the representation are instances of the representation. The real-valued matrices used to specify fashion-based cellular automata clearly have this property. In this section, we will demonstrate a use for morphability and speculate about another.

A *morph* from the rule specified by matrix M_1 to the rule specified by matrix M_2 is the line segment in rule space given by

$$(1 - \lambda)M_1 + \lambda M_2 \,, \qquad 0 \leq \lambda \leq 1 \,.$$

A morph is a linear space of rules, and our second conjectural application, re-evolvability, will attempt to exploit this in a future study. Suppose, however, that we permit the rule used to update the automata to vary spatially, deriving the value of λ to vary based on the position within the cell matrix. Then, we can obtain results like those shown in figure 6.

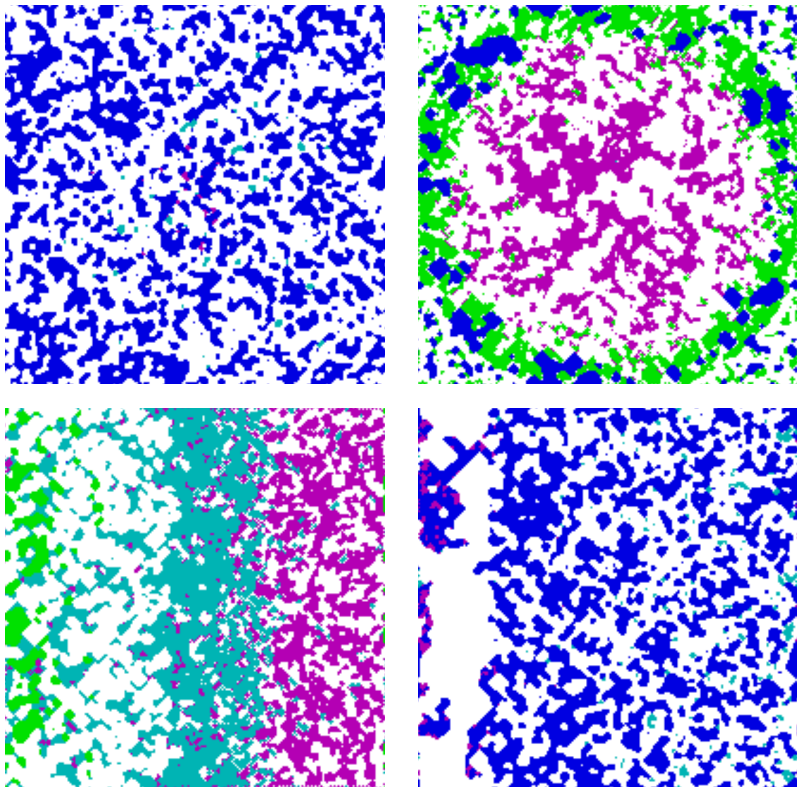


Fig. 6. The top two maps permit the morphing parameter to operate radially, the bottom two laterally.

The maps in figure 6 were selected by examining the 435 possible pairs of morphs between distinct best-of-run (most fit) rules from 30 runs of the baseline algorithm. Two sorts of morphs were performed, *radial* and *lateral*. The upper panels of the figure are radial morphs, where λ is zero at the center of the picture and one in the corners, the lower panels are lateral morphs where λ is the fraction of the distance across the picture in the horizontal direction.

Some of these resulting maps are empty in the regions where λ is not near one or near zero and many of the maps exhibit a solid, impassible zone even more severe than the one in the bottom left panel of figure 6. The left radial morph is subtle — features appear near the center of the picture that do not appear elsewhere, while the right radial morph shows a large change in the character of the caverns from the middle to the edge of the diagram.

5.2. Dynamic programming repair

The use of morphing substantially improves the variety of map types available to a game designer but there is also the issue of compatibility of the rules. The fact that the number of pairs of rules, and hence morphs, is a quadratic function of the number of rules means that there are typically many pairs to examine and so acceptable morph-based maps can usually be located. On the other hand, the barrier-like feature in the lower left example in figure 6 is fairly common as are mostly-empty maps. This suggests that a repair technique might be a viable alternative.

Dynamic programming is already used to measure unobstructed distances between openings during fitness evaluation. If we have a map produced by morphing that does not obey the connectivity constraint, that all openings be mutually accessible, then a path with the minimal number of unobstructed squares between pairs of openings could easily be located and then those minimal obstruction sets cleared. This sort of repair operator would be especially useful when the number of obstructions to be cleared is minimal. More complex, and possibly more effective, suggestions appear in the next section.

5.3. Enhancing morphability

The advantage of morphing between a pair of rules is that it opens up a wider range of possible appearances for level maps, potentially solving the issue of lack of long-range variability in appearance. The down side of morphing, at least thus far, is that many of these morphs are not usable maps. There are at least two possible ways to address this problem.

The first method for avoiding unusable morphs is to evolve pairs of rules that are directly tested, during fitness evaluation, for their ability to generate useful morphs. There is a potential danger here: morphing from a cellular automata to itself. This would generate a morphable pair of rules with a single local appearance. Different matrices can specify the same rules and so, in addition to simultaneously evolving two automata rules, the fitness evaluation must include provisions to avoid the large local optima that involves evolving two essentially duplicate rules. The presence of multiple states representing filled-in cells provides an easy way to do this — simply require that the density of types of filled states be distinct in areas largely governed by each of the automata rules.

The second method involves the use of *fertility* testing. Here, we generate a large collection of automata and test their pairs for viable morphs. The automata are then ranked by the number of other automata rules with which they produced viable morphs. The highest scoring automata in this sense have an enhanced probability of producing viable morphs. These could be used directly or might be an excellent starting population for the evolution of morphable automata.

5.4. The potential of re-evolvability

The space in which optimization takes place to find good cavern-generating cellular automata rules is 36 dimensional. Suppose that, instead of evolving all thirty six parameters, we choose a small number of example maps with desirable appearances. We then evolve a set of weights for these examples. This permits us to search a lower dimensional space constructively enriched with good rules. To avoid the potential or re-evolving one of the original rules, there must be a minimum possible weight — ensuring some contribution to the final rule by all the example rules. The notion of re-evolvability is quite natural for cavern maps but could be applied to any morphable representation.

When applied to cavern maps, re-evolution would not only permit a more efficient evolutionary search in a lower dimensional space, but would also permit additional control of the appearance of the maps. It is quite likely that the appearance of re-evolved maps would share some of their characteristics with the examples.

REFERENCES

[1] L. Johnson, G.N. Yannakakis, J. Togelius, Cellular Automata for Real-time Generation of Infinite Cave Levels, in: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, PCGames '10, ACM, New York, NY, USA, 2010, pp. 10:1–10:4.

[2] D. Ashlock, Evolvable Fashion Based Cellular Automata for Generating Cavern Systems, in: Proceedings of the 2015 IEEE Conference on Computatational Intelligence in Games, IEEE Press, Piscataway NJ, 2015, pp. 306–313.

[3] E. Sapin, O. Bailleux, J. Chabrier, *Complex Systems* **11**, (1997).

[4] S. Wolfram, *Physica D* **10**, 1 (1984).

[5] D. Ashlock, C. Lee, C. McGuinness, *IEEE Comput. Intell. Magazine* **2**, 26 (2011).

[6] D. Ashlock, *Evolutionary Computation for Opimization and Modeling*, Springer, New York 2006.

[7] D. Ashlock, C. McGuinness, Decomposing the Level Generation Problem with Tiles, in: Proceedings of CEC 2011, 2011, pp. 849–856.