Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 436732, 9 pages
doi:10.1155/2009/436732

Research Article

Fine-Tuning Parameters for Emergent Environments in Games Using Artificial Intelligence

Vishnu Kotrajaras and Tanawat Kumnoonsate

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Payathai Road, Patumwan Bangkok 10330, Thailand

Correspondence should be addressed to Vishnu Kotrajaras, ajarntoe@gmail.com

Received 30 May 2008; Accepted 8 September 2008

Recommended by Kok Wai Wong

This paper presents the design, development, and test results of a tool for adjusting properties of emergent environment maps automatically according to a given scenario. Adjusting properties for a scenario allows a specific scene to take place while still enables players to meddle with emergent maps. The tool uses genetic algorithm and steepest ascent hill-climbing to learn and adjust map properties.Using the proposed tool, the need for time-consuming and labor-intensive parameter adjustments when setting up scenarios in emergent environment maps is greatly reduced. The tool works by converting the paths of events created by users (i.e., the spreading of fire and the flow of water) for a map to the properties of the map that plays out the scenario set by the given paths of events. Vital event points are preserved while event points outside the given scenario are minimized. Test results show that the tool preserves more than 70 percent of vital event points and reduces event points outside given scenarios to less than 3 percent.

1. Introduction

For computer games that rely on players interacting with the game maps, emergent and realistic environments become crucial to their degree of realism. Emergent environments can be created on a map by dividing the map into cells, augmenting the cells with various physical properties, and building rules for influencing properties between each cell [1]. Natural phenomena, such as the spreading of fire or the flow of water, look more dynamic and realistic than similar phenomena created using scripts [2].

Emergent maps have been widely used in ecological modeling [3], but for computer games, emergent maps have not seen much use. One reason is because when a given scenario is required to take place on a map, every property of every cell needs to be set in order to produce such scenario. Developers need to put in a lot of time and effort. Moreover, scenarios that have not been well set may behave in ways that the developers had not anticipated. One way to produce a scenario is to make properties inactive and play out the scenario using a script, and then switch the properties back on later. This solution has a limitation because players cannot interfere with the scenario. Players must wait until the scenario is completely played out before being able to do any change to parts of the map that are changed by the scenario. In ecological modeling, an emergent map is used by setting all its properties first (usually imitating a real world map) then allowing the scenario to play itself out. The study of what causes a given scenario is through trial and error. Therefore, it will be very useful if, given a scenario, natural properties that cause it can be found automatically.

We present in this paper a hybrid of genetic algorithm and steepest ascent hill-climbing for adjusting properties of every cell on a cellular automata map in order to produce a given scenario. With a tool based on our technique, game developers are able to focus on designing their scenario and spend much less time setting cell properties. Controllable games scenarios lead to the following features, which are usually unavailable in emergent games.

(i) Editable scenario—scenario are very important for games. Crucial moments in a game story can be revealed using well-set scenarios. Furthermore, a well-set scenario can

provide challenge for players. Normally, a scenario is played out using scripts, sacrificing any possible interactions from players. Our editable scenario is different. Developers can specify how a scene is played out just like writing scripts to dictate what happens during the game, but the environment remains emergent throughout the entire play.

(ii) In-game cut scene—although cut scenes produced as movies can be used, in-game cut scenes can tell stories while players are still in the middle of scenarios, without disrupting game flow.

2. Related Works

Regarding game environment, Sweetser and Wiles [1] have developed and tested a cellular automata map for real-time strategy games. Sweetser's experimental system was called EmerGEnT system. Although EmerGEnT system did not model an environment in great detail, it was good enough for using in games, with fire, water, and explosions being integrated into its cellular automata properties. EmerGEnT system can be divided into 3 levels. The first layer is the behavior layer, which shows the effects that players see. The second layer is the rule layer, which controls behavior both between cells and within each cell. The final layer is the property layer, which determines how cells interact according to the rules. Sweetser's work, although consists of many physical properties, does not provide any feature for setting cell properties following given effects that users see.

A probabilistic model can also be used to simulate fire in broad scale over long time periods [3, 4]. Hargrove's model was designed for simulating real forest fires. It included humidity, fuel types, wind, and firebrand. These factors influenced the probability of fire spreading from one cell to another, and the probability of isolated cells getting ignited. Probabilistic models do not perform well for small cell sizes compared to thermodynamic models. To be able to work with cells in a game map, which generally represents small areas, thermodynamic models are more suitable. Probabilistic models are also more difficult to control compared to thermodynamic models, which have precise rules for events.

Hill-climbing can be used for tuning system parameters. Merz et al. [5] developed Opi-MAX for tuning MAX's numeric parameters. MAX is an expert system for high-level diagnosis of customer-reported telephone troubles. It can be customized by changing a set of numeric parameters. Opi-MAX uses a searching algorithm called greedy hill-climbing to optimize parameters. It works by randomly changing parameters one by one. A parameter is repeatedly changed from its initial value and each of its change effect can be observed from overall output. A change that results in a better output will be carried out, and a change that does not contribute to a better output will be ignored. Each parameter is changed for a constant number of times. The system stops when all parameters are dealt with. This technique, however, is not suitable for our work, since we have no good initial values of any property. It can be used to help improving some partially tuned parameters, nevertheless. Therefore, we use it for fine-tuning environment properties adjusted during our genetic algorithm runs.

Neural networks can also be used for parameter tuning. Legenstein et al. [6] developed a movement prediction method for objects moving on 8×8 grids. Each cell on a grid provides an input to the recurrent unsupervised neural networks. It can predict object movement by predicting sensor inputs, which are numeric values like cell properties. Given previous events, Legenstein et al. work predicts the next event, while our work predicts the initial condition given a sequence of events in time. We have experimented with neural networks but found that results were unsatisfactory. Fire only followed given waypoints for about 50% and spread out of the waypoints more than 100%. We believe it was because cells that had different cell properties were allowed to produce similar events (even though the intensity of fire might be different). Therefore, fires with different intensity were trained with similar events, causing the neural network to fail to learn effectively. Therefore, we switched to genetic algorithm.

Breukelaar and Bäck [7] used genetic algorithm to generate cellular automata's transition rules that display a desired behavior. The work was demonstrated by finding rules that evolve all cells to the same state by majority, evolve cells to form a checkerboard, and evolve cells to form desired bitmaps. The main focus was to find transition rules, given a single parameter. In our work, the rules are given, but we need to find several parameter values of all cells at the start of each scenario. Therefore, the genetic encoding is totally different from [7]. Karafyllidis [8] used genetic algorithm to convert continuous-state cellular automata that predicts forest fire spreading to discrete-state cellular automata that outputs nearest results. The number of cells and states for the discrete-state version were also minimized. The outcome cellular automata were used to build a dedicated parallel processor for real-time processing. Only one parameter was used in each cell. It was the rate of fire spreading. Using only one parameter allows faster execution. However, such approximation is not applicable for our work since we want players to still be able to physically change all properties of each cell.

3. Tool for Adjusting Properties of Emergent Environment Maps

3.1. Overview. We base our cellular automata maps and rules on EmerGEnT system [1]. Our tool can adjust properties in EmerGEnT system-like map automatically to create a scenario of fire spreading and water flowing that matches the scenario given by a user. Properties that our tool adjusts include material, temperature, mass, damage, and wetness for the spreading of fire, and height for the flow of water. Paths of both kinds of events are controlled by waypoints (currently, there are two kinds of events, fire and water flow). Each waypoint is defined by its position, the time an event takes place at that position, and the radius of the event, as shown in Figure 1. The tool then creates a timetable containing the beginning and the end of each event on every

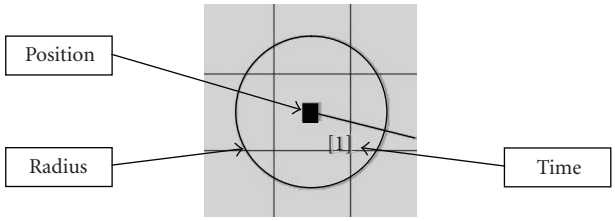


Figure 1: Waypoint.

cell. Impossible paths, such as fire crossing water ways or fire spreading in the opposite direction to the wind, are not produced. When a fire is spreading, its burning area moves along the given waypoints. For water, flooding areas never dry out. At the beginning of the spreading of fire, an object that changes temperature to 3000 units is put into the first waypoint at the time given by a user. For water, an object that adds 30 units of water per unit time is put into the first waypoint at the time given by the user. There can be more than one starting waypoint for both fire and water, depending on each designed scenario. The map properties are then adjusted by genetic algorithm, with steepest ascent hill-climbing being applied at selected generations. Finally, the best chromosomes are kept to be included in the initial population of genetic algorithm when adjusting other scenarios. These chromosomes will be chosen during genetic algorithm if they fit any newly given scenario.

3.2. Genetic Algorithm. Each individual in our population contains 2 strands of chromosomes, chromosomes that store properties related to fire (see Figure 2) and chromosomes that store properties related to water (see Figure 3). Separating fire and water properties results in a reduced search space for each type of events because for each type of events, there are some properties unrelated to it. Properties related to fire include material, temperature, mass, damage, and wetness. There is, according to EmerGEnT model, only 1 property related to water. It is the water's height. All property values are stored in real number, except the material value, which is stored as an integer representing the material identifier of a cell. Each cell does not need a special firing condition because its properties (including how much it burns and how much water is in it) constantly change according to properties of neighbor cells and related properties within that cell, according to the terrain physical rules given for EmerGEnT model. In order for a scenario to take place, we only need a starting condition, which the system gives by putting in fire sources or water sources at starting waypoints.

Our genetic algorithm has population of 1000 chromosomes and evolves for 100 generations. The initial population is generated randomly, with some limited range defined for each value in order to speedup convergence. A new generation is selected from the following.

(i) The highest fitness chromosome.

(ii) The first 1% elitist chromosomes. These will be subjected to mutation.

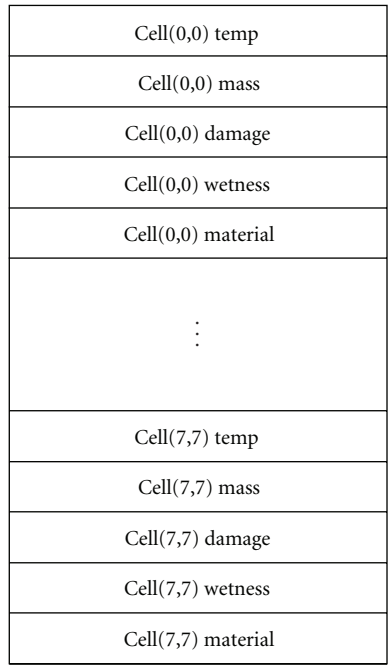


Figure 2: Chromosome for fire event.

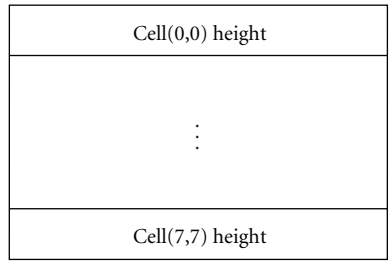


Figure 3: Chromosome for water event.

(iii) The crossover of chromosomes chosen by (1). selectPopulation is an integer used to identify a chromosome. The lower the value, the higher the fitness of the selected chromosome. For example, if selectPopulation is 0, we know we have selected the highest fitness chromosome. Function random(a, b) returns a random number between a and b, but not including b. Equation (1) guarantees that the bottom half of the chromosomes will never be chosen for crossover. After the crossover finishes, half of the resulting chromosomes are mutated.

Our genetic algorithm uses elite strategy because it tries to select elite chromosomes first. The first 1% of the highest fitness chromosomes is selected for certain and the rest are in the top half:

$$\text{selectPopulation} = \text{random}\left(0, \frac{\text{populationSize}}{\text{random}(2,6)}\right). \quad (1)$$

When we perform a crossover between two chromosomes, each property value on the first chromosome can be

combined with its counterpart from the second chromosome. We use uniform crossover with blending defined by (2):

$$p_{\text{new1}} = \beta p_1 + (1-\beta)p_2,$$
$$p_{\text{new2}} = \beta p_2 + (1-\beta)p_1. \quad (2)$$

From (2), p_1 is a property value (in real number) selected from the first chromosome, for example, it can be the temperature of a cell at coordinate (2, 3) in our map, as defined by the first chromosome. p_2 is a corresponding property value from the second chromosome. For example, if p_1 is the temperature of a cell at coordinate (2, 3), as defined by the first chromosome, then p_2 must be the temperature of a cell at coordinate (2, 3), as defined by the second chromosome. p_{new1} is the new value of that property in the first resulting chromosome. p_{new2} is the new value of that property in the second resulting chromosome. β is used to combine the values from p_1 and p_2. The value of β varies in each crossover. β is selected randomly from 0, 1 and a random value between 0 and 1. The chance of selecting each choice from these 3 choices is equal. We experimented with 2 other ratios for these 3 choices on 3 sample maps (the first map has more than 1 fire path, the second map has a very long fire path, and the third map contains a very long water flow). With all other settings equal, on average, equal ratio gave the best result. Therefore, we decided to use it in our genetic algorithm.

Mutation rate of 10, 20, and 30% were tested on the 3 sample maps above. It was found that the mutation rate of 20% gave the best result on average. Therefore, we choose the mutation rate of 20% for our genetic algorithm. The mutation range is constrained to be within 50 units away from the old value in order to prevent very odd chromosomes with low fitness from being produced.

The fitness value of each chromosome is calculated from the average of the fitness of each time unit that events occur, with weight defined by (3). Any time frame that contains event(s) at waypoint(s) is given a higher fitness value than a time frame with no event point in order to make the scores at event points stand out. Events occurring later in a scenario are also considered more important than events occurring early in the scenario. This allows different starting scenes for a single scenario

$$\text{fitness}_t^{\text{waypoint}} = \left(1.8 + \frac{0.2 \times t}{\text{time}_{\max}}\right) \times \text{fitness}_t,$$
$$\text{fitness}_t^{\text{non-waypoint}} = \left(0.8 + \frac{0.2 \times t}{\text{time}_{\max}}\right) \times \text{fitness}_t. \quad (3)$$

Our fitness functions were defined after several experiments. $\text{fitness}_t^{\text{waypoint}}$ is the fitness value at time t, if that time contains event(s) at waypoint(s). The importance of that time frame has also been weighed into its value. $\text{time}_{\max}$ is the maximum time frame that the current scenario takes place. fitness_t is the fitness value at time t before being given any weight. $\text{fitness}_t^{\text{non-waypoint}}$ is defined similar to $\text{fitness}_t^{\text{waypoint}}$, except that it is given less weight due to its lack of event points.

The fitness of each time frame before weighing (fitness_t) is defined in (4) as follows:

$$\text{fitness}_t = \frac{\sum_{i=1}^{n} \text{fitness}\,(\text{relevantCell}_i)}{m \times 10}, \quad (4)$$

relevantCell_i is a cell that burns or floods at time t or was designed to burn or flood at time t (we do not count a cell twice, however). The number n is the sum of the number of cells that actually burn or flood at time t and the number of cells designed to burn or flood at time t (again, we do not count a single cell twice). fitness(c) is a fitness value of cell c. It can have a negative value depending on whether the cell being in the designed path or not (see below). m is the number of cells designed to burn or flood at time t. The maximum fitness score for each cell is 10. If our parameter tuning is perfect, cells designed to burn or flood at time t will actually burn or flood at that time frame with fitness value equal to 10. In addition, no other cells will burn or flood. Therefore, n will equal to m and the value of fitness_t will be 1.

The value of fitness(c) is calculated from each of the following steps.

(i) If a cell outside specified paths produces events—lose 1 point if an adjacent cell is inside any event path. If the cell does not have any adjacent path, 5 points are deducted instead. This discourages events outside the specified path.

(ii) If a cell does not produce an event when the event is set to occur—gain points equal to two times the cell temperature divided by maximum temperature if the event is a spreading of fire. The maximum score obtainable from this portion of the function is 2. Water events get no score here. The reason the fire situation gets some score even though the cell does not produce the event is because high temperature gives the cell a probability of burning in later time frames, which can result in similar fire events later on.

(iii) If a cell produces an event at its specified time—gain 8 points. Get additional points according to (5) and (6) for fire and water, respectively,

$$\text{addScore}_{\text{fire}} = \frac{2 * \text{Burn}}{0.5 * \text{MaxBurn}}. \quad (5)$$

For (5), Burn is the intensity of fire in the calculated cell and MaxBurn is the maximum possible value of Burn. The maximum score from this equation is limited to 2. Any burn that spreads with at least half the intensity of the maximum intensity will get full mark. The reason we need $\text{addScore}_{\text{fire}}$ is to encourage all fires to burn fast. From our experiment, this can prevent fires from dying out unexpectedly in the middle of scenarios:

$$\text{addScore}_{\text{water}} = \frac{2 * \text{fluid}}{0.5 * \text{MaxFluid}}. \quad (6)$$

For (6), fluid is the amount of water currently in the cell. MaxFluid is the maximum amount of water that cell can contain. Similar to (5), (6) has its maximum value being 2 and it is needed in order to prevent water from flowing not as far as designed. The difference from (5) is that we use the amount of water instead of speed. This is because in our model, based on Sweetser's, there is no water speed parameter.

Table 1: Result of hill-climbing test.

Setting	20th–100th	60th–100th	100th	Not use
Average Map 1	0.516818	0.542901	0.527788	0.476567
Average Map 2	0.448464	0.457916	0.447966	0.406109
Average Map 3	0.669976	0.66909	0.669541	0.669655
Average All	0.545086	0.556636	0.548431	0.517444

```
int  tuneValueFire = 27;
real bestFitness;
for each cell{
      //tune for fire event
      real currentFitness = · · · · //calculate the cell's fitness value
      While (tuneValueFire >0){
         Find the fitness when add the temperature value by tuneValueFire
         Find the fitness when subtract the temperature value by tuneValueFire
         Find the fitness when add the mass value by tuneValueFire
         Find the fitness when subtract the mass value by tuneValueFire
         Find the fitness when add the damage value by tuneValueFire
         . . .//try both add and subtract for all parameters of fire chromosomes
         . . .

         if (the best fitness value (compared to currentFitness) is found from the tuning
         trial results above){
               select a modification that causes such fitness
               commit changes according to the selected modification
         } else {
               tuneValueFire = tuneValueFire;
         }
      } // end while
} // end for each cell
```

Algorithm 1: Pseudocode for steepest ascent hill-climbing on fire related values.

(iv) If a cell produces a specified event before its intended starting time, but not more than 1 time unit—gain 4 points. This is to allow a slightly different scenario to still gain points.

(v) If a cell still produces a given event after its intended end time, but not more than 4 time unit—in case of fire, gain 5 points minus the difference between current time and end time. If the difference in time is just 1, the score will be 4, similar to the score when an event takes place before its intended starting time. But we give points for other nearby time frames in order to allow for fire trails. From our experiments, fire trails are very important for an overall fitness of fire events. There is no score for water remaining in a cell, however, since our system follows Sweetser's model that lets water stay in a cell indefinitely.

3.3. Steepest Ascent Hill-Climbing. We use steepest ascent hill-climbing to the best chromosome, with the same fitness function as our genetic algorithm, in order to improve map's properties. Table 1 shows the effect of steepest ascent hill-climbing used in our tool, in term of fitness values compared among four settings. Three settings employ steepest ascent hill-climbing at every 20 generations of the genetic algorithm.

In the first setting, we start applying it at the 20th generation. In the second setting, we start applying it at the 60th generation. In the third setting, we apply the algorithm only to the last generation (100th generation). In the fourth setting, we do not use steepest ascent hill-climbing algorithm. Each setting is tested on three different maps of 8×8 cells. For each map, each setting is tested 3 times. The test maps are chosen to represent 3 scenarios that could be set by developers. Map 1 contains a fire that breaks into 2 paths. Map 2 contains a long path of fire. Map 3 contains water flow. Each property is tuned in sequence until it cannot be tuned further. The values used in tuning come from observations during experiments. The algorithm for tuning fire-related parameters is shown in pseudocode in Algorithm 1. For water events, the algorithm is similar, except it works only on water-related parameter.

The result shows that, on average, tests that steepest ascent hill-climbing are applied to have noticeably better fitness values than the tests without steepest ascent hill-climbing. From Table 1, it seems that starting to apply steepest ascent hill-climbing at the 60th generation gives the best fitness value (except in Map 3, where its fitness value is slightly lower than the fitness values from other

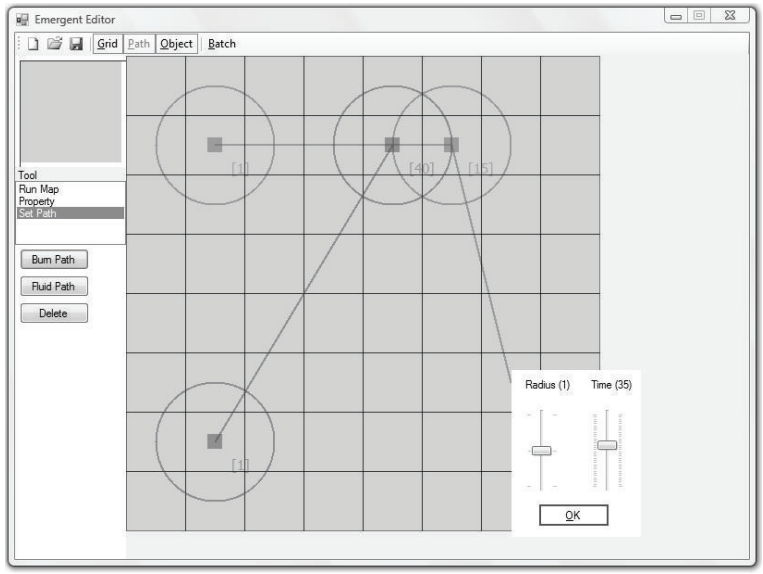


Figure 4: Path editor.

tests). More experiments are needed in order to determine whether steepest ascent hill-climbing produces a significantly better result than ordinary genetic algorithm in our problem. We, therefore, run our parameter tuning with genetic algorithm alone and with our steepest ascent hill-climbing as the genetic algorithm enhancement (we apply the second setting from Table 1). The testing setup and its outcome are discussed in Section 4.

3.4. Emergent Editor. Our tool for adjusting properties of emergent environment maps is in the form of a map editor. We name the editor emergent editor. Its features can be divided into 4 parts.

(i) Path editor (Figure 4)—a user can define, edit, or delete waypoints of fire events and water events.

(ii) Automatic properties adjustment (Figure 5)—this feature adjusts map properties automatically according to a given scenario defined by paths of events.

(iii) Event player (Figure 6)—it can show the original scenario defined by paths of events, and the scenario created after the map properties are set.

(iv) Property editor (Figure 7)—a user can also view or edit map properties directly from this view.

4. Testing and Results

In this section, we first discuss results from experiments using genetic algorithm enhanced by steepest ascent hill-climbing (which gives better results than using genetic algorithm alone). Then we discuss whether using steepest ascent hill-climbing really gives significantly better results statistically. Finally, an experiment showing how our parameter-tuning tool can help map designers save time is presented.

Testing is initiated by creating paths of events randomly on a map of 8×8 cells. Paths of events are limited to total of 6 waypoints. Events with the same number of waypoints are tested between 2–5 times. Each waypoint has its radius of one or two cells. The total running time of each event is limited to 50 time units. The wind direction is chosen randomly between no wind and random direction. From 100 tests, using genetic algorithm and steepest ascent hill-climbing, our result shows that the tool preserves 75.09% of event points at waypoints on average and produces event points outside given scenarios by only 2.3% on average.

Figure 8 shows our test result grouped by the number of waypoints. The horizontal axis represents various fire and water scenarios. $\mathrm{Fm}_1\mathrm{m}_2 \ldots \mathrm{m}_n$ represents n fire paths in the map, where the first path has m_1 waypoints, the second path has m_2 waypoints, and so on. $\mathrm{Wm}_1\mathrm{m}_2 \ldots \mathrm{m}_n$ represents water paths in the same way. The vertical axis represents the percentage of event points for each scenario.

Figure 9 illustrates one of the test scenarios with 3 waypoints of fire spreading. Figure 9(a) shows the events designed to take place at 3 points in time, while Figure 9(b) shows actual events that take place after the actual parameter adjustment at the same points in time. Darker cells are forest cells, while lighter cells are grass cells. There is also a water cell at the middle-bottom of the map (white cell). Cells with white circles are cells that catch fire.

There are 11 tests that our tool gives less than half of events correctly at waypoints. When we look into their causes, we discover that their scenarios are impossible to take place. Fire was set to burn longer than the fuel in the map could support. Water was set to flow too quick or too far from its source. In our system, the further away from its source, the slower the water can flow. This is because there are more cells to absorb water.

In order to find out whether steepest ascent hill-climbing significantly enhances the accuracy, we run the experiment again, with and without steepest ascent hill-climbing, and compare their results using a paired t-test.

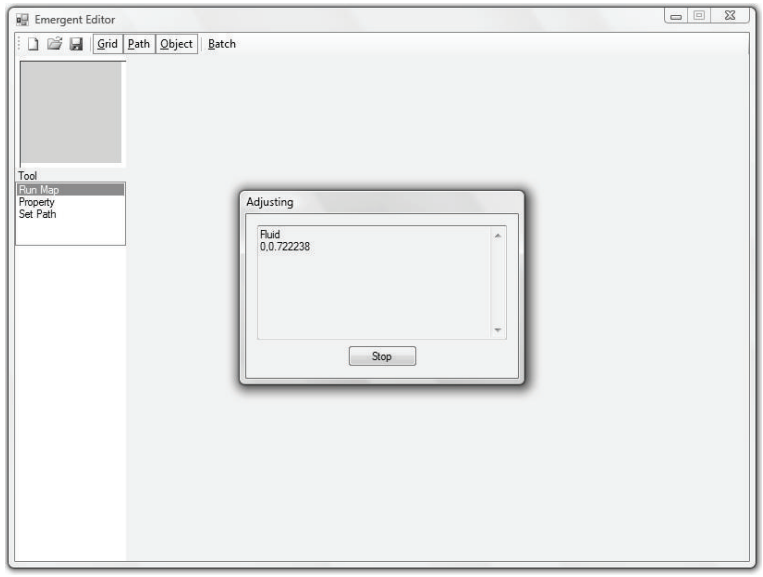


Figure 5: Automatic properties adjustment.

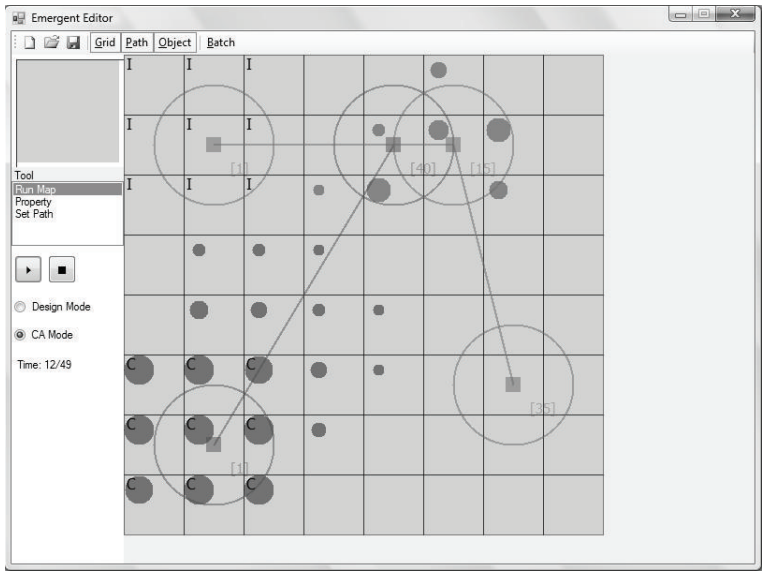


Figure 6: Event player.

The paired t-test result for the number of preserved waypoints informs us that the two-tailed P value is less than .0001. By conventional criteria, this difference is considered to be extremely statistically significant. The difference between the mean of the experiment using steepest ascent hill-climbing and the experiment that does not use steepest ascent hill-climbing equals to 2.2953264. The 95% confidence interval of this difference is from 1.3483512 to 3.2423016. The intermediate values used in calculations include $t = 4.8094$, df $= 99$, and standard error of difference $= 0.477$. For the percentage of outside fire path, the difference is found to be not quite statistically significant. The 95% confidence interval of this difference is from -3.10873116 to 0.26262415. For the percentage of outside water path, the difference is found not to be statistically significant. The 95% confidence interval of this difference is from -0.29874686 to 0.40685569.

From the t-test results, we can conclude that using steepest ascent hill-climbing gives a significant boost to the number of preserved waypoints. Therefore, it should be used in conjunction with genetic algorithm for tuning map parameters.

In order to test whether the tool that uses our parameter-tuning technique actually benefits scenario designers, we perform an experiment by having 6 testers manually tune the maps from Section 3.3 for 30 minutes per map

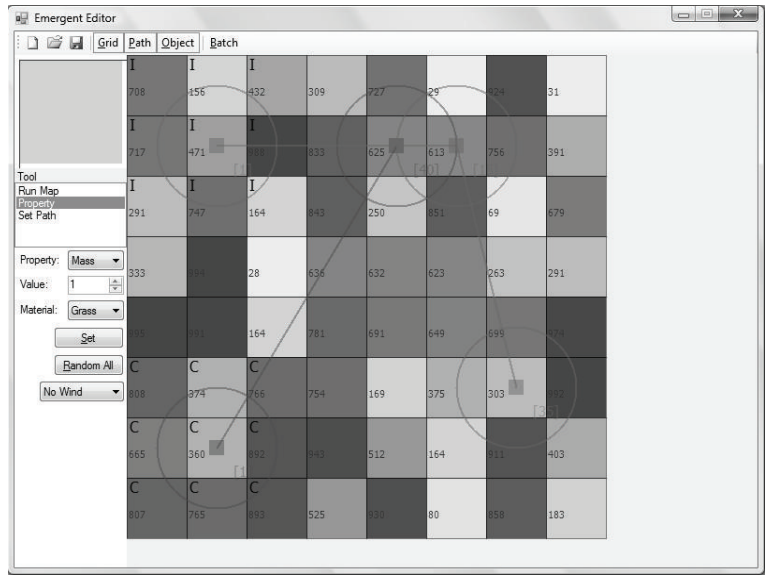


Figure 7: Property editor.

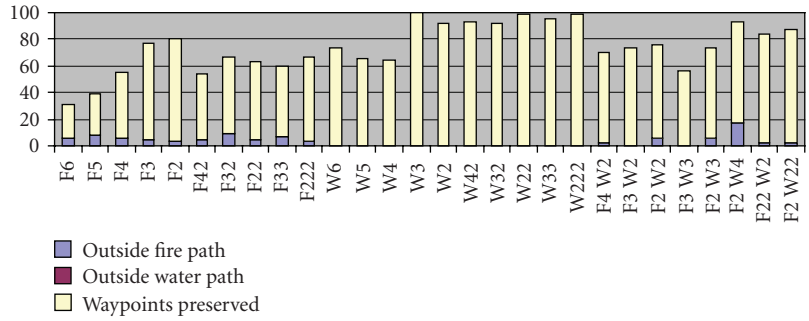
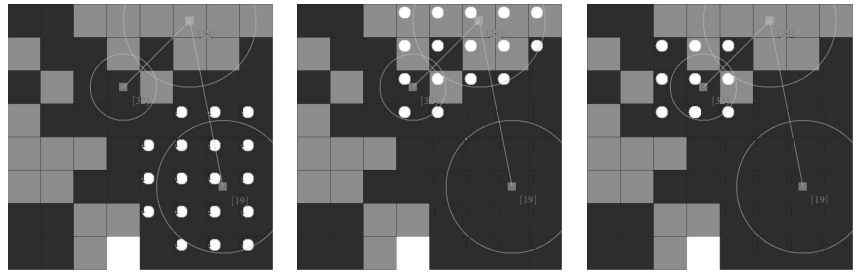
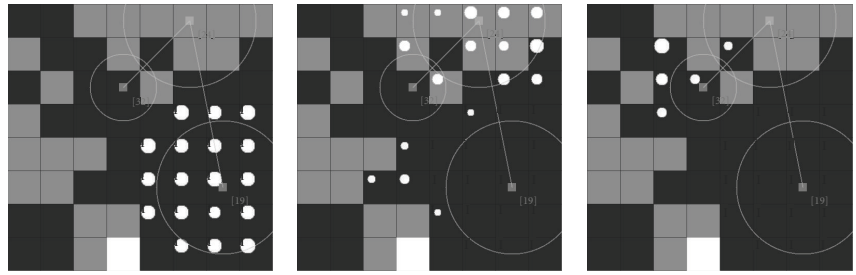


Figure 8: Test result.



(a)



(b)

Figure 9: Test scenario.

Table 2: Tuning accuracy results from testers and the tool.

Tester ID	Waypoint preserved (%)	Outside path (%)
1	55.68	1.23
2	61.11	0.00
3	62.47	0.00
4	42.22	45.47
5	51.11	0.00
6	35.56	0.00
Average tester result	51.36	7.78
Our tool result	76.05	10.06

(the 30-minute period is the time our testers are willing to spend). We compare the testers' results with the results obtained by our tool in Table 2.

From Table 2, it can be seen that our tool is much more precise in preserving waypoints, given an equal period of operation time. Therefore, our tool is capable of producing accurate scenarios faster than human. For outside fire and water paths, our tool performs worse than human. This turns out to be because most of the testers cheat by removing fuel from all outside paths. This cheat cannot be done in actual nature simulations or games because it will produce very unnatural maps. For tester 4 who does not cheat, the amount of outside path is 45.47%. This is another good indication that using our tool can save valuable development time.

5. Conclusion

From our experiment, we conclude that genetic algorithm, with help from steepest ascent hill-climbing technique, can be used effectively for adjusting parameters in emergent maps which leads to simple-to-control scenarios without the need to manually edit any property.

Some problems, such as fire burning out before the expected ending time and water running too slowly or too short in distance, still need to be solved. This can be solved by having the algorithm also adjust the initial temperature of fire and the amount of water at the first waypoint. There are also other possible improvements. It may be useful to allow users to control scenarios with other means besides creating paths of events. Increasing the speed of the tool will allow a better use with bigger maps and more complex environments. Other kinds of emergent environments, such as environments used for actual ecological modeling, are good candidates for expanding the value of our tool.

Acknowledgment

This research is sponsored by Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand.

References

[1] P. Sweetser and J. Wiles, "Using cellular automata to facilitate emergence in game environments," in *Proceedings of the 4th International Conference on Entertainment Computing (ICEC '05)*, Sanda, Japan, September 2005.

[2] P. Sweetser and J. Wiles, "Scripting versus emergence: issues for game developers and players in game environment design," *International Journal of Intelligent Games and Simulations*, vol. 4, no. 1, pp. 1–9, 2005.

[3] W. W. Hargrove, R. H. Gardner, M. G. Turner, W. H. Romme, and D. G. Despain, "Simulating fire patterns in heterogeneous landscapes," *Ecological Modelling*, vol. 135, no. 2-3, pp. 243–263, 2000.

[4] W. Song, F. Weicheng, W. Binghong, and Z. Jianjun, "Self-organized criticality of forest fire in China," *Ecological Modelling*, vol. 145, no. 1, pp. 61–68, 2001.

[5] C. J. Merz, M. Pazzani, and A. P. Danyluk, "Tuning numeric parameters of a knowledge-based system for troubleshooting the local loop of the telephone network," *IEEE Expert*, vol. 11, no. 1, pp. 44–49, 1996.

[6] R. Legenstein, H. Markram, and W. Maass, "Input prediction and autonomous movement analysis in recurrent circuits of spiking neurons," *Reviews in the Neurosciences*, vol. 14, no. 1-2, pp. 5–19, 2003.

[7] R. Breukelaar and Th. Bäck, "Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: emergence of behavior," in *Proceedings of Conference on Genetic and Evolutionary Computation (GECCO '05)*, pp. 107–114, Washington, DC, USA, June 2005.

[8] I. Karafyllidis, "Design of a dedicated parallel processor for the prediction of forest fire spreading using cellular automata and genetic algorithms," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 1, pp. 19–36, 2004.