Cellular automata for real-time generation of infinite cave levels

Lawrence Johnson
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
lkj@itu.dk

Georgios N. Yannakakis
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
yannakakis@itu.dk

Julian Togelius
IT University of Copenhagen
Rued Langgaards Vej 7
Copenhagen, Denmark
juto@itu.dk

ABSTRACT

This paper presents a reliable and efficient approach to procedurally generating level maps based on the self-organization capabilities of cellular automata (CA). A simple CA-based algorithm is evaluated on an infinite cave game, generating playable and well-designed tunnel-based maps. The algorithm has very low computational cost, permitting realtime content generation, and the proposed map representation provides sufficient flexibility with respect to level design.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*; H.1.2 [**Models and Principles**]: User — Machine Systems—*Human factors*

1. INTRODUCTION

Maps are vital components of level design for many types of games (e.g. first-person shooters, real-time strategy games and flight simulators) and their careful design (manual or procedural) contributes vastly to player experience. There are several reasons for why procedural generation of maps is important for game development. First, having an inexhaustible source of new maps means that levels become less predictable, which contributes to the players' curiosity [7] and the game's life-span. Second, for some games (e.g roguelikes) the core mechanic of the game design requires the realtime generation of maps; this paper considers an infinite cave map system. Third, given that content is represented in an efficient manner and the procedural content generation (PCG) algorithm is parametrizable in the right ways, maps can be adjusted to match player needs and abilities identified via the player's behavior during gameplay. Finally, PCG can be used as an assisting authoring tool for complementing human creativity and level design expertise existent within commercial game development.

This paper considers presents an efficient real-time PCG approach for generating infinite maps consisting of cave tunnels. The motivation for this algorithm is that the core mechanic for the test-bed game requires infinite map generation. The PCG approach proposed utilizes cellular automata (CA). Cellular automata self-organize and determine the existence of floors and rocks in the cave map and offer a simple, yet efficient and reliable solution to real-time cave-map generation.

PCGames 2010, June 18, Monterey, CA, USA

2. BACKGROUND

This section reviews previous research on procedural map generation and on using cellular automata for proecdural content generation.

2.1 Procedural Map Generation

Fractals [4, 8] such as mid-point displacement algorithms [2] are in common use for realtime map generation. Their main advantages are that it's easy to generate believable maps using such algorithms, and that they are typically very computationally efficient. However, an important limitation of such approaches is their highly indirect representation; for many such representations, the only "parameter" is a random generator seed, and very small changes in this seed will lead to a completely different map. This is an issue as it is often desirable for the PCG algorithm to be *controllable* or meaningfully parametrizable, meaning that certain aspects of how the generated map should look (e.g. size, frequency and distribution of various) features be specifiable by the designer.

The *roguelike* genre of games (the original *Rogue* game as well as countless successors, such as *Nethack*, *Moria* and *Diablo*) are based completely on random map generation. The map generation algorithms typically work either similarly to the fractal terrain generation approaches above or by glueing together a number of prefabricated segments [1].

Another, more controllable approach is to use software agents. Doran and Parberry [3] describe an authoring tool that generates terrain elevation heightmaps based on designer-defined constraints, and letting loose large numbers of terrain-forming agents of different types in different phases.

Search-based procedural content generation (SBPCG) is a paradigm which offers very high controllability, and the ability to generate particular forms of content which could not be generate otherwise, at the expense of computation time [12]. Among the few studies on search-based generation of maps, Frade et al [6] generate maps ensuring that terrains have enough accessible area for the players using genetic programming. Similarly, Sorenson and Pasquier [9] use a stochastic constraint optimization algorithm (feasible-

infeasible 2-population genetic algorithm) for maximizing the lengths of paths from start to finish in 2D maps. However, both the above search-based PCG approaches run offline opposed to the realtime CA approach presented here.

2.2 Cellular automata for PCG

Cellular automata have been used extensively in games for modeling environmental systems like heat and fire, rain and fluid flow, pressure and explosions [5, 11] and in combination with influence maps for agent decision making [10]. Moreover, in [8] CA are used for thermal and hydraulic erosion in procedural terrain generation

To the best of the authors' knowledge there is no study reporting the use of CA for generating complete 2D terrains. There are a few webpages that propose the use of CA on small single grids providing no reliable evaluation of the algorithm. This paper provides a CA-based algorithm for the procedural generation of multiple interconnected grids which can be combined to the effect of infinite cave maps.

3. CA-PCG APPROACH

The proposed CA-PCG mechanism starts with a square grid — a 50×50 cell grid is used in the test-bed game examined. This initial grid will be referred to as the central *base* grid. Then four adjacent grids are built to the north, south, east and west of the central *base* grid. Advanced versions of the algorithm could consider the northeast, northwest, southeast, and southwest adjacent grids. Each cell within the grid contains information about the cell's location in the base grid, the state of its neighbor cells represented by an aggregated *neighborhood value*, the type of the cell (floor, wall or rock), and the cell's group number. The neighborhood value is calculated as the sum of rock cells available within the neighborhood.

The central base grid is initialized with floor cells. The algorithm then uses uniformly distributed random numbers to convert r% of the cells from floor to rock state — r equals 50% in the experiments presented in this paper. Iterated cellular automata are then applied to the generated grid, examining the relationships between the floor and the rock cells generated. The state of the *neighborhood* cells of each cell is examined. In this paper, the eight neighbor cells of each cell are considered at each iteration of the algorithm and define its neighborhood (i.e. we use the Moore neighborhood). The single rule embedded in the cellular automata characterizes the cell as *rock* if the neighborhood value is greater than or equal to T and as *floor* if otherwise; T equals 5 in this paper.

The iterative CA algorithm runs for a number of times, n. This parameter has an impact on the average width of the caves generated; on average, the higher n the wider the cave (more floor, less rock).

3.1 Content Representation

Given the specifics of the CA algorithm, content is represented as four parameters a designer (or a machine) could vary to generate a significant number of dissimilar cave levels. The parameters considered are summarized as follows:

- r: initial percentage of rock cells.
- n: CA iterations.
- T: neighborhood value threshold that defines a rock.
- M: Moore neighborhood size.

In order to completely specify the set of parameters that uniquely define a given level, the random seed for the generation of initial cell distribution can be considered an additional parameter; given the same random seed and the same values of r, n, T and M, the same cave can be regenerated.

3.2 Wall Cells

The wall cells need to be defined after the completion of the CA algorithm. Walls are designated rock cells which have at least one neighboring floor cell. After wall cells are determined non-connecting floor areas are assigned a number.

3.3 Adjacent Base Grids

Once the initial base grid is generated the same procedure is followed for generating its four adjacent base grids. The random seed is stored into a data structure which contains all the data for regenerating the cave terrain in case the player returns to the same base grid. A typical data structure contains the seed for a given base grid and pointers to the base grid data structures for all neighboring base grids.

The final step is to evaluate if the cave is acceptable, and if possible, generate a smooth route between tunnels. For this purpose the center (initial) base grid and its four adjacent base grids are checked for continuity. If there are no connections between two adjacent grids, the algorithm picks the two floor cells closest to the border and generates a tunnel of a predefined width between them.

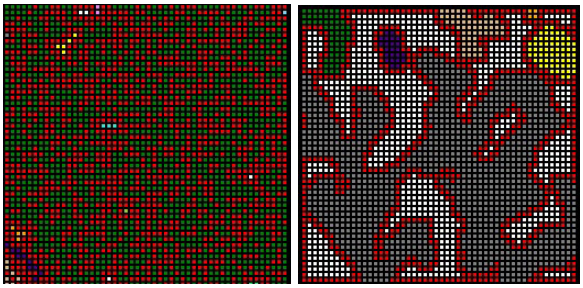
Subsequently, an additional n CA iterations (n equals 2 in the experiments presented here) are run *on the whole 5-base grid together* in order to eliminate inconsistencies between base grids and smooth-out the generated tunnels. Each time a new base grid is generated it is interconnected to its adjacent grid by running two more iterations of the CA algorithm on the new and its adjacent base grid.

4. CAVE CRAWLER

The game embedding the procedurally generated cave map is named *Cave Crawler*. *Cave Crawler*'s game mechanics are inspired by the games *Penn & Teller's Smoke and Mirrors*, *Diablo*, and *Castle Crashers*. *Cave Crawler* is a four player, co-op, game. The game world is displayed in the same manner as in *Diablo* (bird-eye view), yet all four players play on the same screen like *Castle Crashers*. Cave Crawler is an abusive game in that players are led to believe that they are progressing towards the end of the game, while there is in fact no end to the cave tunnels. In order to achieve this effect reliably, the game must procedurally generate all the maps as well as the enemies the players have to face. Games like *Left for Dead* have shown that players value the game when playing against an intelligent entity that spawns enemies at periodic intervals.

5. PERFORMANCE MEASURE

To test the efficiency of the CA approach we designed a performance measure as follows. Given that the operation occurs in realtime, computational effort via CPU time defines a meaningful heuristic of performance. The computational cost of CA operations on a Moore neighborhood is on the order of $(2M+1)^2$ where M is the Moore neighborhood



(a) Random map (b) CA map

Figure 1: Comparison between a CA and a randomly generated map ($r = 0.5$ in both maps); CA parameters: $n = 4$, $M = 1$, $T = 5$. Rock and wall cells are represented by red and white color respectively. Colored areas represent different tunnels (floor clusters).

size. Given that the operation is performed on all the cells of the grid the computational effort equals $n(wh(2M+1)^2)$, where w and h are the width and height of the grid, respectively and n is the number of CA iterations.

All experiments presented in this paper were run on a laptop with Windows 7, Intel Pentium M processor 1.73 GHz, 1.50 GB ram. The game and algorithms are implemented in C# using the XNA framework. While CPU time fits our requirements at this stage, other performance measures could be considered in future studies including the *likeliness* of the generated tunnels which could be expressed as a level of wall roughness/smoothness.

Random cave maps are generated in $1.4 \cdot 10^{-4}$ milliseconds (average value out of 10 runs) while the CA-based algorithm generates the map in $4.1 \cdot 10^{-1}$ milliseconds on average making both very efficient for realtime PCG.

6. EVALUATION

Figure 1 shows a randomly generated level (equivalent to the base step of the CA-based algorithm) next to a level generated the CA-based algorithm. It is apparent that CA generate playable, good-looking maps at a very low computational cost, whereas purely random generation does not. Such an outcome provides the first indication of the efficiency and appropriateness of the algorithm for real-time map generation.

Figure 2 depicts different runs of the CA algorithm with respect to the number of iterations (n), Moore distance sizes (M) and neighborhood value threshold (T) that defines a rock. It can be seen that the algorithm converges quickly to cave-like layouts, the smoothness of which can be varied via the M parameter. The T parameter can have a great impact on the initial ($n = 1$) ratio of rock over floor cells and it is up to the designer to adjust accordingly.

Figure 3 depicts a 3×3 map generated with the proposed algorithm. This map was built in only 349 ms indicating the appropriateness of the CA algorithm for realtime PCG. It is also clear that well-shaped and smooth tunnels are generated very quickly using the CA base-grid interconnection algorithm described in section 3.3.

7. DISCUSSION AND CONCLUSION

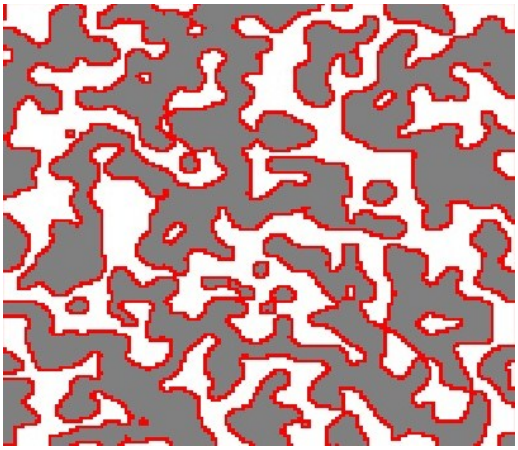


Figure 3: A 3×3 base grid map generated with CA. Rock and wall cells are represented by red and white color respectively. Grey areas represent floor. ($M = 2; T = 13; n = 4; r = 50\%$)

This paper presents a self-organization approach, via iterative cellular automata, for generating infinite 2D cave-maps in realtime. Evaluation of the algorithm in the *Cave Crawler* game level editor shows the computational efficiency of the approach in generating playable tunnels and the variation a level designer has access to by adjusting content parameters.

The main reason for using 2D grids instead of 3D cubes to generate the cave map of the game is that it is easier the keep the floor level this way, which makes connectivity testing much more tractable. The obvious next step is to generate 3D maps based on the 2D cave terrains created by the algorithm proposed here. In order to create a 3D world, one can project the tunnels generated along the y axis. Mid-point displacement algorithms such as diamond square or scatter noise could be then used to generate the walls of the tunnels. Alternatively, a designer could investigate the efficiency of 3D cellular automata for creating 3D maps directly, relying on cubic Moore neighborhoods.

8. REFERENCES

[1] T. Adams. Re: Optimization-based versus "constructive" pcg (post to the "procedural content generation" google group).

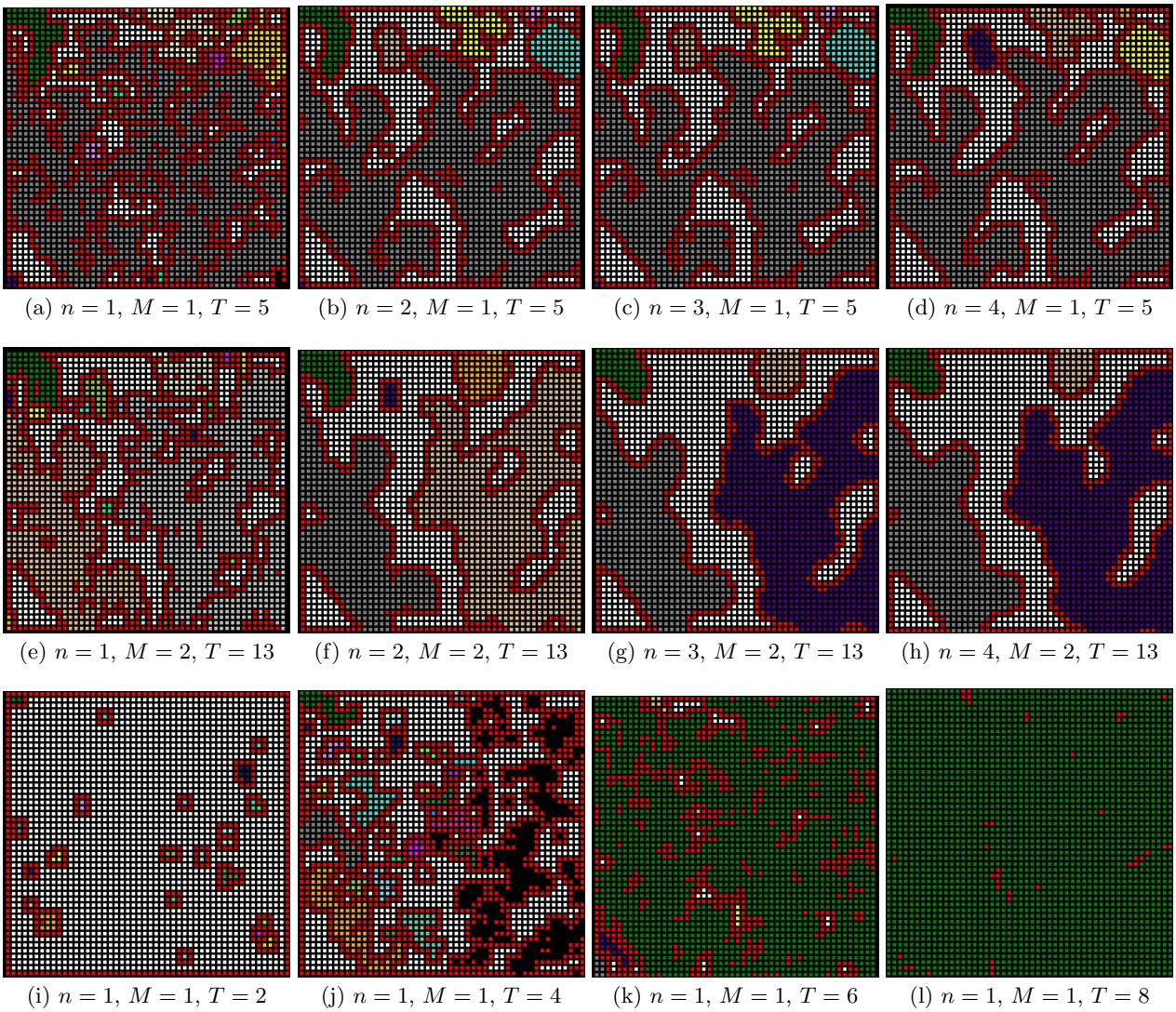
[2] F. Belhadj. Terrain modeling: a constrained fractal model. In *5th International conference on CG, virtual reality, visualisation and interaction in Africa*, pages 197–204. ACM, 2007.

[3] J. Doran and I. Parberry. Controlled Procedural Terrain Generation Using Software Agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2010. to appear.

[4] D. Ebert, K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 3rd edition edition, 2003.

[5] T. Forsyth. *Game Programming Gems 3*, chapter Cellular Automata for Physical Modelling. Charles River Media, Inc., 2002.

[6] M. Frade, F. F. de Vega, and C. Cotta. Evolution of artificial terrains for video games based on



(a) $n = 1$, $M = 1$, $T = 5$ (b) $n = 2$, $M = 1$, $T = 5$ (c) $n = 3$, $M = 1$, $T = 5$ (d) $n = 4$, $M = 1$, $T = 5$

(e) $n = 1$, $M = 2$, $T = 13$ (f) $n = 2$, $M = 2$, $T = 13$ (g) $n = 3$, $M = 2$, $T = 13$ (h) $n = 4$, $M = 2$, $T = 13$

(i) $n = 1$, $M = 1$, $T = 2$ (j) $n = 1$, $M = 1$, $T = 4$ (k) $n = 1$, $M = 1$, $T = 6$ (l) $n = 1$, $M = 1$, $T = 8$

Figure 2: Map evolution over CA iterations (n), Moore distances (M) and rock threshold values (T). The initial percentage of rock cells is 50% in all figures.

accessibility. In *Proceedings of EvoApplications 2010*, volume 6024, LNCS, pages 90–99, Istanbul, 2010. Springer.

[7] T. W. Malone. What makes computer games fun? *Byte*, 6:258–277, 1981.

[8] J. Olsen. Realtime procedural terrain generation. Technical report, Oddlabs, 2004.

[9] N. Sorenson and P. Pasquier. Towards a generic framework for automated video game level creation. In *Proceedings of EvoApplications 2010*, volume 6024, LNCS, pages 130–139, Istanbul, 2010. Springer.

[10] P. Sweetser and J. Wiles. Combining influence maps and cellular automata for reactive game agents. In *Intelligent Data Engineering and Automated Learning — IDEAL 2005*, volume LNCS 3578, pages 524–531. Springer Berlin / Heidelberg, 2005.

[11] 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*, 4(1):1–9, 2005.

[12] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation. In *Proceedings of EvoApplications 2010*, volume 6024, LNCS, pages 140–149, Istanbul, 2010. Springer.