Generating Real-Time Strategy Heightmaps using Cellular Automata

Peter Ziegler
peter.ziegler@stud-mail.uni-wuerzburg.de
Julius-Maximilians-Universität
Würzburg

Sebastian von Mammen
sebastian.von.mammen@uni-wuerzburg.de
Julius-Maximilians-Universität
Würzburg

ABSTRACT

This paper presents a new approach of heightmap generation for Real-Time Strategy games (RTS) based on Cellular Automata (CA) in the context of various established techniques. The proposed approach uses different CA rulesets to generate and modify maps for the RTS game Supreme Commander. To evaluate the quality of the generated maps, a survey was conducted asking 30 participants about map quality compared to user-generated maps. The participants rated the maps more balanced and novel but less aesthetically pleasing. The paper concludes with according future work propositions to improve the presented approach.

CCS CONCEPTS

• **Applied computing** → *Computer games.*

KEYWORDS

cellular automaton, procedural generation, real-time strategy

ACM Reference Format:
Peter Ziegler and Sebastian von Mammen. 2020. Generating Real-Time Strategy Heightmaps using Cellular Automata. In *International Conference on the Foundations of Digital Games (FDG '20), September 15–18, 2020, Bugibba, Malta.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3402942.3402956

1 INTRODUCTION

Real-Time Strategy (RTS) is a sub-genre of strategy video games. In RTS games, players control buildings and units which they can command to generate resources or destroy opponents. Resource management is important for restoring or constructing buildings and mobile units, whereas attack and defense are important to reduce one's losses, to expand one's territorial claims, and eventually to win the game. The playground terrain often poses an important game element as it favors or hinders certain movement or building strategies depending and challenges the players by considering, e.g. elevations or cliffs, impenetrable rocks or slackening marshes. Competitive RTS matches are usually fought on one of a small pool of pre-selected maps. In order to be competitive, players have to memorize viable strategies for each map in the pool. This results in good memory to be the players' most important skill, even though strategy games are meant to be centered around strategic thinking. Of course, if no one has ever seen the map, the focus of competitive play is shifted from memorized strategies to real time strategic thinking. This can be accomplished by means of procedural generation of new maps. In this paper we present an according procedural approach to creating new RTS maps based on CA, exemplary for the RTS game Supreme Commander.

FDG '20, September 15–18, 2020, Bugibba, Malta

ACM ISBN 978-1-4503-8807-8/20/09…$15.00
https://doi.org/10.1145/3402942.3402956

2 RELATED WORK

Frequently deployed techniques in procedural content generation of landscapes are fractal Perlin Noise, Midpoint Displacement and the Diamond Square algorithm [4]. Synthesis based on Perlin Noise, or Simplex Noise [10], often interpolates between the dot products of gradient vectors on a grid [9]. Midpoint Displacement [8] introduces and lowers or elevates vertices in a geometric topology to generate terrains even in real time. As a conceptual extension, the Diamond Square algorithm [13] elevates the midpoint of a square to generate realistic 3D terrains. Code libraries as in [6] and [12] provide fast access to terrain generation techniques as seen in Fig. 1. The displayed heightmaps integrate six height levels and are smoothed by five iterations of thermal erosion.

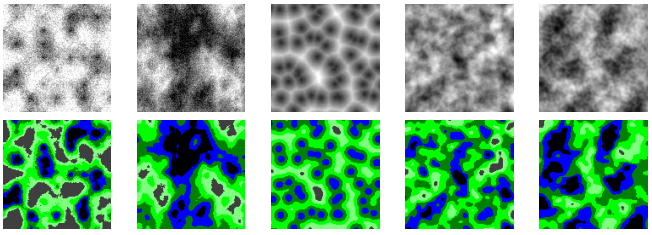
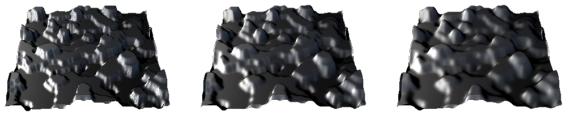


Figure 1: Heightmaps generated with different techniques

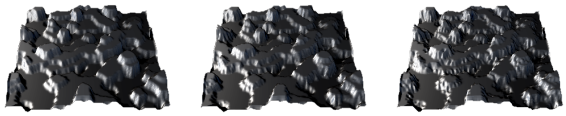
Thermal erosion mimics the effects temperature changes and gravity have on terrain. If a cliff is too steep, material from the top detaches and deposits at the bottom. This algorithm is generally faster than other erosion techniques. Thermal erosion makes the heightmap look smooth and creates rolling hills which can be seen in Figure 2. Hydraulic erosion mimics the effect of rainfall on terrain. Simulated raindrops hit the terrain at random positions. They pick up material and carry it downhill. The material aggregates on the terrain, where the drops come to a stop. This algorithm is slow in comparison to other erosion techniques but results in realistic looking heightmaps which can be seen in Fig. **??**.

Burks and von Neumann developed CA for the purpose of modeling self-organized, cellular procreation by means of a large set



5 Iterations 10 Iterations 15 Iterations

Figure 2: Simulating Multiple Iterations of Thermal Erosion



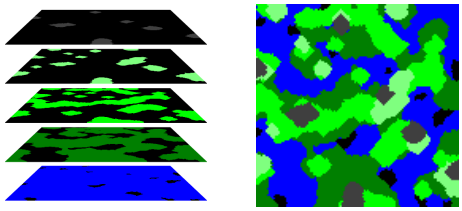
5 Iterations 10 Iterations 15 Iterations

Figure 3: Simulating Multiple Iterations of Hydraulic Erosion

of according finite state machines [2]. As an according abstraction, CA organize a large number of cells with discrete states across a lattice structure that acquire new states based on their and their neighbors' previous states. There are two commonly used definitions for neighborhood on a two-dimensional square grid: The Moor neighborhood considers all the 8 neighbors encircling a cell, whereas von Neumann's original neighborhood definition only considers the four along its edges. A widely known CA is Conway's Game of Life (GoL), discovered by J. H. Conway in 1970 and popularized in Martin Gardner's Scientific American columns. The GoL is a binary CA, i.e. considering two states for a cell, with a Moore neighborhood. The GoL's state transitioning rules consider the amount of inactive/active neighbors. Using the same setup in terms of states, neighborhoods and quantitative instead of explicit rules, [5] present an approach to procedurally generate cave level maps based on local self-organization. Some of the same authors also used CA to generate a map representation for the RTS game Dune II [7]. Yet, to our knowledge, the first represents the only preceding scientifically published approach to be directly used for CA-based heightmap generation. There have been numerous related approaches. Landscape Automata that augment a heightmap with a quadtree structure to hierarchically introduce changes to individual pixels and patches of pixels [1]. Texullar is an application to create textures for large-scale terrains, whereas CA were used to introduce graphical patterns [11]. Various geological models have been researched that retrace real world terrain evolution by means of CA, e.g. [3].

3 CA-BASED TERRAIN GENERATION WITH CHANGING RULESETS

Most RTS maps integrate distinct terrain types such as water, land, plateaus. This could be achieved by applying various filters to a heightmap, but this is slow and leads to results with lots of patterns and artifacts. It also causes one terrain type, e.g. a plateau, to always be surrounded by the next lower terrain type, e.g. land. Generally, creating a large number of viable terrains with distinct features and elevation levels is a challenge. CA can be used to do this efficiently. In most CA, the rule that determines the next generation never changes. We propose to apply several different deterministic or stochastic rules sequentially. In addition to applying an iteration of a specific CA, we also allow to apply a modification to the grid structure itself: In our current implementation, the designer may trigger a subdivision of the whole CA-grid, doubling the resolution in both dimensions. Each cell is simply subdivided into 2 x 2 new cells that inherit the predecessor's state. CA-rules in combination with additional modifiers enable the designer to define concrete generative sequences. In order to create different height levels, multiple CA-grids are stacked on top of each other. As can be seen in Fig. 4, this technique exhibits some directional artifacts while generating very unique maps.



(a) CA Grids (b) Resulting Map Layout

Figure 4: (a) Merging a stack of CA-generated grids (b) yields a heightmap layout.

Our implementation features a 2D lattice grid with four von Neumann-neighbors. Missing neighbors at the edge of the grid are assumed to have the same state as the considered cell. Like in GoL or Johnson's cave generation technique, we only consider the total count of active neighbors. Like in the GoL, we also disregard the position of the neighbors and just count how many neighbors are active. This gives us 10 possible states that a cell and its neighbors might be in. For each of these states, the rule defines whether the cell will be active or inactive in the following generation. Every possible rule can therefore be defined by the 10 booleans determining the resulting states. The stochastic rules work like the deterministic ones but provide a probability for the resulting target cell's state, instead of a deterministic boolean value. For example, if the probability value of a rule is 0.1, there is a 10% chance of the cell being active and a 90% chance of the cell being inactive in the next generation. An example of a stochastic rule is given in Fig. 5. This example rule causes all inactive cells which have one or more active neighbors to turn on with 50% probability. See Fig. 7 for the outcome of its application with intermittent subdivisions, i.e. following the sequence *subdivide* → *example rule* → *subdivide* → *example rule*.

state	off	off	off	off	off	on	on	on	on	on
neighbors	0	1	2	3	4	0	1	2	3	4
probability	0	0.5	0.5	0.5	0.5	1	1	1	1	1

Figure 5: Stochastic rule used by the generator.

state	off	off	off	off	off	on	on	on	on	on
neighbors	0	1	2	3	4	0	1	2	3	4
probability	0	0	0	0	0	0	1 - p	1	1 - p	1

Figure 6: Erode rule used by the generator.

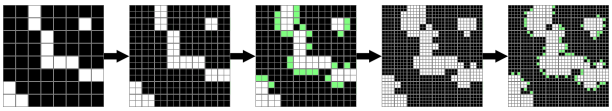


Figure 7: Sequential CA-application with intermittent *subdivision* modifier to create a CA-grid level.

4 RTS MAP GENERATOR

The map generator is comprised of six components: (1) Layout Generation, (2) Erosion Simulation, (3) Marker Generation, (4) Detail Generation, (5) Texturing and (6) Export. The implementation of these components may vary slightly from game to game but should always follow the same structure. All components can generate their output with either point or axis symmetry. This is important to ensure that no team starts with an advantage. We added the exemplary configuration for the game Supreme Commander to the explanations in this section. The layout component generates the map layout. A predefined sequence of rules yields a grid for each terrain layer that determines whether a specific terrain feature is present across the map. These grids are named: SEAFLOOR (blue), LAND (dark green), LOWPLATEAU (green), HIGHPLATEAU (light green) and MOUNTAIN (gray). An exemplary generation process which uses these grids can be seen in Figure 8. Except for mountains which will be introduced in the next step, theses grids will then be converted into a heightmap. If a cell is activated, the height of the layer will be added to the heightmap at this position. The base height of the map is 5.0. The height of SEAFLOOR is 15.0. The height of LAND, LOWPLATEAU and HIGHPLATEAU is 5.0. The water height of the map is 22.0 resulting in all points of the heightmap which are lower than this value to be underwater.

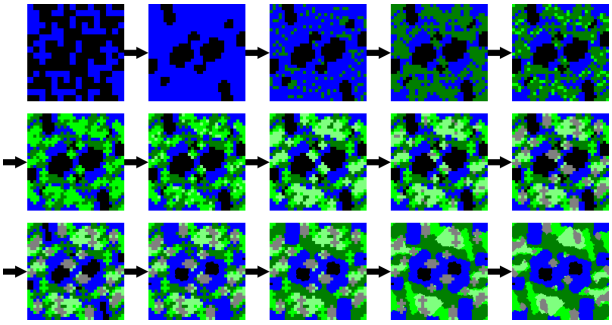


Figure 8: Exemplary heightmap layout generation

To smooth the heightmaps, we run multiple iterations of thermal erosion which results in height differences of less than approximately 7.0 to become passable terrain. The mountains are now generated with a special technique that utilizes the "Erode"-rule, see Fig. 6. This rule relies on a parameter p which determines its impact. Erode is run on the MOUTAIN grid causing some cells to turn off. For every cell that is still on, 0.25 will be added to the heightmap at this position. This is repeated until all cells are off. The technique is faster than using hydraulic erosion but the results look less realistic. Finally, we run a single iteration of thermal erosion. This concludes the heightmap generation. Markers for the players starting positions and resources, as well as details such as rocks and trees are distributed across the map. This is mostly done using simple constraints such as minimum distance and maximum steepness of the terrain. All markers are mirrored to ensure map symmetry. All generated data is then saved in the game's file format.

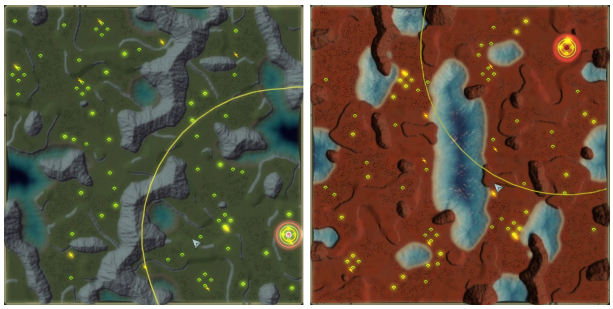


Figure 9: Maps Exported for Supreme Commander

5 EVALUATION

To evaluate the performance of our proposed approach, we compared the playability of generated maps and user created maps. To this end, 30 Supreme Commander players were asked to participate in a study. Experience of the players was the only criterium we considered choosing the participants of the study as it is the sole basis for judging a map's playability. To ensure that all participants are experienced they had to have played over 50 matches against other players (PvP). In the scope of the experiment, they played a three versus three match on a generated map. In this game mode the first team which eliminates all enemy commander units wins. Every match was supervised using the spectator mode to ensure that it was finished without issues like disconnects or lag. If the simulation speed displayed by Supreme Commander (which is 0 per default but can be reduced to −10) dropped below −2 for longer than 10 seconds, it was considered lag and the game was not evaluated. After the game, participants were asked to fill out a questionnaire. First, players had to confirm that they had played over 50 PvP games and that the game on the generated map was finished without issues. They also had to state whether they won or lost the game. After that, we asked two questions each about fun, novelty, aesthetics, balance and whether the players wanted to play these maps in a competitive setting. These questions could be answered with one of five options on a Likert-scale: Strongly Agree, Agree, Undecided, Disagree, Strongly Disagree. The answers for each category are mapped to the interval −2 (favoring user created maps) to 2 (favoring generated maps).

As can be seen in Figure 10, the *fun score* is 0.07 with a standard error of 0.26. Whether the map is generated or user created doesn't

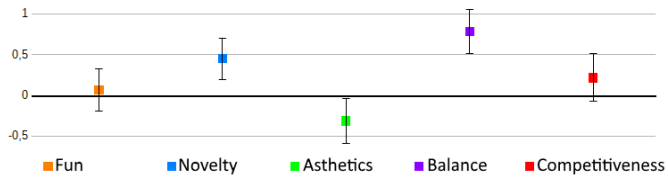


Figure 10: Average Answer Value per Category

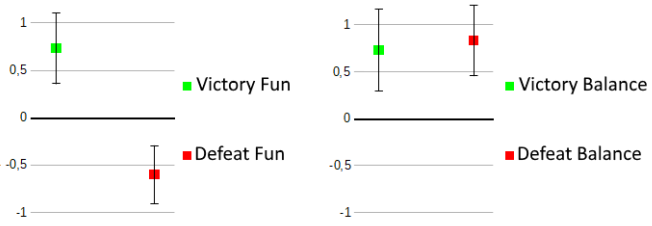


Figure 11: Fun, Balance vs Outcome

impact how much fun the players have. As can be seen in Figure 11, players who win have considerably more fun than players who loose. The fun score for winning players is 0.73 with a standard error of 0.37. The fun score for loosing players is -0.60 with a standard error of 0.30. The novelty score is 0.45 with a standard error of 0.25. Many players could use new strategies while playing on the generated map. The generated terrain often features patterns which are rarely found on user created maps. Players were able to invent new strategies which use these terrain features to create an advantage. One player (played over 1500 PvP games) wrote that the map felt "refreshing". The *aesthetics score* is -0.31 with a standard error of 0.27. Many players thought that the generated map looked worse than user created maps. user created maps often feature hand crafted details which are missing in generated maps. Also the detailing on generated maps is repetitive and the texturing only depends on the steepness of the terrain and doesn't add additional details. The *balance score* is 0.78 with a standard error of 0.27. Many players perceived the generated maps as more balanced than user created maps. All generated maps have perfect point or axis symmetry. Players value this precise symmetry. Players also wrote that the terrain felt well balanced which allowed them to use water, land and air units equally. As can be seen in Figure 13, the Balance rating is not impacted by the outcome of the game. Unlike fun, which was rated much higher if a player won, balance was rated equally high by both winning and loosing players. The balance score for winning players is 0.73 with a standard error of 0.43. The balance score for loosing players is 0.83 with a standard error of 0.37. The *competitiveness score* is 0.21 with a standard error of 0.29. Some players want to play generated maps in ranked matches against other players. Because the standard error is higher than the score we can't assume that a majority of players thinks the same way. When asked why they don't want to play the maps in ranked matches most players wished for aesthetic improvements. However, the correlation between aesthetics and competitiveness is quite low at 0.25.

6 FUTURE WORK AND CONCLUSION

There are multiple ways in which the proposed map generation technique could be improved. Different from the proposed CA technique, rules which consider the Moore neighborhood or an entirely different neighborhood can be explored. Other neighborhood definitions could result in performance improvement, simplification of the map generation process or new cell patterns. Similarly, more than two possible cell states could enable the map generation to take place on a single grid. Although this most likely wouldn't affect the performance, it may result in new interesting patterns which consider different layers at once. If the CA was extended to continuous states, one could work directly on a heightmap. Subdivision could be extended to allow cells with different sizes to exist on the same grid, to increase the level of detail only where needed and thereby, potentially, to much greater magnitude without loosing more memory. The CA-based approach to heightmap generation presented in this paper shows that CA can generate terrain features which are difficult to obtain with other established heightmap generation techniques. The generated maps feature unique terrain with compelling and diverse gameplay. They allow players to create new strategies and can be used as a balanced alternative to user created maps. Unlimited amounts of maps can be created on demand to make every match an entirely new challenge. When map generation is implemented into an RTS game, players are no longer forced to memorize viable strategies for each map. This shifts the focus of Real Time Strategy games from memorization to the mastery of strategic thinking.

REFERENCES

[1] Daniel Ashlock and Cameron McGuinness. 2013. Landscape automata for search based procedural content generation. In *IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, Niagara Falls, ON, Canada, 1–8.
[2] Arthur W Burks and John Von Neumann. 1966. Theory of self-reproducing automata.
[3] Min Cao, Guo'an Tang, Fang Zhang, and Jianyi Yang. 2013. A cellular automata model for simulating the evolution of positive–negative terrains in a small loess watershed. *International Journal of Geographical Information Science* 27, 7 (2013), 1349–1363.
[4] Jonas Freiknecht and Wolfgang Effelsberg. 2017. A Survey on the Procedural Generation of Virtual Worlds. *Multimodal Technologies and Interaction* 1, 4 (2017). https://doi.org/10.3390/mti1040027
[5] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. 2010. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM, 1–4.
[6] Martin Kahoun. 2013. *Realtime library for procedural generation and rendering of terrains*. Diploma Thesis. Univerzita Karlova.
[7] Tobias Mahlmann, Julian Togelius, and Georgios N Yannakakis. 2012. Spicing up map generation. In *European Conference on the Applications of Evolutionary Computation*. Springer Berlin Heidelberg, Malaga, Spain, 224–233.
[8] Ivo Marak. 1996. Random midpoint displacement method. http://old.cescg.org/CESCG97/marak/node3.html. Accessed: 2019-11-13.
[9] Ken Perlin. 1985. An Image Synthesizer. *SIGGRAPH Comput. Graph.* 19, 3 (July 1985), 287–296. https://doi.org/10.1145/325165.325247
[10] Ken Perlin. 2001. Noise hardware. Real-Time Shading SIGGRAPH Course Notes.
[11] Swapnil Sinvhal. 2007. *Mapping textures on 3D Terrains: A hybrid cellular automata approach*. Ph.D. Dissertation. Texas A&M University.
[12] Patricio Gonzalez Vivo and Jen Lowe. Last Accessed: 2019-11-21. Chapter Noise in The Book of Shaders. https://thebookofshaders.com/11/.
[13] H. Wang, W. Chen, X. Liu, and B. Dong. 2010. An improving algorithm for generating real sense terrain and parameter analysis based on fractal. In *2010 International Conference on Machine Learning and Cybernetics*, Vol. 2. IEEE, 686–691. https://doi.org/10.1109/ICMLC.2010.5580560