Liam Liden

Leonard Maxim

Procedural Content Generation

3 May 2020

Cellular Automata Dungeon Generation Report

Cellular automata have several uses in procedural content generation. One topic they are especially useful in is generation of dungeons. This is because cellular automata can generate cave systems that look natural when compared to the structured results of algorithms like binary space partitioning. Cellular automata can even be expanded into spawning items into dungeons. This project utilized cellular automata to create a system to give Unity developers the functionality of cave and item generation with customized parameters.

The algorithm followed the basic technique outlined in *Procedural Content Generation in Games* (2016) and added some additional Unity features. A map of user specified with and height is initiated with randomly placed walls at each coordinate. The probability of each cell being a wall is input by the user (with a default value of 50%). The algorithm then proceeds to visit each cell, calculate the Moores neighborhood (where a wall has the value of 1 and floor has a value of 0), and determine if the cell should be a wall based on the input wall threshold. To prevent the player from leaving the play area, the value of a cell outside of the map is equal to the wall threshold (this will guarantee a placement of a wall on the border of the map). This process occurs for a user input number of iterations. Higher iterations tend to create more open spaces and less narrow tunnels.

After the base generation process finishes the map will often look like several unconnected rooms with some possible windy passages. To guarantee that the player can reach all rooms in the level, a connection algorithm is used. This algorithm uses several smaller algorithms to achieve connectivity between rooms. First, a fill algorithm is used to visit all connected floor cells and create objects of the class ‘Room’. ‘Room’ keeps track of the edge cells and other room connected to it. Once all rooms are created, the room with the largest amount of edge cells is found and designated as the main room. This room is the room which all other rooms will attempt to connect to. Each room attempts to connect to the nearest room until it connects to the main room. The A-Star search algorithm is used to find the shortest path to the nearest unconnected room. A straight-line distance is used as the heuristic to inform and speed up the search process. Connections are made on the map by setting cells, with a user defined width, to floors.

After guaranteeing that all rooms on the map can be reached by the player, the system begins furnishing. Furniture is user defined using custom rules and basic operations. Examples of the user's choices include neighborhood type, the target value of the neighborhood, and the value of the item itself. The algorithm will visit a random sequence of floor cells and test if any furniture meets the requirement to be spawned. Like the cave generation algorithm, all cell values are updated simultaneously between iterations. This method is similar to Green et al. in *Two-step Constructive Approaches for Dungeon Generation* (2019).

The algorithm has a high level of customizability. As seen in Figures 1 through 3, a change in iterations can create a large difference in the shape of the generated caves. Connection size adjustments also create vastly different looking cave systems. A connection size of 1 (connections are created by groups of 9 cells) is usually more natural looking, but by switching to a connection size of 0 (Figure 4) could be useful to developers. Values above 1 tend to create too much open space but could be a niche use. The furniture system was also successful in allowing a large amount of customizability. A vast variety of furniture can be created. Examples included in the project included goblins (spawn if cell is surrounded by 3 or more walls), potions (spawned if Moores neighborhood value is greater than or equal to 6, which occurs often near enemies), and bandits (appear when Moores neighborhood is a value of 0).

Additional enhancements would be oriented towards increasing human interaction with the system and embracing mixed content initiative. One idea is allowing the developer to change individual cells to different values and see changes real time. This is possible in Unity but was too time intensive to include in this project. Other features could also be added to the furniture system. The types of spawning rules that could be included is nearly limitless. Using the base system, many more parameters could be added to achieve rules developers need.

Appendix

Figure 1: Iterations 3

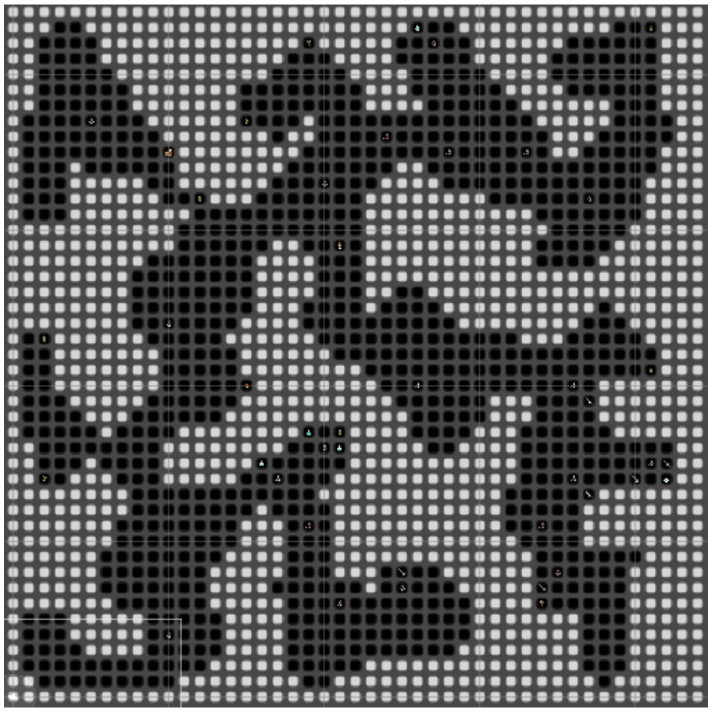


Figure 2: Iterations 5

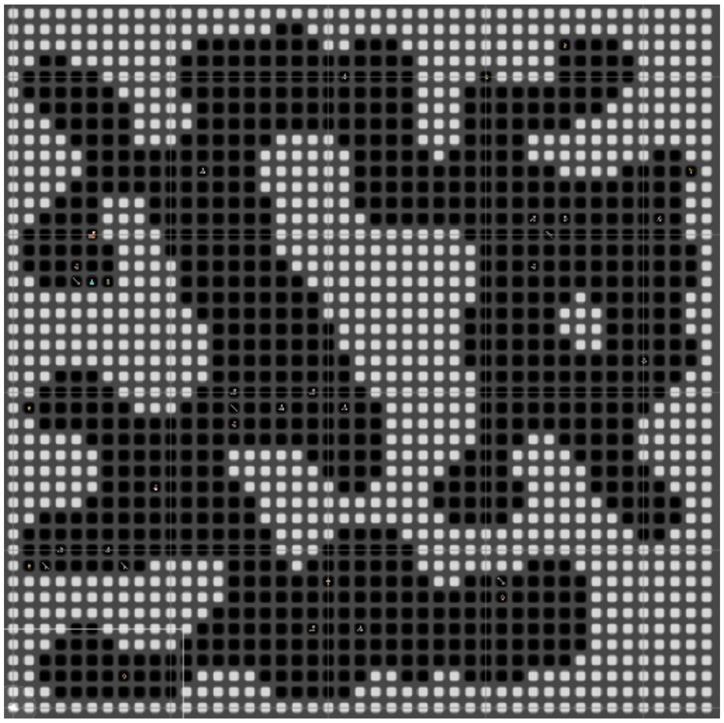


Figure 3: Iterations 7

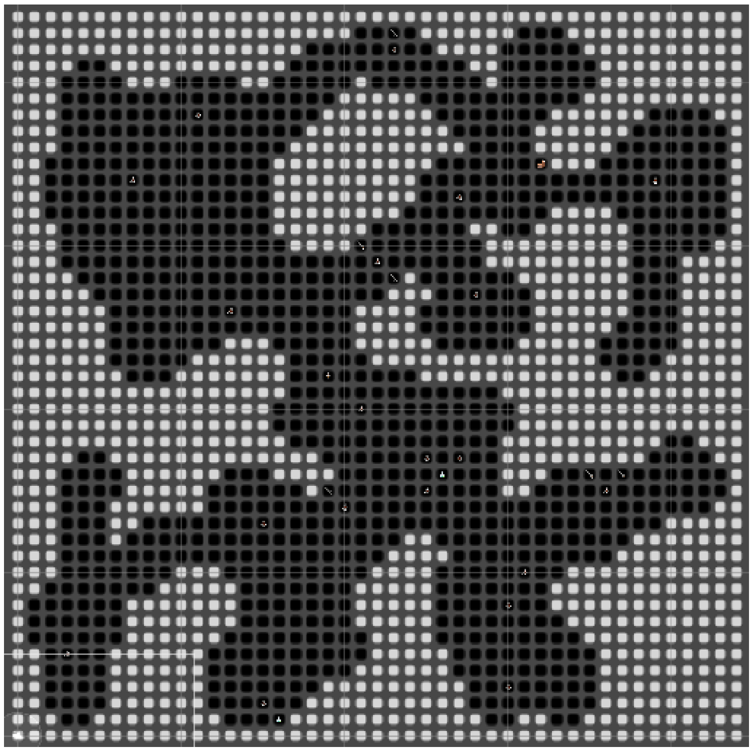
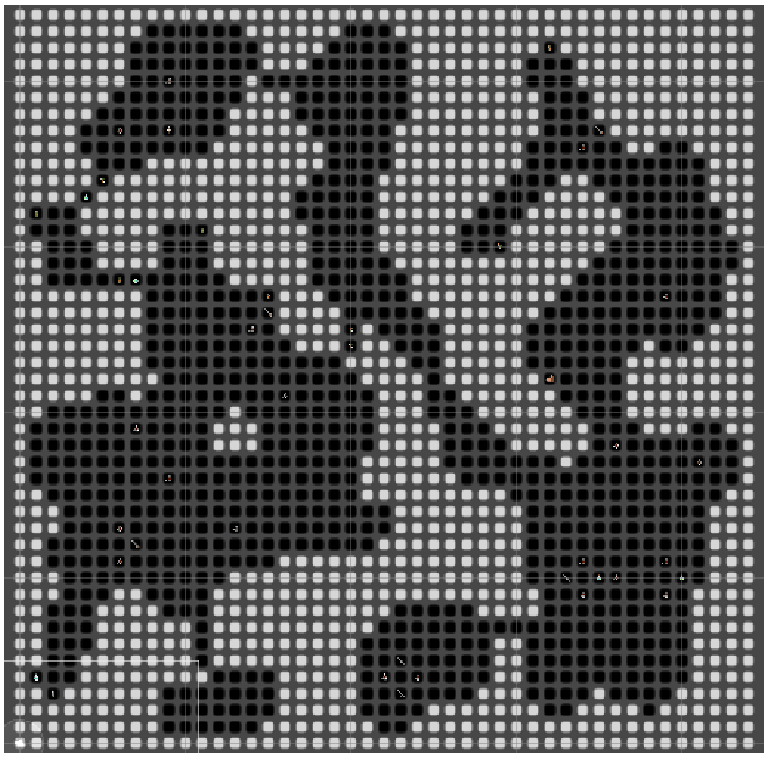


Figure 4: Connection Size 0



Works Cited

Michael Green et al. (2019). *Two-step Constructive Approaches for Dungeon Generation*. In FDG ’19: Procedural Content Generation Workshop, August 26-30, 2019, San Luis Obispo, CA.ACM, New York, NY,USA, 7 pages. https://doi.org/10.1145/1122445.1122456.

Noor Shaker, Julian Togelius, and Mark J. Nelson (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. ISBN 978-3-319-42714-0.