HOGESCHOOL VAN AMSTERDAM

Procedural Cave Generation

Project Game Technology

Jasper Meier, 500703267

February 2017

Glenn Hoff, Wouter van de Hauw, Michael Fuents Rodriguez, Rosa Corstjens, Guus van Gend, Kevin de Leeuw, Frederick van der Meulen & Jasper Meier

Project Game Technology
Team 5

Under guidance of
Alexander Mulder, Product Owner
Anders Bouwer, Scrum coach
Eric Klok, Technisch consultant

Procedural Cave Generation

Project Game Technology

Jasper Meier, 500703267
Game Technology
Hogeschool van Amsterdam
February 2017

ABSTRACT
In this research paper, multiple methods of quest generation will be compared, based on their usefulness for Project Game Technology.

Three distinct methods will be compared with each other. The minimum requirements for any of these methods are:

- The method can be used during runtime;
- The method can generate a 3D cave system, it also should support different parameters and not just a random seed;
- The system supports the placement of items or enemies, after it has been generated;
- The method includes height differences, whether this is during or after the cave generation;
- The generated caves must be saved, the proposed method supports this.

1. INTRODUCTION
The need for procedural level generation exists for a few reasons. The first reason is the need for an inexhaustible map. Real-time level generation ensures this. Having an inexhaustible map also adds to the replayability, because each game has a uniquely created map. Seeds can also be used to save a map, or remake it in a new game. Secondly some games, including Minecraft, rely on procedural generation as their core mechanic, especially games in which exploration is an important factor. Finally, content of the game will be saved very efficient, which will prevent the program from becoming extremely big. PCG also eliminates a big part of the need for designers.

The game, for which this research is written, Diabro, will feature a complete procedural cave system, filled with NPC's and enemies. The player can complete quests, which are also procedurally generated, to advance his level and gain new items. Furthermore the game is about exploration and influencing the world around you. Because Diabro only features cave generation, Only caves will be discussed as form for the level.
The three methods of (cave-like) level generation discussed in this paper are:

- Cellular automata (CA);
- Gluing together prefabricated elements;
- A new proposed graph based method.

This paper will discuss each of the methods separately. After this a method, which will be discussed in the conclusion, will be chosen.

2. CELLULAR AUTOMATA
A cellular automaton, or CA, is a program which simulates single cell based life, by implementing a set of rules (L. Johnson, et al, 2010). The cells change their state after each iteration of the program. This way cells form shapes by clotting together. Two rules which are always included are: the rule of cell death; and the rule of recreation of cells. This ensures life doesn't disappear, or stop moving after a small amount of iterations.

Adding further rules would make the cells behave different and possibly even more controllable, but would also make the program increasingly more complex. Barr et al. uses a cellular automaton which enables the cells to release chemicals, which attract or repel other cells (1995).

2.1. GENERATION
Because all the generation must be done in real-time, the method of generation must be very efficient and optimized. This ensures the target speed and framerate can be achieved more easily. Furthermore, to also implement the function of procedural quests, the generator has to be able to place and identify certain locations within the generated map.

The generation process is started by declaring and filling a square field of cells (each tile is a cell). The width and height are the maximum width and height for the part that needs to be generated. The cells are either alive or dead, extra states could also be added, but this also raises the required processing power. Within level generation a live and dead cell are often represented by a wall or floor tile, respectively.

If a dead cell, or a wall element, is surrounded by floor once the program is finished, you can either remove the block, connect it to excising walls or replace the block with a rock or another solid object.

2.2.RULES

The CA, used by Johnson, Yannakiks and Togelius, has parameters which can be used to influence the creation of caves(2010). First, they have implemented a *Moore distance size*, which can be used to influence the smoothness of the different rooms. Furthermore, there is a *neighborhood threshold*, which changes the amount of rocks created. The last parameter defines the amount of iterations and is called *n.*

The rules all cells will live by need to be defined and further researched. This will ensure the quality and controllability of your generated levels.

2.3.CONNECTING TO EXISTING PATHS

Because the newly generated paths have to connect with the current ones the cell generation is performed with a seed. By reusing the seed the same part of a dungeon, and the parts that lie behind it, can be regenerated.

3. PREFABRICATED ELEMENTS

In some games levels are generated by placing prefabricated blocks of geometry next to each other, like in Diablo 3 or The Binding of Isaac. This ensures your level design will always remain at a certain quality. Furthermore, this method is very simplistic and requires less processing power, but more system memory. One con of this method is the creation of modules for the dungeon, a designer must still design all these parts.

3.1. MODULES

The so called modules are represented by a 3D model, furthermore they have connection points and a flag type. The connection points are locations on the model, consisting of a position and rotation, which enable other modules to connect and align with them. The flag type is used to define the type of a model. Seredynski researched a generation model which used three flags types for its modules: path, junction and room (2014).
In this model a pathway can connect with any flag type except for pathway, the same applies for junction, with any flag type except for junction. The room can only connect with a pathway.

3.2. GENERATION

Just creating and placing modules is just a small part of the overall level generation. The overall design of the game should be good, pathways should loop around and not just turn op as a dead end, pathways, rooms and junctions should not collide, unless they are connected and smaller details like rocks and torches should also be designed.

To ensure the modules don't collide a collision detection system must be implemented. Once a collision is detected the last placed module is replaced by another model with the same flag type. If all these options don't work either the second to last part is deleted and replaced with a new part, or a new flag type could be tested.

3.3. SMALL OBJECTS

Within the caves, cities and or houses itself new connection points are created. These connection points within the objects can then be used to place items as torches, tables or treasure chests upon. Once again the correct placement of items is handled by flags types, which allow certain items to be placed on certain locations.

3.4. SHORTCOMMINGS

This method offers a nice solution to the generation of dungeons, but it has its shortcomings. The method still requires every possible room to be designed. Another problem is assigning connection points. This has to be done for each room manually. Finally the rooms are able to connect and form loops, but this is a part in which the small differences in lengths in different parts could make this very complicated to implement in a procedural method.

The method does offer a way of connecting the dungeon and not be limited to 90 degree angles, however the CA method does offer this as well.
Although the method requires design, it also offers a great way to have a lot more control over the looks of your game. I would not recommend the use of this algorithm, because it might repeat in an infinite game, unless a lot of different modules are created.

Furthermore the level design and connection difficulties, could retain the design of your levels. Designing bigger modules, so certain puzzles or design elements can be implemented, can cause the caves to look very similar.

4. GRAPH-BASED METHOD

A new hypothetical method has been composed during this research. This method exist of two main parts: The graph-based level generator and a cellular automaton for cave and pathway generation.

4.1. GRAPH-BASED LEVEL GENERATION

The graph-based generation creates a virtual representation for the level. This representation exists completely of nodes (cities, enemy hideouts, etc.) and connections (caves), between these nodes. Intersection between caves can be created with an extra node flag type, but this will be discussed in chapter 4.3.

In order to detect a player, before he reaches the edge of the generated level, zones will be designed. These zones will be square blocks, which cover the whole map. Once the player comes near the edge of the generated terrain, the program will generate a new map. This way a player is unable to reach the edge of the world. The zones are also used to unload generated terrain (if the player gets too far away).

If a part of the generated terrain has to be blocked for a reason, terrain with a level variable could be generated. The connection between the first higher level node and the normal node will be blocked off by a door. The key can be given to the player at an appropriate time, for example as a quest reward after the player reaches level 5.

4.2. CELLULAR CAVE GENERATION

The earlier mentioned method of generation, with a CA, can be used for the generation of caves. Because the caves all move from a node to another, the algorithm just has to move from A to B. Because the nodes are at the end, the only requirement for the algorithm is that it doesn't become wider than the node, at the point they connect.

The rocks and small items throughout the caves will be placed by the location of dead cells. Normally these will indicate a wall, however the cells within the caves can be replaced with smaller and bigger objects.

Another method which could be used to place small items, is the cellular automaton. It can be reused on the generated levels to generate another layer of cells. These cells can be generated with a new set of rules, complying to the needs for objects.

4.3. NODES

The nodes will contain a position, a flag type and an ID. The ID is used to identify all the rooms and after loading a game, the ID's can be used to reconstruct the level. The position is used for placement. Finally the flag type is used to assign functions or types to nodes. The flag types for rooms are the following:

- City, this is a safe-space for the player, where he can buy items or find quests;
- Treasure room, this rooms spawns near bosses and is filled with treasure chest. It is a way to reward the player;
- Junction, this room flag type enables connections to cross, without writing extra generation code;
- Enemy hideout, these spots are filled with enemies and are used for quests. The player might also just raid an enemy hideout for fun of course.

The junction flag type is used to connect multiple connections and create a crossing. These rooms are simple and could just be empty or contain a small square or post in the middle.

4.4. CONNECTIONS

In order to connect the nodes and to give the overall cave system an realistic look, connections will be drawn in-between nodes. As mentioned before, these connections will be drawn out by a cellular automaton.

Because saving the whole cave system is very hard and requires a lot of memory. This is why the zones are used to track the player. Once the player reaches a certain distance the generated connections are unloaded. This part of the map will only exist in a virtual model (the graph model). All programs could, if necessary, still exist and run in this model, although the 3D models and textures are not needed for now.

4.5. FLAGS

Flags are assigned to rooms, so the program is able to assign certain functions to rooms. The generation of the nodes and connections can be steered by selecting these flags.

Another feature that could be added is the earlier mentioned level system. This is another node variable, which indicates a certain permission (item) is needed to enter the node.

4.6. SAVING A DUNGEON

To enable saving and ensure the level doesn't have to regenerate itself every time the game gets loaded, the nodes and connections have certain variables, which describe the level.

All connections will get a seed, which can be used to regenerate the pathway in-between nodes. The seed will also ensure pathways could be unloaded once the player reaches a certain distance from it. This depends on the need for system memory within the game.

If the continuous generation costs too much CPU power, the caves have to be saved in another way. The preferred way is to save the level in the form of a 2 dimensional grid. This is done by multiple generation methods (Johnson, et al. 2010; Barr, et al. 1995). The 2D map will get a generated height to simulate the effect of a 3D map.

CONCLUSION

The proposed method, graph-based level generation is easily tested. This is because the graph and CA layers are completely separated. The nodes can then be filled in by the city generator.

Because the CA has already proven itself in working in an infinite game world, and the graphs won't require much power (this is because they don't have a visual representation and only contain a small amount of variables), the method will be able to render the infinite game world.

One point that can't be overlooked is the unloading and loading of the caves. This can't be a continuous thing, because the player could then crash the game by (accidentally) zigzagging across a zone border. This problem must be addressed while implementing this system.

REFERENCES

Johnson, L., Yannakiks, G. N., & Togelius, J. (2010, 18 juni). Cellular automata for real-time generation of infinite cave levels. Geraadpleegd van http://www.itu.dk/people/yannakakis/a7-Johnson.pdf

Barr, A.H., Currin, B.L., Fleischer, K.W. & Laidlaw, D.H. (1995). Cellular Texture Generation. Opgehaald van: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.8241&rep=rep1&type=pdf

<reference list APA>