T.C.
GEBZE TECHNICAL UNIVERSITY

COMPUTER SCIENCE AND ENGINEERING

Procedural Map Generation

Azmi Utku Sezgin

ADVISOR
Prof. Dr. F. Erdoğan SEVİLGEN

January, 2020
Gebze, KOCAELİ

This project was approved by juries mentioned below on 25/11/2019 as Computer Science and Engineering Graduation Project

Graduation Project Jury

Danışman Adı	Prof. Dr. Fatih Erdoğan SEVİLGEN	
Üniversite	Gebze Technical University	
Fakülte	Computer Science and Engineering	

Jüri Adı	Doç. Dr. Erchan APTOULA	
Üniversite	Gebze Technical University	
Fakülte	Computer Science and Engineering	

Jüri Adı	Doç. Dr. Hasari ÇELEBİ	
Üniversite	Gebze Technical University	
Fakülte	Computer Science and Engineering	

PREFACE

This project has been in my mind since the day I started this school and I want to thank everyone who has contributed in this project and my sincere gratitude to Prof. Dr. F. Erdoğan SEVİLGEN for making this possible and guiding me throughout the whole semester and helping me become a better engineer.

I also want to thank my family for supporting me and my teachers for being such a good role model and give my sincere regards.

January 2020 **Azmi Utku Sezgin**

TABLE OF CONTENT

PREFACE……IV

TABLE OF CONTEN……IV

FIGURE LIST……VI
ABBREVIATION……..VI
SUMMARY……VII

1. **INTRODUCTION …….1**
2. **PROJECT DESIGN ……..2**
   - **2.1) GENERATING THE LAYOUT…….3**
   - **2.2) POPULATING THE LAYOUT …….3**
   - **2.3) PARAMETERS OF THE GENERATOR……5**
3. **IMPLEMENTATION DETAILS ……6**
   - **3.1) THE LAYOUT……6**
   - **3.2) SOLVING THE POINTLESS AREAS……10**
4. **TESTS AND RESULTS……12**
5. **CONCLUSION……..16**
6. **SUCCESS CRITERIA……..16**
7. **EQUATIONS…….17**
8. **REFERENCES……..18**

FIGURE LIST

Figure 2.0 Graph Representation With pointless Areas.....3
Figure 2.1 Parameters for the generator.....5
Figure 3.0 Rectangle divided into multiple randomly sized rectangles.....4
Figure 3.1 Path found by A* Search Algorithm.....5
Figure 3.2 Finalized version of layout after merge.....6
Figure 4.1 Example run with parameters on Figure 4.3.....8
Figure 4.2 Pointless Areas marked in map on Figure 4.1.....9
Figure 4.3 Parameters for the map on Figure 4.1.....9
Figure 4.4 Example run with parameters on Figure 4.6.....9
Figure 4.5 Pointless Areas marked in map on Figure 4.4.....9
Figure 4.6 Parameters for the map on Figure 4.8.....9
Figure 4.7 Example run with parameters on Figure 4.8.....11
Figure 4.8 Parameters for the map on Figure 4.7.....11
Figure 4.9 Another angle for the map on Figure 4.7.....11
Figure 6.0 Example for quality level gaps of rewards.....13

ABBREVIATION

UE4 **:** Unreal Engine 4
PMG **:** Procedural Map Generation
ARPG **:** Action Roleplaying Game

ABSTRACT

The genre of dungeon games, or first-person shooter games as they are more commonly known, has emerged over the last ten years to become one of the most popular types of computer game. At present, the levels in this type of game are generated manually, which is a very expensive and time-consuming process for games companies.

If levels could be generated automatically then this would not only reduce development costs, but allow levels to be generated at run-time, giving game players new playing experience each time a game was played and greatly improving the replayability of the game.

This project is a tool for generating random levels for the games that are developed using UE4 game engine. The main goal is to increase the replayability of the Dunegon Crawler type ARPG games and give the players new experience everytime they play the game and give the developers flexibility of playing with the parameters to generate levels that are more suitable to their games.

1. INTRODUCTION

In the rapidly changing world of computer games, game players are becoming ever more demanding and new hardware technology is constantly pushing back the limits of what can be achieved in games. This has led software developers to produce games that are more complicated and with better graphics than ever before.

Due to this greater complexity, increasing time and effort must be put into levels, which are the units that make up a game. It is estimated that it takes a development team a year to produce a game with forty hours worth of content. This is likely to grow as more features are added to games and as the graphics improve.

It would therefore be desirable if a system could be developed that can take parameters as a set of rules defining properties of the levels that must be produced for a specific game, and automatically create levels for that game.

Chapter 2 describes the steps of generation of the map in detail, parameters and their impacts on the generated map.

2. PROJECT DESIGN

There are two major steps involved in the process of generating the map here. First one is creating a layout from randomly sized and placed islands and coridors that connect an island to another. The second step is to populate these islands with rewards, monsters and events.

What makes a map good map is a really broad question, there are so many parameters and everyone has their own preferences. There's no secret formula to generate the best maps.

In terms of generation a fun map or level, this project focuses on generating layouts with multiple paths from start to end and dead ends along the way. This is one way to explore the idea of a good map. Multiple paths mean giving the player the choice of multiple possibilities of ending the level, more areas to explore, more risks and more rewards. Single path means linear progression with no options which can decrease the replayability of the game.

Another important thing is to encourage players to explore those dead ends, other paths. This is important from both players and developers point of view. Player would want to explore other paths to gain rewards, increase their chance of beating the game by getting stronger. Developer would want player to experience the events they designed, beat the bosses they worked so hard on, use the weapons they carefully balanced for players to use. In short the idea of a good level for this generator is, the higher ratio of pointless islands to all islands the better level.

2.1. GENERATING THE LAYOUT

In the process of generating the layout, first it needs 4 parameters. Height and width of a rectangle, maximum size of the islands.

It divides rectangle into multiple randomly sized rectangles by first dividing the rectangle into 1x1 squares and then merging those squares with their neighbors randomly sized between 1 and maximum room height/width, which the rectangles the islands and then it stores them inside an undirected graph where all the rectangles have an edge between their neighbors.

After that it randomly selects start and end islands from opposite sides of the rectangle the random number is between 0 and ten percent of the rectangle's height and width. For example if the height and width of rectangle is 20 then the start and end islands' locations are between 0 and 2 and then it applies a modified version of A* Search Algorithm to find a path from start to end islands. The islands within that path merge with their neighbors which they will be branches with length of 0, the root of branches and from those roots, will merge with their neighbors again and again to extend the length of the branches.

See Implementation details section (3.1) for more info on the generation of layout and the modified A* algorithm.

2.2. POPULATING THE LAYOUT

There are 3 options to populating an island. Monster, event or reward. Island can be mixed with any number of either of these or empty. The pointless area mentioned earlier means there's no reason for player to explore that area. The player can finish the level without exploring them, they are outside of the path from start to end. These areas are marked during the generation of the layout. Every room that is merged with sidepath merge are marked as pointless areas.

Monsters with good loots, events with awards and rewards can convert a pointless area into a valuable one, one that worth exploring. Placing a good reward at the end of a long dead end would make the whole path valuable/non-pointless as the player would want to explore that area in order to increase their chance of completing the level or the game. (Pseudocode at Section 3.2)

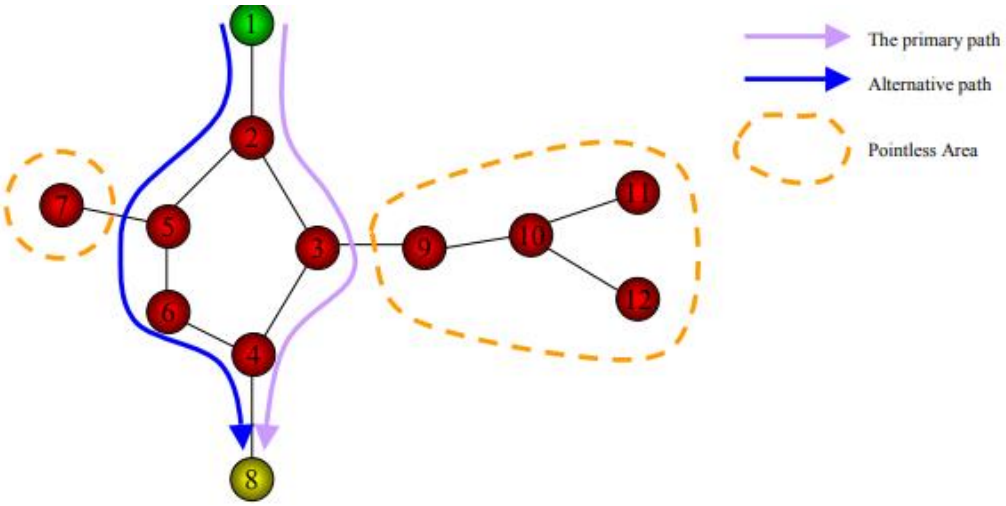


Figure 2.0 Graph representation of an example run with pointless areas [2]

The number and difficulty of monsters or the difficulty of an event to be placed on an island is directly proportional to the size of the island. The bigger the island, the more difficult events, monsters and the harder the path the better the reward at the end of the path.

Combining this kind of level generator with a game design such that the difficulty of the game gets harder as the game goes on and puts up harder and harder challenges in front of you via variety of monsters and events and rewards that will make you stronger will give players to explore those not-so-pointless-anymore areas.[6]

2.3. PARAMETERS OF THE GENERATOR

Figure 2.1 Parameters for the generator

Rectangle Height&Width: Initial size of the map.

Max Island Height&Width: Maximum size of the islands.

Max Dead End Count: Number of dead ends without rewards.

Pointless Area Threshold: Minimum length of path that is considered pointless.

Merge Chance: Chance of islands from single layout merging with neighbors.

Side Path Merge Chance: Chance of merged islands merging with their neighbors.

Island Gap: Gap between islands.

Island Scale: Size of islands.

Event Proc Chance: Chance of placing an event.

The rectangle height and width are the size of initial rectangle to be divided, the bigger rectangle, the more islands in the level to be used. Increasing the size of the initial rectangle will increase the number of nodes in the initial path, which means there will be more islands to merge hence the overall size and branches in the layout will increase hence the complexity will as well.

The other things that affects the size and complexity are merge chance and side path merge chance. First one is responsible of chance of having a branch from a node within the single path. The higher this chance, the more branches and dead ends. What side path merge chance affects is the length of those branches, after merging the islands from single path with merge chance, every merged island from there will be merging and branching with side path merge. Chapter 3.1 describes how the branching is implemented.

The island gap is the distance between islands on the level and islanc scale is the scale of the islands. UE4 uses metric system and the size of a cube with a scale of 1 is 100m so the island of 1x1 sizes with scale 1 will be 100x100. Changing the scale of the island will require the islang gap to be changed accordingly as well.

Event proc chance is the chance of placing a event instead of reward while solving the pointless areas. See Chapter 3.2 for the process of solving pointless areas in more detail.

3. IMPLEMENTATION DETAILS

3.1. THE LAYOUT

The main goal of generating the layout is generating a layout that has multiple paths from start to end with multiple dead-end branches. So that the players can have multiple options and forced to make decisions.
The first steps are pretty straightforward,

- Divide rectangle into one unit sized rectangles. (Figure 3.0).
- For each cell in the grid that hasn't been merged, randomly assign a height and width between 1 and given maximum island size, fix any potential overlapping issues with grid borders or other merged cells by adjusting sizes, merge the cell with other nearby cells to reach the assigned size.
- Pick a random start and end islands.
- Store the islands into a graph.

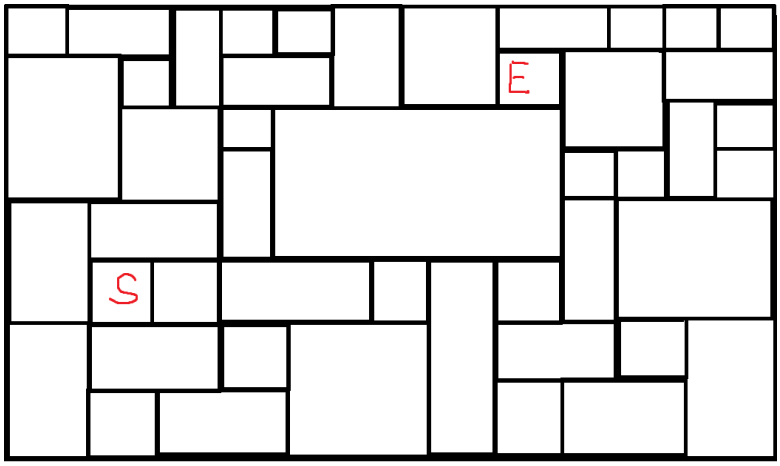


Figure 3.0 Rectangle divided into multiple rectangles [1]

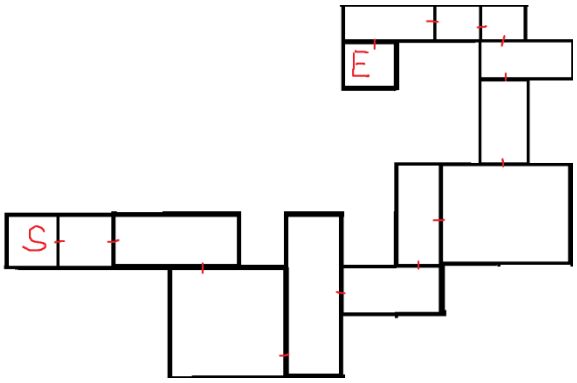


Figure 3.1 Path found by Modified A*

After storing the islands on a graph, in order to find a path from start to end, first method that was tried is A* Search Algorithm, but the paths from that algorithm were too short as the objective of A* is finding the shortest path from source to destination.

The modified version has a small difference, instead of expanding the closest island to the end island first, it picks a random island from 3 closest islands to the end island. Which is the step 3.a on the following pseudo-code.

1. Initialize the open list

2. Initialize the closed list put the starting node on the open list (you can leave its f at zero)

3. while the open list is not empty

a) find the node with the least f on the open list, call it "q"

a) find 3 nodes with the least f on the open list call it "q" (modified version)

b) pop q off the open list

c) generate q's 8 successors and set their parents to q

d) for each successor

i) if successor is the goal, stop search

successor.g = q.g + distance between successor and q

successor.h = Euclidian Distance

successor.f = successor.g + successor.h

ii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

iii) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

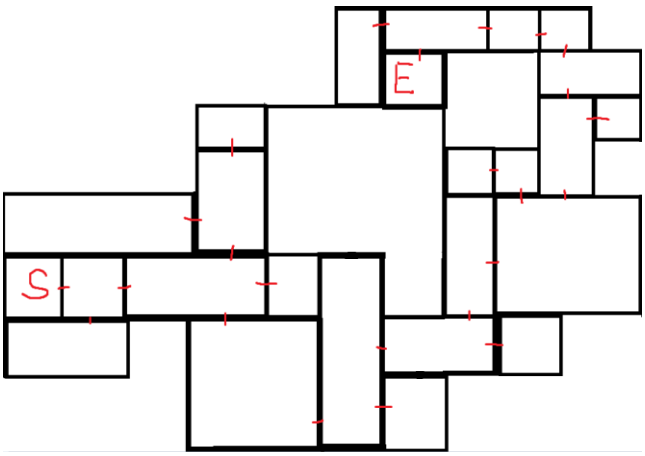


Figure 3.2 Example finalized version of layout after merge

After finding a path between start and end islands, the islands within that path merges with a chance with their neighbors this chance is a parameter of the generator. The merged islands stored in a queue and until the queue is empty the merged rooms have a chance to merge with their neighbors, until the queue is empty. The chance of these islands merging is another parameter of the generator and any island that is merged from now will be marked as pointless area as these islands will not be a part of the main path from start to end and player will not have any reason to explore those islands in order to finish the generated map. These pointless areas will be solved in a way that will encourage the player to explore these islands. Further information can be found at next section. Here's the pseudocode of branching from the root,

Initialize the queue with branch roots.
While the queue is not empty
    Get the first island from the queue,
    Mark the island as pointless.
    Merge the neighbors of the island with their neighbors with a chance of side path merge.
    Add every unique merged island into the queue
End while

Note that the islands on the graph have multiple neighbors(See Figure 2.1) the islands may be checked for merging multiple times, chance of merging parameters should be adjusted accordingly as 50% for the merge chances almost enough to include every single possible island on the rectangle.(Figure2.1) Although not every island will have connection between them so there will still be branches and pointless areas.

3.2. SOLVING THE POINTLESS AREAS

The pointless areas solved in a way that it encourages players to explore the pointless areas. The generator places rewards and events to those pointless areas so that the player will have a reason to explore those areas.

Players should always make the decision of moving on or exploring other areas. They should be aware of their characters strength and their ability and make the decision of taking the risk of exploring other areas or moving on.

While generating the layout of the map, it marks the islands as pointless and while populating, the generator calculates the distance from every pointless area to their closest non-pointless area using breadth first search. The reason it uses BFS rather than DFS is the length calculated here means the nodes in the pointless area as DFS would give the number of nodes between a pointless node and non-pointless node, which would leave other pointless nodes in the area. It is crucial to not place too many rewards to a single pointless area as that would take the decision away from player as if the player gets too strong, they will have no reason to stick around to explore the whole level.

The quality of the reward must be directly proportional to the difficulty of the map. Placing a simple reward after a very long and tough detour will make the player feel cheated. There must be consistency between the difficulty of the branch and the reward that the player gain so that the player can have a consistent idea of risk and reward of their detours.

Following is the pseudocode of solving the pointless areas. As discussed earlier calculating the difficulty of the map is an important part of the solvation of pointless areas. The things that directly affect the difficulty of the pointless area are length of the area and the sizes of the islands on the area (this affects the number of monsters in the island which defines the difficulty of the island).

Equations for the calculating the difficulty and quality of reward can be found on Chapter 7.

a. Selecting the pointless island with the greatest distance from any main path node

b. Performing one of the following actions

i) Depending on the difficulty of the of the pointless area, drop a small, medium or large reward in this pointless island.

ii) Pick an island on the path that and place an event.

iii) If the size of the pointless area has less than 2 nodes, populate it with just monsters and a small reward with a slight chance.

c. Mark islands in the pointless area as solved and recompute pointless islands repeat until there's no pointless area left.

This algorithm may vary from game to game as this is a very basic version of solving pointless areas. It only adds 1 reward or event to a single node and marks the rest of the nodes as non-pointless. While adding more rewards may not always be a good solution, for example adding more monsters or traps to other rooms in that area or different game mechanics could also be easily implemented into to system.

4. TESTS AND RESULTS

As an example game design case to generate the maps, the monsters, events and rewards are,

Monsters:

Tier1: Ranged, Warrior,

Tier2: Assassin, Mage,

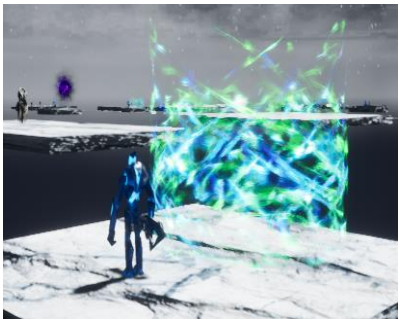
Tier3: Juggernaut, Necromancer.

Rewards:

Tier1: HP Boost, Stamina Boost,

Tier2: Weapon, Armor,

Tier3: New Ability1, New Ability2.



Events:

Clear monsters,

Survive for 25 seconds,

Stay inside the pentagram for 10 seconds. The pentagram applies a random debuff.

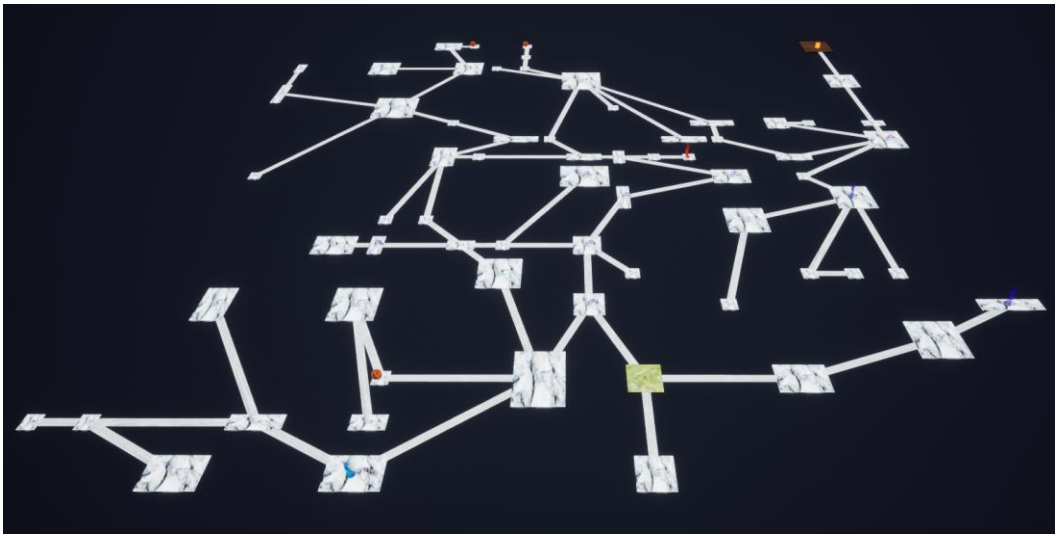


Figure 4.1 Generated from 20x20 in 38ms
Size of the generated area is 284.
0.71 of the whole rectangle.

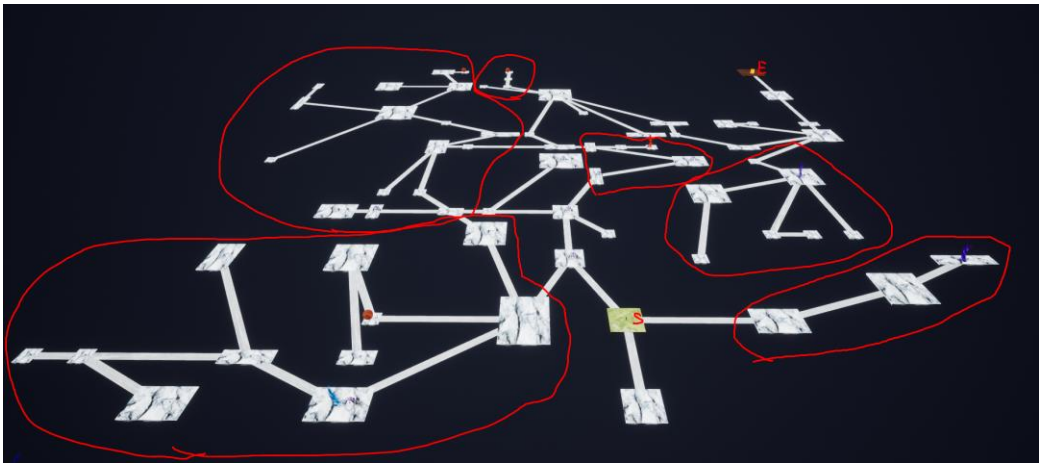


Figure 4.2 Pointless Areas marked in map on Figure 4.1

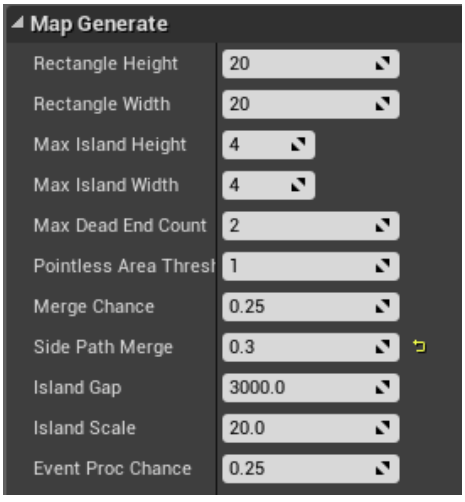


Figure 4.3 Parameters for the map on Figure 4

The glowy objects on the marked areas are rewards and events that are placed by the generator. Figure 4.4 is another run with same parameters except for merge chances.

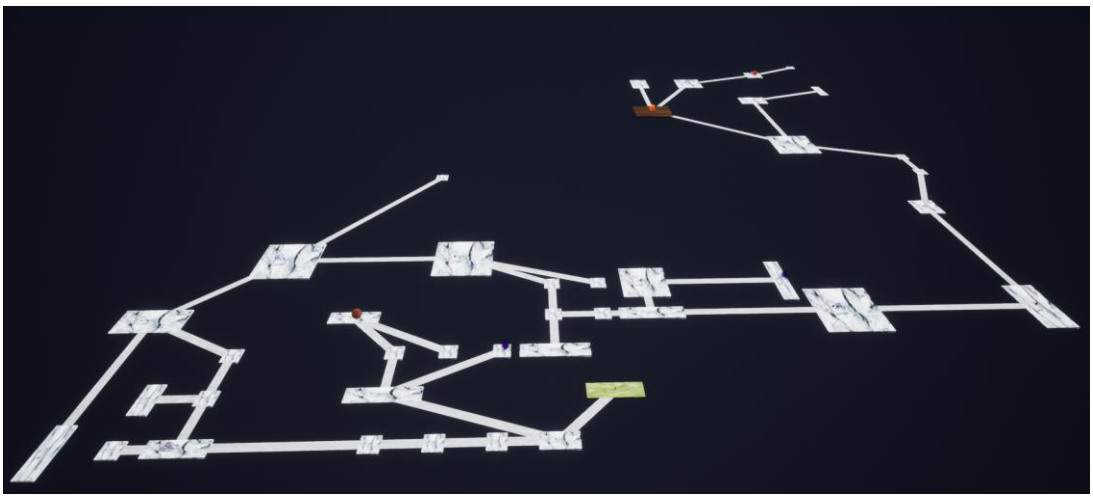


Figure 4.4 Generated from 20x20 in 29ms
Size of the generated area is 220.
0.55 of the whole rectangle.

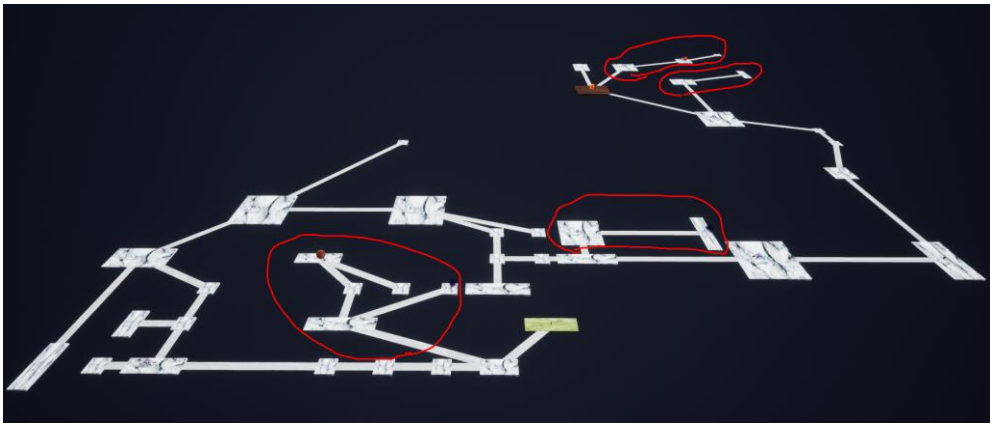


Figure 4.5 Pointless Areas marked in map on Figure 4.4

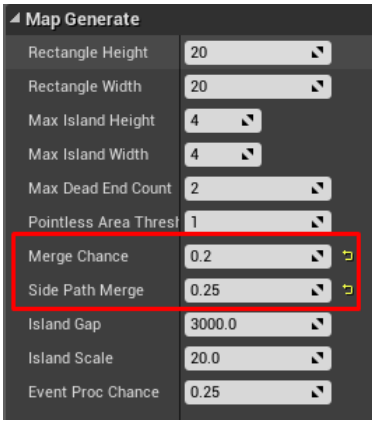


Figure 4.6 Parameters for the map on Figure 4.4

Another run with only the rectangle size parameters changed.

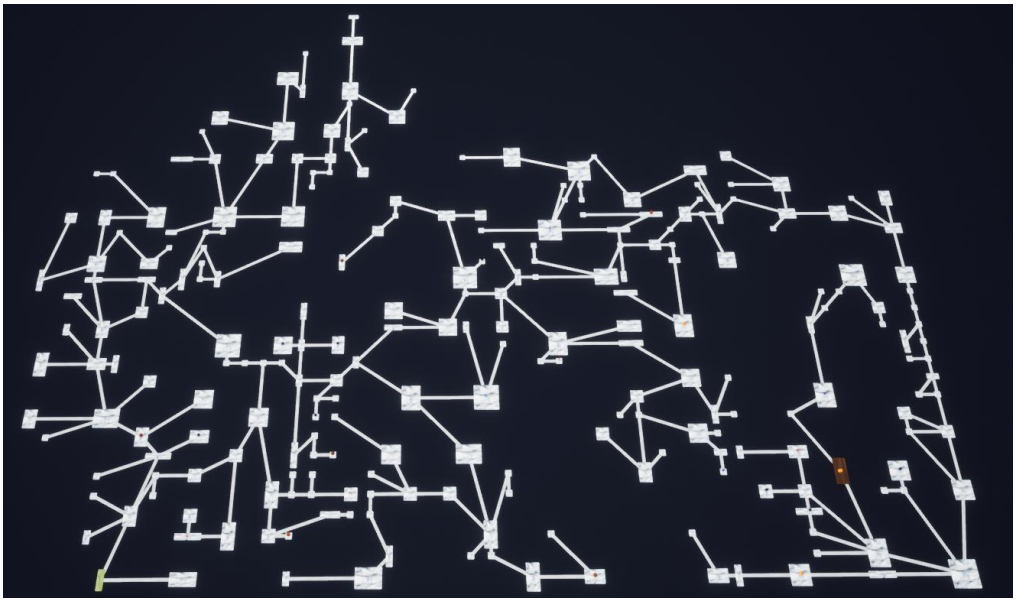


Figure 4.7 Generated from 50x50 in 461ms
Size of the generated area is 1554.
0.62 of the whole rectangle.

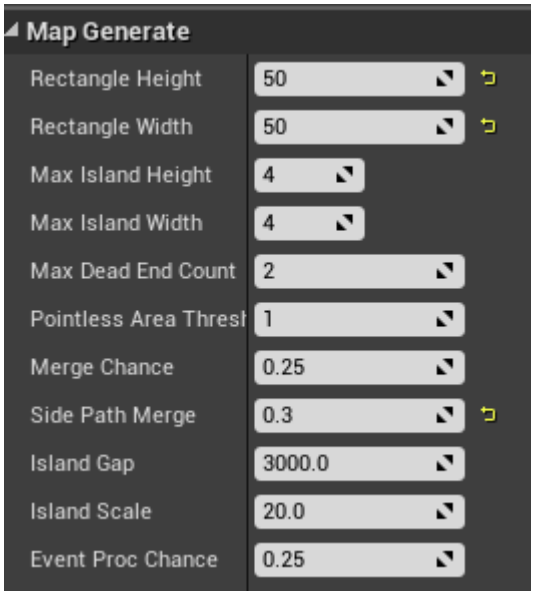


Figure 4.8 parameters for the map on Figure 4.7

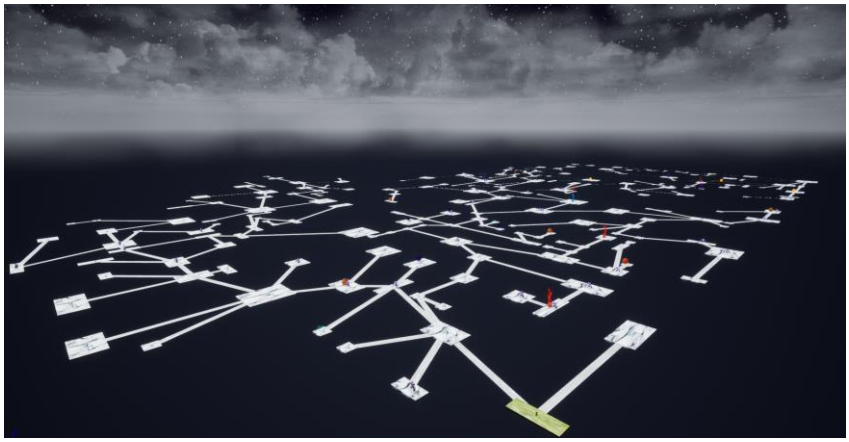


Figure 4.9 Another angle for the map on Figure 4.7

5. CONCLUSION

This project offers up a solution to one of the most common problems in game industry, replayability. It is a tool that developers can integrate into their games and modify it to their need. It can generate a layout from a 50x50 rectangle under 500ms.

The parameters allow changing the size and complexity of the generated levels which makes it easier for game developers to implement this system into their games to generate maps in runtime and save resources that goes into level design.

This tool can be further optimized and improved by giving an option to use pre-designed islands instead of using just a plane, ability to chose the material of corridors and islands, adding props to island, ability to group certain pre-designed islands with monsters, events rewards and dynamically change between these groups to create a sense of flow or a way to generate levels of certain themes.

Lastly, I want to thank Prof. Dr. F. Erdoğan Sevilgen again for accepting this project as my graduation project and helping me develop it throughout the whole semester.

6. SUCCESS CRITERIA

1- Generate a map from 50x50 rectangle under 10 seconds (Specs: GTX 960, Ryzen 7 2700 ~3.7GHz, 16GB Memory)
2- Generated map's size must be at least 1/2 size of the initial rectangle.
4- Generating a map without any pointless areas.

7. EQUATIONS

Measurment of a good map:

pointless island count / total island count.

Calculating the difficulty of a path:

Sum of areas of islands on the path / Maximum area size + (number of islands in the area * 0.1)

The reason adding 0.1 for every island in the area is to have the length of area impact the difficulty of the area. Without it the path with length of 100 with average area 2 would be easier than area with only 2 nodes with difficulty of 2 and 3.

Range between quality levels of rewards:

Maximum possible difficulty of a path / Number of quality levels for a reward

Quality level of a reward:

Figure 6.0 Example for quality level gaps for rewards.

8. REFERENCES

[1] Jonathan Rogers, Auckland Game Developers' Meetup, Path of Exile Random Level Generation Presentation, https://youtu.be/GcM9Ynfzll0 [September 2019].

[2] Adams David, Automatic Generation of Dungeons for Computer Games, http://xenophule.com/girscloset/Dungeon_Procedural_Generation.pdf [September 2019].

[3] Togelius, J., Preuss, M., & Yannakakis, G. N. (2010). . PCGames '10 Article No. 3 *Towards Multiobjective Procedural Map Generation*

[4] Roland van der Linden, Ricardo Lopes, Rafael Bidarra IEEE Transactions on Computational Intelligence and AI in Games (Volume: 6 , Issue: 1 , March 2014) *Procedural generation of dungeons.*

[5] Khantanapoka, K., & Chinnasarn, K. (2009). 2009 Eighth International Symposium on Natural Language Processing. *Pathfinding of 2D & 3D game real-time strategy with depth direction A* algorithm for multi-layer.*

[6] Hopson, J., (2001). Behavioural Game Design. Gamasutra, *https://www.gamasutra.com/view/feature/131494/behavioral_game_design.php [December 2019]*.