Procedural Content generation: Adaptive dungeon generator

Mikkel Stolborg, Mats Stenhaug
Games and Technology
IT-University of Copenhagen
Copenhagen, Denmark
Email: msto@itu.dk

***Abstract*—In this project we create a dungeon generator which uses player input as base for the generation. As the player moves throughout a dungeon, the actions taken determines how the rest of the dungeon is generated.**

1. Introduction

We wanted to make a board game dungeon map creator in this project. Having rooms, doors, items and interactables being procedurally generated on the fly, as we journeyed through the dungeon. Our idea was to create this in a way that utilized PCG in some sort of manner that would make the user feel like it was part of a ”story” in some sense, or that everything had some sort of connection.

This also meant that we wanted to give the user some sense of choice. Whether it was to possible directions to venture, or to have some sort of interactive decision-point where the user would have to set the scene for how the next parts of the dungeon would be generated. A simple example we came up with was to have the player choose whether he or she felt adventurous or not. Using this information in order to toggle if the dungeon should have a lot of bends and be made very spread, or if it should be more linear and maybe shorter.

Our problem statement was formed to be something like this:

- *Procedurally generate a dungeon and it’s contents in real-time, as the player explores what is already generated. Making points along the way where enemies, treasure or other points of interest can be found and interacted with.*

Initially we wanted to create something really big. Having the dungeon implement some sort of history. So say that if the player were to kill an enemy, the game would then know that that type of creature would act more hostile the next time the player would encounter them. And on the other side, if the player were to help that ”enemy”, maybe it would become a friend instead?

If we wanted to implement this type of history and decision making into the generator, we would also have to find some sort of way to generate these options and decisions as the player progress through the dungeon.

In this document, we will elaborate and discuss on the implementations we made, how we narrowed down our scope in accordance with the supervisor, and how we came to the conclusions that we did.

2. Background

Procedural content generation for maps and dungeon creation has been used a multitude of time over the years. So what we wanted to do for our project, in order to hopefully make it stand out, was to try to implement the element of decisions and history. By having close to everything being generated as you go along, would make it possible to generate a huge amount of different scenarios, maps and game plays. Technically speaking, every play-through could be different. Our goals and wants for this project was initially set very high, as we wanted to have the opportunity to implement as much of our vision as possible. How much we can implement, and how the scope may be narrowed down, remains to be seen.

There are plenty of concepts, ideas and pre-made generators out there, and it can be quite informative to look into those, read about how other people have made their solutions, and get ideas to how we can improve and make our project as good as we are able.

A project with a similar idea, created dungeon layouts using cellular automata algorithms to create a cave like dungeon layout. The article, *Cellular automata for real-time generation of infinite cave levels* [1], explores creating infinite levels of interesting and organic dungeon caves. However does not take player input into account, which is our goal in this project.

In figure 1 you can see a map layout of a level of the digital version of the game Space Hulk [3]. This was used as initial reference for our idea, as it is a digital representation of a board game. It is a very simple layout, but it has the key elements that we wanted as well. Rooms, doors and hallways. The video game is directly based on the board game version with the same name [4], and as such is related to the idea we had about generating a board game map through a piece system in a digital tool and context.

Further games generates dungeon structures, an example being the game from Blizzard North, Diablo II [5]. It generates the levels the player traverses on game creation. Though

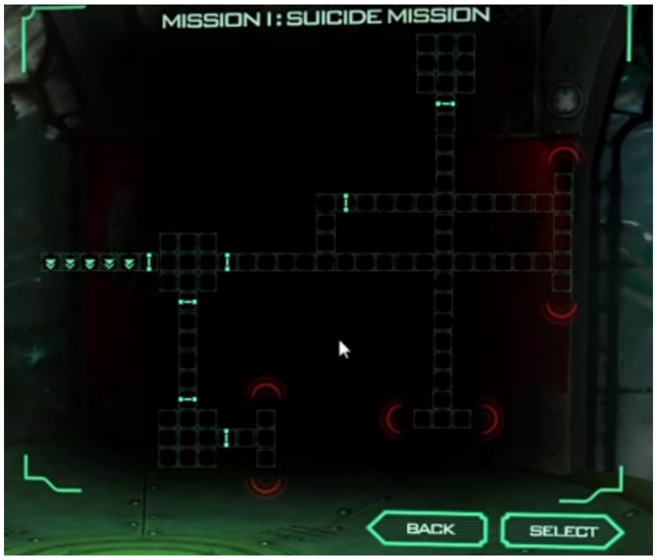

Figure 1. Layout of a mission map for the game Space Hulk.

the layout is not fuelled by player actions, its generation of levels provides a greater replayability.

When our generator is complete, we are hoping to use some smart algorithms, like A-Star, in order to validate if our dungeons are viable. We will have to use this in-game as we generate as well, if we hope to create our maps ”on the go” since we want the room placements to be dependent on how the player interacts with the world we create for them.

We believe that making this dungeon generator/exploration game will be an interesting project, as we will get to put our minds together to come up with what we hope will be great ideas as to how we can create fun and interesting dungeons.

3. Game Design

The core mechanics of the game are purely based on PCG. Without it, the game would simply not be possible. Of course, we could always create dungeons using manual design, and a paper-to-code approach. But since we want to make a game that is purely based on PCG, we need to create every algorithm from scratch, and we have to make calculated design approaches in order to make the algorithms as we want them.

Our first and foremost milestone, is to come up with a way of creating the dungeon in such a fashion that we can validate how good it is, and from there, make it so that we can add more features into it, like creating objects, enemies and other interactables.

Our approach to generating the dungeon, rooms and content in real-time, will be to first create a dungeon generator that will instantly create the entire dungeon in one go. That means all the rooms, doors and hallways. This means that we will have to come up with a clever way of making a complete layout of the generated content. Making sure that the sizes of the rooms do not exceed the total size of the map, and to enable the dungeons to be generated in different sizes and shapes. Just because we initially want to be able to create a complete dungeon, doesn’t mean we always want to create the same one, as that would not really constitute the sort of procedural content generation that we want. From there our task will be to break it down into pieces. Have only a single room be created from the start, and then expand on that room. Having one or more doors or ways to exit the room in different directions, and then generate new rooms that will be visible to the player.

Once we are able to generate a dungeon be creating new rooms and connecting them to the existing map, as the game is being played, the next step will be to have some form of content in the rooms. The start point will, for the sake of simplicity and control, always be in the center of the first room that is created. But as we expand on our dungeon, we will need to have some sort of end condition. So what we will do is to implement an exit area. At first we thought about having the exit point be set from the start, and then create the rooms in the map so that they would eventually reach that point. But as we thought about it more and more, we realized that this was not how we wanted it.

We wanted the players to explore as much as possible, without having to think about where the exact point of the exit were. So what we came up with, was to have the ”adventure” scale. Basically what that means, is that the player gets the choice to select how ”adventurous” he or she is feeling, and have that decision factor into the generation of the game space. If the player is feeling adventurous, the dungeons will be generated with rooms that spread more out, and that has more options of directions that are accessible. And going back to the end point; we believe the game would be more fun if the exit is generated suddenly, when you enter a new room that was previously unexplored. And when the exit would appear, would in some way be connected to how adventurous the player was.

With the ending condition now figured out, the thoughts on how we would generate other items in-game, like treasure, enemies, etc. arose. After brainstorming for quite some time, we came to the conclusion that these items and interactables should be dependent on the adventure-meter, but also be dependent on an element of luck and chance. Creating rooms that were dead ends, could significantly increase the chance for an encounter of some sort to happen.

4. Methods

Our algorithm starts off by creating a board element, then having a room spawn in the center of said board. The center room, see figure 2, will contain the starting point for the player, and will have 4 doors leading out of it in all possible movement directions, North, East, South and West. After the board and starting room are initiated, the player object is generated and placed on the starting point. From here, all the elements that are now contained in the room will be drawn up on the screen and the player can now explore.

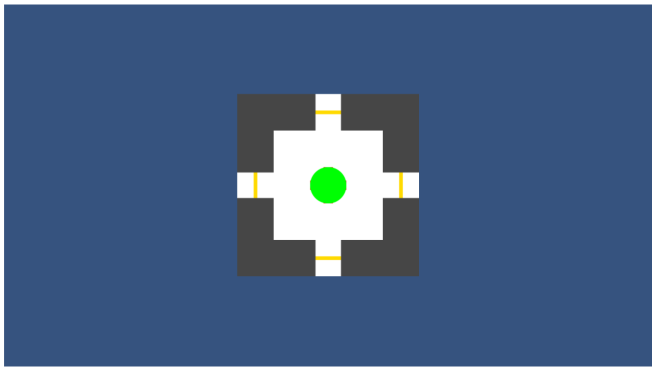

Figure 2. Initial dungeon layout. A center room with the player token and 4 doors leading out from it.

When the player walks through the first door (any of the initial 4 doors), the algorithm creates a new room in that direction. Since this is the first room that is generated, it will also contain an interactable spot where, if you walk on it, an option will appear on the screen that will allow the player to choose whether or not he or she wants the game to be adventurous or not. Interacting with the spot, known as the dialog interaction spot, allows for the game to get player feedback on their desire for how they want to play the game.

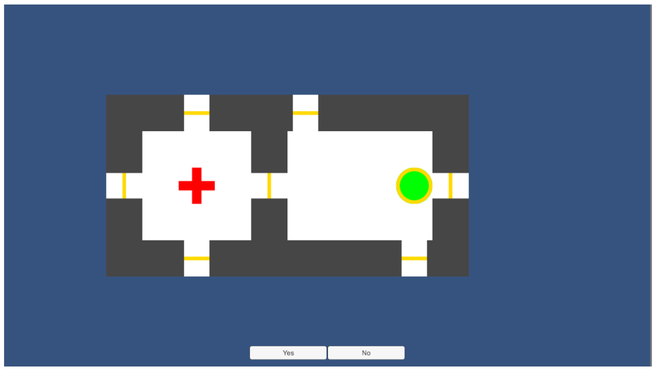

Figure 3. Dungeon with dialog interaction point placed. The Interaction point is represented by the yellow circle.

The rooms have a chance of making from 0 to 3 doors. Number of doors spawned depend on the adventureness of the players as described in section 4.1.

Once you have explored a lot of the dungeon, the exit point will have an increasing chance of spawning. Even if it spawns, you can always avoid going on it, and keep exploring, if that is what you want. But once you enter the exit spot, the dungeon will reset and you will start a new dungeon adventure. In figure 4 you can see an example of a partially explored dungeon with an exit spawned. As one can see there is still more dungeon to be explored, even though the exit has already been spawned.

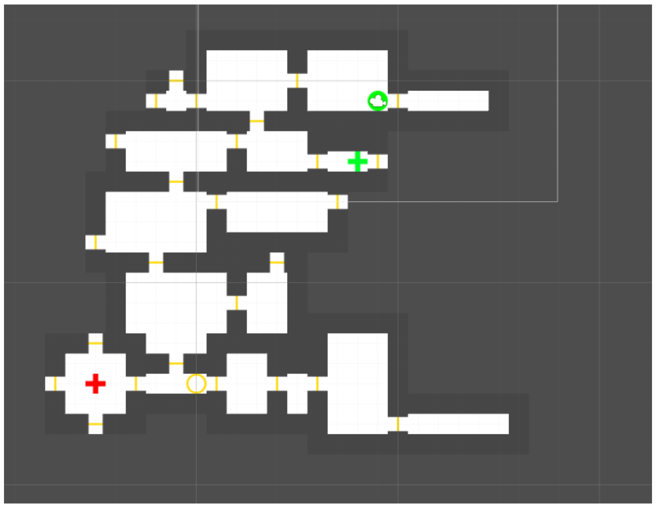

Figure 4. A partially explored dungeon with an exit spawned

4.1. Adventureness of the player

The adventureness of the player is set based on the interaction with the dialog interaction spot. It is more a stand-in for deeper gameplay interaction, and serves as a point where the algorithm can get a value which determines the players mindset when playing the game.

The adventureness scale is used in several of the random algorithms to push the outcome towards the players wishes. A more adventurous player will have a higher adventureness score, whilst a lower score is awarded to less adventurous players.

The two most important uses are for when to generate the exit and the number of doors in a newly generated room. The exit placement chance is based on the adventureness of the player and how long the player has been playing, measured in terms of moves made. The actual chance also uses a combined value of how much a player have moved, compared to the distance from the center of the map. This parameter, called advententurenessCalc is explained in more detail in section 4.1.1. The final calculation used can be seen in equation 1.

$$chanceOfExit = \frac{1}{1 + e^{-\frac{Tc}{AS*AC*t+1}}} \tag{1}$$

The terms used in equation 1 are as follows.

Tc is the turn count of the game. It is a measure of how long time the game has passed.

AS is the adventureness of the player and takes a value between 0 and 1.

AC is the advententurenessCalc, which is a calculated value explained in section 4.1.1.

t is a time scale factor.

The general gist of the function is that the longer the player spends in the game, the more of a chance there is to spawn the exit. The rest of the terms are there to prolong the game based on the players adventureness.

The number of doors created in a new room is a direct consequence of how adventurous the player is. There is always a chance for creating any number of doors, but the chance of more than one door is significantly increased based on the players adventureness.

4.1.1. AdventurenessCalc scale value. The adventureness-Calc value is an attempt to gather information about the players mindset real-time throughout the game. The value is calculated based on the how much the player moves around compared to how long the player has moved. The idea behind the value is that player who seek to explore more, will move around more than players we simply wishes to complete the level.

4.2. A-star distance measure algorithm

In addition to our own algorithms, we use the A-star [2] algorithm to determine the distance to the start of the dungeon.

The well known algorithm finds the path through a grid structure and calculates the fastest path to a given point from a current position. It uses a cost for each move it can make and tries to optimize its search towards a given target by minimizing the cost.

The value were used to determine the how the player moves around in the dungeon and how close to the center the player moves

5. Results

5.1. Did it work

We were not able to implement all the features that we wanted, which were described in the introduction and background. Our original idea was too out of scope, with much too many goals that would be unreachable given the time we had for our project. Even though not all we wanted to do, was completed, we were able to implement the most important features; having a dungeon being created in real-time, allowing the players to explore the dungeons as they see fit. Whether they are feeling adventurous, or just want to have a quick map and be done with it. By utilizing the A-Star algorithm, we were able to make the rooms get different ”adventure”-values based on how far away from the start they are, how much the players have explored, and how adventurous they have been. The chance for a room to spawn more doors, and in turn, more opportunities and choices for the player, are also based on the adventure-meter and the A-Star. If you for instance are far away from the start (have explored quite a bit), the chance to walk into a dead end are increased.

The win condition is also based on how adventurous the player is. If they are feeling like exploring a huge dungeon, then the end-point will not spawn until the map ”feels” like it has offered a lot of options and diversity.

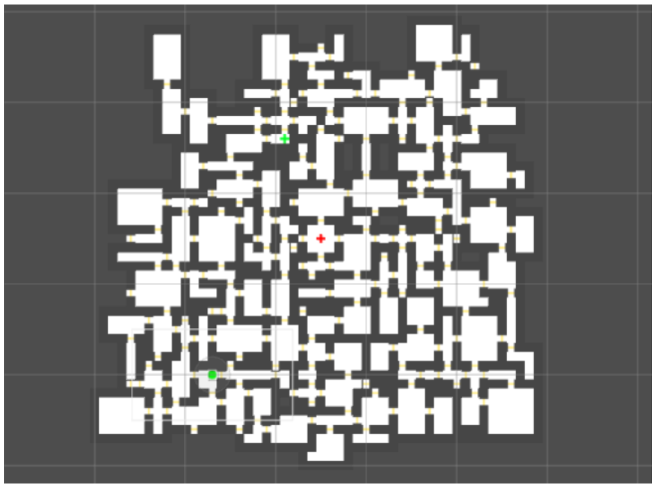
Figure 5. Picture of a massive dungeon created by adventurous player

5.2. How well

There were a lot of debugging required in order to create the dungeons as we wanted them to be created. Things took more time than we originally planned, which forced us to narrow down our scope, in accordance with our project supervisor. We still wanted to have multiple choices for the player, something that would give the feel of adventure and exploration. So by adding the option for the player to enable ”adventure-mode”, gave us the sensation of creating bigger and more diverse dungeons. Some of the issues that came up during development were that the generator itself wasn’t acting as we wanted it to. Rooms were created with doors that would lead nowhere or into a wall. So during development, we had to restructure and create new algorithms for the generator to work. This took a lot of time. Too much time in fact. Because of us having to remake the generators so much, we decided to make a completely new one from scratch, that took the best features we knew worked, and utilized them in a new way. This turned out to be a great choice, which allowed us to fix many of those bugs we had with the doors and what not.

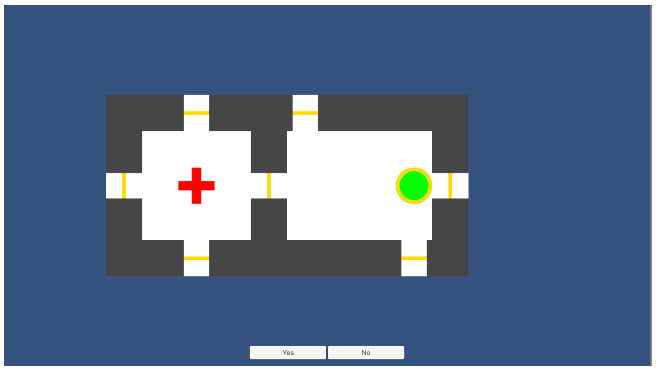

Figure 6. On this picture, you see the simple interface for selecting adventure mode or not.

Our finished product turned out to be very up to par with the feel we wanted to go for. We only use a simple interface, and graphics, although simple, are there only to let the player see which possibilities that are presented. A fully generated dungeon will be quite huge, and as you can see from the next figure, the player will have had to do a lot of choices in order to make it.

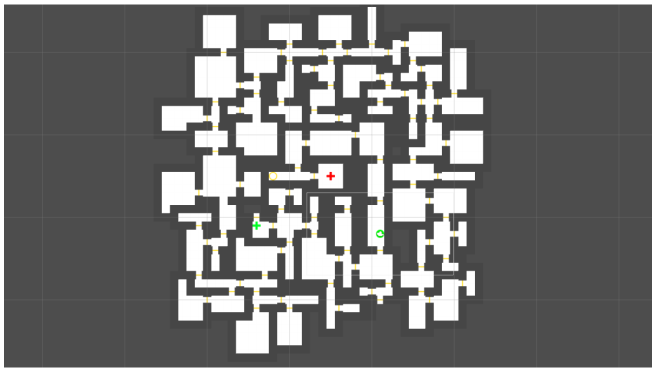

Figure 7. This is a full dungeon, all explored with no doors left to open.

As you can see from the last picture, all the walls are closed off, and the player has explored the entire dungeon. The bugs with doors overlapping were fixed so that one door might be removed if the player chooses to go a different direction. This also only amplifies the feeling of adventure and choice/change that we wanted our game to have.

6. Discussion

The method achieves a generation of a dungeon which is loosely based on the players choices. The dungeon has limitless possibilities and serves to create an interesting board. The dungeon creates a 2D layout as the player moves through it and generates the structure dynamically rather than just revealing new pieces at it goes along. Though simplistic it achieves a generation which creates something new and interesting in each iteration. Even though it generates a new dungeon each time, the content of the dungeon is simplistic in its nature. Using a more detailed and complex set of input data, one could generate a greater deal of content. Giving a player stats and goals, and using the progress to determine the next generation, could in term give a more interesting and compelling experience.

Another way to use the player input and content generation, would be to have a fixed map layout and focus only generating content based still on player input, and fix the style of the encounters to suit the players experience.

Further work on this project could be the improvements mentioned above. More content could be added to the game, such as, enemies, items and/or special rooms, all of which the player could interact with. The interaction could then be used to give a greater sense of influence in how the game develops. The game could have a greater story driven feel for the players as the story itself could be generated from their own actions. Further giving the player more the feeling that the game is being shaped due to their own actions.

What are the strengths and shortcomings of your method? Why did you choose method X instead of Y? How well would it generalize to other game genres? How would you develop it further, if you had time?

Further development: NPC spawning. NPC types, groups, interaction. Better connections between doors/rooms/walls Dungeon size reconfiguration

Acknowledgments

References

[1] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius, *Cellular automata for real-time generation of infinite cave levels*, 2010.

[2] Patrick Lester, *A* Pathfinding for Beginners*, http://homepages.abdn.ac.uk/f.guerin/pages/teaching/CS1013/practicals/aStarTutorial.ht Last visit: 13-12-2015

[3] Full Control and Games workshop, http://www.spacehulk-game.com/, Last visit: 13-12-2015

[4] Games workshop, http://www.games-workshop.com/en-AU/Space-Hulk, Last visit: 13-12-2015

[5] Blizzard Entertainment, http://eu.blizzard.com/en-gb/games/d2/, Last visit: 13-12-2015