Game Design-friendly Procedural Generation of Game Levels

# Abstract

Procedural Generation (PG) is often used in games for smaller aspects of the game’s scope and as such is scarcely used to produce entire levels for games, specifically in the Action genre, however, the increasing popularity of Roguelike games has led to further exploration with the concept of fully PG levels and experiences. A Roguelike is a game designed to have the player perform multiple runs of the same game with the overarching goal of reaching its end. While the player is completing multiple runs of the same game, each run differs in various magnitudes from enemy placement, types, strength and sometimes the room itself. This idea of generatively creating slightly different experiences of the same game each run makes a Roguelike game a suitable showcase for the usefulness of PG with the added challenge of making the resulting level designable while the designer is not handcrafting the levels themselves.

# Introduction

For Procedurally Generated content to be Game Design-friendly, the content must use elements of handcrafted and designed content in tandem with the content produced by the system when run. This means that for this project, a system will be built to showcase the power of Procedural Generation while still using the work and content created by a designer without the designer creating entire levels themselves. To that end, a Roguelike game will give the structure needed to create a game in which the player aims to clear a dungeon of enemies without knowing the layout of the level each time the player enters the game.

Roguelike games such as *Hades* (SuperGiant Games, 2020) feature a string of rooms filled with enemies, these rooms form the level or dungeon that the player must clear to either finish the game or progress the story. *Hades’* dungeon however does not have the rooms occupying the same space, instead, the rooms are treated as if in a vacuum. The proposed system for a procedurally generated dungeon is then to have these rooms explicitly connected through hallways and have them coexist with one another.

To achieve this and achieve a desirable design level, a similar workflow to Vazgriz’s *Procedurally Generated Dungeon* (Vazgriz, 2019) is adapted to use prebuilt rooms that contain at least one designed encounter inside. The proposed workflow has been chosen due to its flexibility and to lighten the work needed to create the hallways by hand.

The workflow includes three key concepts to succeed, Delaunay Triangulation (DT) (Rebay, 1993), Minimum Spanning Trees (MST) (D.Kalpanadevi, 2013), and the A\* (A Star) pathfinding algorithm (Xiao & Hao, 2011). DT allows the rooms to be connected to each of the closest immediate neighbours, MST allows for the path through the dungeon to be the shortest route through every room, and A\* will handle the positioning and groundwork for the generation of hallways connecting each room as dictated by the MST.

Once completed, the workflow should produce a well-connected dungeon that includes all the placed rooms with the shortest path. The workflow requires a low level of input from the developer at runtime and runs fully autonomously once started.

# Overall Algorithm

* Prebuilt rooms are placed within the bounds of a grid.
  + The rooms are chosen randomly from a list and then placed at random coordinates.
  + The grid is of a specified size that can be adjusted without causing faults.
    - The size of the grid can be a limiting factor to the number of rooms being placed.
* The rooms are then connected into a triangulated mesh through a Delaunay Triangulation.
  + This triangulation uses the Bowyer-Watson Algorithm (Rebay, 1993).
* The edges of the Delaunay Mesh are then used to create a Minimum Spanning Tree (MST)
  + This is obtained through Prim’s Algorithm (D.Kalpanadevi, 2013).
  + This gives the shortest path that visits all vertices on the Delaunay Mesh.
* Each of the edges left in the MST is then used to guide the Pathfinding Algorithm
  + The specific Algorithm is A\* (A Star) (Xiao & Hao, 2011)
  + It uses a cost function[[1]](#footnote-1) as well as the start and end points to find the fastest route from A to B.
* Given the data and changes to the grid, the hallways are then built procedurally.
  + A\* marked paths for the hallways to take.
  + Floor tiles are placed at the marked coordinates.
  + Floor tiles are then combined into one mesh.

# Grid Mapping

Grid mapping is used primarily for the designation of space within a given range. Grid positions can currently be referenced as empty, a room, a buffer, or a hallway. These designations help to keep rooms separated and to clearly demarcate where hallways can and cannot go.

Hallways can fill any cell on the grid that is not a room, but the type of room before being a hallway changes the value of that position when the Pathfinding Algorithm calculates the cost of the path, e.g., other hallways are made much cheaper to increase the likelihood that hallways will lead into each other.

The grid also helps to keep items equally spaced and easily accessible to the code and other scripts given its use of discreet[[2]](#footnote-2) values.

# Room Placement

Room prefabs are placed at random coordinates on the grid. These coordinates only include integer XY positions thus meaning that rooms need to occupy even amounts of space to keep the room allocation accurate across the entire dungeon.

Room allocation is the process of marking all positions within a room as a room to tell the Pathfinder to steer clear of those positions. This process is trivialized with the use of the Unity RectInt library as it can return a list of all the positions inside that space to be processed efficiently.

While a room is being placed a few things need to successfully occur; a random rotation must be performed on the room[[3]](#footnote-3), the entire room must fit inside the grid’s designated space, and it must not encroach on another room’s space. If any of these conditions fail the room is reset and the process restarts.

Once a room is placed a buffer is placed around it to give space for hallways and to present the other rooms with a reference to its presence in the general area, thus ensuring a gap between the rooms.

# Delaunay Triangulation

# Minimum Spanning Trees and Prim’s Algorithm

# A\* Pathfinding

# Hallway Generation and Building

# References

D.Kalpanadevi. (2013). Effective Searching Shortest Path in Graph Using Prim's Algorithm. *International Journal of Computer & Organization Trends*, 310-313.

Rebay, S. (1993). Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm. *Journal of Computational Physics*, 125-138.

Vazgriz. (2019, November 18). *Procedurally Generated Dungeons*. Retrieved from VAZGRIZ: https://vazgriz.com/119/procedurally-generated-dungeons/

Xiao, C., & Hao, S. (2011). A\*-based Pathfinding in Modern Computer Games. *International Journal of Computer Science and Network Security*, 125-130.

1. The cost function refers to the values A\* gives to each cell it evaluates as it chooses its path. [↑](#footnote-ref-1)
2. Discreet values refer to the use of integers over decimal value numbers. [↑](#footnote-ref-2)
3. The range of rotations is 0, 90, 180, and 270 degrees for simplicity. [↑](#footnote-ref-3)