Complex Game Systems – Procedural Dungeon Generation

Mara Dusevic

Purpose of the system

The system aims to allow users to generate two-dimensional dungeon crawler maps within Unity's editor mode that can be customised. Applying the library allows for easy implementation of a procedurally generated map and requires very little input from the user. It features custom inspectors and editors that can be used to edit aspects of the generation as well as scripts that contain all the necessary mathematical algorithms. The user will have the ability to apply custom asset packs to the system, create editable rooms within a tile map which can be navigable and edited by the generation. Additionally, they are given the ability for random item spawning for specific rooms or corridors as well as the choice to create the dungeon procedurally or create it physically on a tile map.

Third-party libraries required

The following are the libraries that are required to build the system:

- Basic Unity libraries included
- Unity Editor Library – used to create a custom editor within Unity to change variables
- Unity IMGUI Library – used to add UI elements to custom editor

Mathematical operations to be used

To randomise the dungeon simple random operations will be used to position the various sections. Additionally, operations to check for overlapping of those sections, basic vector math for calculating distances between objects and quaternions to deal with rotation.

Advanced algorithms to be implemented

To place the corridors connecting rooms, they are first generated through a depth-first search algorithm. It firstly checks one single piece on the grid where the surrounding pieces are checked to determine whether they are in use or not. If the piece has not been visited, this piece then travels across the grid in a random direction. This process is then repeated until there is a dead end. At a dead end the algorithm backtracks to the previous piece and chooses a new random direction. If not, the piece will continue to backtrack. When all pieces have been checked we return to the initial starting point so the algorithm will find a new start position to create new corridors.

After creating all the corridors, the system will then search for tiles to place ‘potential doors’ on where the two adjacent tiles are rooms or a room and corridor. This search is done first with horizontal tiles and then the vertical titles. These ‘potential doors’ are places into a list and then looped through to randomly select doors. After being looped through all ‘potential doors’ are removed from the list. All doors are then grouped together by the room in which they are connected to. These groupings are then checked to ensure they meet the quota of doors allowed to be connected to the room. This allows the user to limit the number of doors connecting to a room. When the doors have been located, the tile it will sit on is used to locate the room in which it is a part of.

Once found, the instance of the room prefab is edited by the system to remove the wall tile. This then edits the Composite Collider 2D applied to all room prefabs added to the pool of rooms. As each tile has a collider applied to it on start-up through my system, the removed tile will not affect the entire room’s collider. This piece is then replaced with the specified door object that can be rotated depending on its location in the room which will have no collider applied by the system unless the user adds one themselves.

How it will be made modular

The system will be made available to users via the Unity Asset Store as a package they can download and implement into their project. The package will contain all the scripts needed, a demo scene and a README to inform the user on how to use the system. In the generation system, the user will be able to customise rooms to various sizes on a tile map, add random item they have created spawning to those rooms, apply any custom asset package and create either procedurally generated or specific dungeon levels. Rooms created will be prefabs the user has made and applied all the necessary items or components to. The system will then access this prefab from a provided pool of objects to create the procedurally generated dungeon and edit the room’s walls to apply doors from connecting corridors.

Integrating the system with a new or existing application

It can be added into any new or old application via the package manager within Unity and the Unity Asset Store. When implemented the user will be told via a README file where to apply the scripts and how to customise the generation with their own assets or scripts. To add the main manager scripts, the user will need to apply them to empty game objects in scene to run during the game, the customisable tile map script is applied to the tile map the user wants to create dungeons on and any other script should be applied where necessary or left in the package folder.