Technical document on how to use the procedural generation in the project

To start with an empty object is created and then the script KdTree is added to it. The script contains several different designer parameters which can be modified to suit the needs of the procedural generation required. This document will be broken down into the key ideas and which parameters relate to them. For the purpose of this project the sample scene in the unity project has been set up with the KdTree along with some parameters filled in (player, square, stairs, items, traps and minimap). These prefabs can be changed to use prefabs created by the designer.

**Dungeon Axis** (DungeonXAxis, DungeonYAxis)

The first two parameters are the x and y axis of the total dungeon. Each can be set between 50-1000 depending on how big the designer wants them. They do not need to be the same although when setting the rest of the parameters in the spawner, the designer should make sure to refer to the smallest dungeon axis. For the generator while 1000 by 1000 is possible, it is not recommended due to the large framerate drop and should generally only have one parameter with 1000.

**Room size** (MinAxisSize, MaxAxisSize)

The next two parameters are the min and max axis size of the room spawning area. This is used to determine how far the total space of the dungeon needs to be divided, making sure that each area is between the min and max. Rooms themself are partly based on these as for example if the min is 5 and max 20 then rooms sizes will be between 3 and 18 allowing some space between areas for pathways.

**Items** (AmountOfItemsPerRoomMin, AmountOfItemsPerRoomMax, minSizeForItemsToSpawn)

The next three parameters focus on items.

The first is what is the minimum amount of items per a room and the other is the maximum. Using these two numbers the algorithm will select a random number between them and including themself. The next parameter is the min size for items to spawn which is used to limit spawning of items in certain size rooms. For example if it is set to 10 then any room 10 and under will not have items spawn in them. This can be useful for making small rooms empty if the designer does not want there to be items in them.

**Traps** (AmountOfTrapsPerRoomMin, AmountOfTrapsPerRoomMax, minSizeForTrapsToSpawn)

The next three parameters focus on traps.

The first is what is the minimum amount of traps per a room and the other is the maximum. Using these two numbers the algorithm will select a random number between them and including themself. The next parameter is the min size for traps to spawn which is used to limit spawning of traps in certain size rooms. For example if it is set to 10 then any room 10 and under will not have items spawn in them. This is useful as it can prevent small rooms being full of traps allowing only larger rooms to contain them.

**Room Division** (ChanceOfRoomDivideAgain)

The next parameter determines the likelihood of an area being divided again if it is between the minimum and maximum of the room axes. The higher this number is, the less likely an area will be divided again which can be used to make more large rooms common.

**Prefabs used by the generation** (Square, Player, Stairs)

The next three parameters are square which is the prefab for the ground, player which is the character the player will control and stairs which is the object the player needs to find to finish a floor. The square prefab gets resized to create rooms and is also used by the path between rooms.

**List of items** (ItemsToSpawn, OddsForItemsToSpawn)

The next two parameters are a list of item objects and their odds of spawning. This is used to add new items to the list and when a new item is added the designer should also add the odds of item spawn as well. The way this functions is the algorithm will take all the different values put in and add them all up. It will then make a random number between 0 and the total number and will then go through the int list until it has reached this number. It will then use the list point in the item list to get which item to spawn.

**List of traps** (TrapsToSpawn, OddsForTrapsToSpawn)

The next two parameters are a list of trap objects and their odds of spawning. This is used to add new traps to the list and when a new trap is added the designer should also add the odds of trap spawn as well. The way this functions is the algorithm will take all the different values put in and add them all up. It will then make a random number between 0 and the total number and will then go through the int list until it has reached this number. It will then use the list point in the item list to get which item to spawn.

**Minimap** (Minimap)

The last design parameter is the minimap camera. This is used by the algorithm to set the size of the camera to cover the whole area which allows everything to be displayed.

General explanation of generation

The way the generation works is it uses a kd tree type approach with the dungeon area divided at different points. Once the algorithm has decided that the area is divided enough it will then determine the size of the room in the area using the min axis size -2 to determine the smallest size it can be and -2 on the area’s x and y length for the largest it can be. Once a room has been generated the items and traps for a room will be generated. Once all rooms have been generated the paths between rooms and the player/stair for the floor will be generated. The speed of the generation will vary depending on how powerful the computer is with smaller dungeons taking less time to generate.

The limitation of the procedural generation

Currently each floor (scene) can only have one tile type which makes the rooms bland. This is due to using prefabs rather than a tile map which would increase the level of interesting floor design. Another limitation of the procedural generation is the rules required for each parameter which are dependent on other parameters. This is due to the limitation of the range function which requires the range not to rely on other parameters.