Complex Game System Brief

Purpose

For this assessment, the modular complex system I am creating will be a system that procedurally generates a dungeon in a 3D environment. The system will have an extensive amount of customization in the in the inspector that can tweak features like :

- Textures and materials
- Dungeon size
- Corridor sizes
- Ceiling height
- Starting and ending points,
- Procedurally generated furniture.

Other features that don't directly relate to the dungeon such as enemy and player attributes will be additionally added. Ultimately, this modular system is essentially a starter pack for people making 3D dungeon crawlers that allows multiple levels of customization with player and enemy integration. Essentially becoming like a dungeon crawler starter pack for developers.

Libraries

The system will use only anything that can be included in the scripts namespace. The main libraries being used include:

- System.Collections
- System.Linq
- UnityEngine

The unity collections library allows the system to utilize lists and functions that can add, remove items from a list. The collections library also lets the system enqueue and dequeue objects within queues.

The unity linq library allows the system to reorder and organise queues by specific orders and also allows the system to join multiple arrays. The system also has access to 'where' functions that work similar to 'if' statement except the 'where' function applies the condition before effecting the data while 'if' statement applies the condition after the data is affected.

The unity engine library gives the system access to more mathematics. The system will utilize some of these functions / classes such as all the vector variants (vector2, vector3, vector2int, etc), math functions, the random class, and essentially any advanced math functionality will come from this library.

Furthermore, additional libraries will need to be installed for the system to work as they are all integrated in the source code and can be accessed within the namespaces.

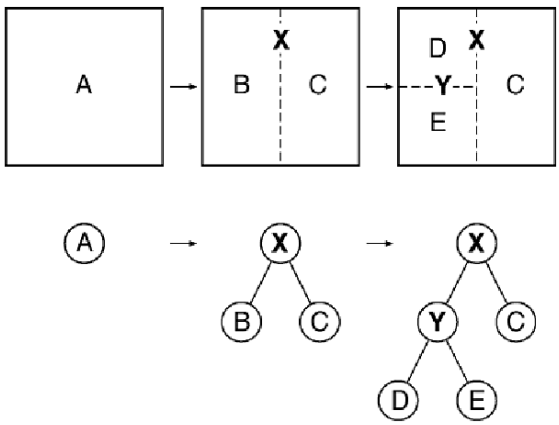
Mathematics

As a system that uses procedural generation, there are many mathematic equations used. Some of the core math equations that the system will utilise include absolute value functions as some aspects of the procedural generation cannot have negative values, such as the number of children an object has or the max number of rooms. Radians to degree functions alongside trigonometry functions will be utilized to calculate the orientation of the connecting corridors.

Some of the basic math equations that will be used includes finding middle points between objects that simply just used addition and division to add the two points and divide in half to return a middle point. Multiplication will be used when loading in mesh to create different sizes and adjust offsets. Subtraction will also be used to change the index in certain loops.

Algorithms

The basis of the modular system utilizes binary algorithm called binary space partitioning. In its simplest form, the algorithm in the form of a binary tree, uses a set space and divides it into smaller parts.



The split spaces would then be trimmed so all the split spaces aren't connecting. Each separated space would then choose a random neighbour to connect to with a corridor being place right in the gap which is calculated using a middle point formula.

The advantages of using this algorithm is that it adapts well to the parameters and environments as week as it being a hierarchy structure which allows for a modular design / reusability. Debugging is also made easier with this structure. The disadvantages of this algorithm is that it can be unbalanced and can the BSP tree can get very big. The algorithm also can't be compiled dynamically in runtime, but just once on start-up. Although there is a way around this which involves basically deleting every game object apart of the dungeon and running the script again using the inspector which isn't very optimal.

Modularity

This system aims for anyone using the system to have to use as little third-party scripts as possible to create a base for a dungeon crawler. Hence being a starter pack, if someone were to use this package, they would have a scene where a procedurally generated dungeon spawns with player and enemy prefabs. Elements such as dungeon materials and textures, player / enemy models and animations will have default ones, but user will be intended to swap them out for their own.

Integration

To integrate the system onto an application, all the user will have to:

- Drag the dungeon generator game object onto the scene from the prefabs folder which is just an empty game object with a dungeon generator script and default values for the variables.
- Assign the desired parameters for the dungeon generator script in the inspector to specific elements changed such as max rooms, dungeon size, room spacing, ceiling height, etc

The user would then be encouraged to swap out textures, models, and animations for elements such as dungeon floor and walls, player / enemy models and animations so that the user has used this system to create their own unique project with minimal third-party elements used.

References

Navi, V. (2020) *Binary Space Partitioning*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/binary-space-partitioning/

Studio, S.V. (2019) *Unity 3D Procedural Dungeon Generator*, *YouTube*. Available at: https://www.youtube.com/playlist?list=PLcRSafycjWFfEPbSSjGMNY-goOZTuBPMW

Uribe, G. (2019) *Dungeon generation using binary space trees*, *Medium*. Available at: https://medium.com/@guribemontero/dungeon-generation-using-binary-space-trees-47d4a668e2d0