Procedural World Generation Design Proposal
Aliesha Garrett, Charles Goddard, Tenzin Choetso, Morgan Zhu

Introduction

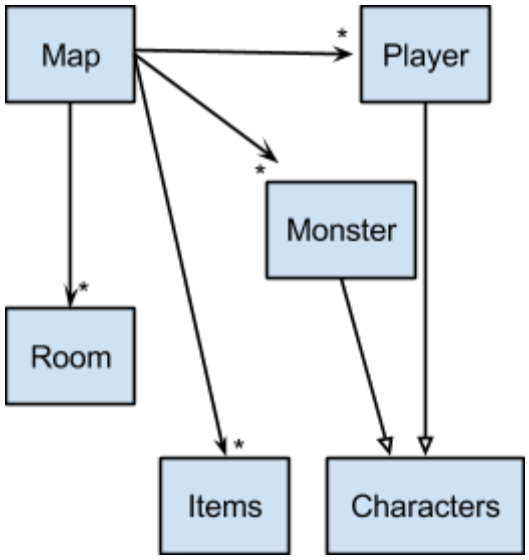
For our Software Design project, we are creating a top-down 2D dungeon crawler with procedurally generated maps. Players will randomly spawn within a room and be assigned a number of randomly generated tasks to complete. As the player proceeds through the dungeon, the difficulty of the game will adjust based on their performance (fighting monsters, picking up items, etc). Once all of the tasks have been completed, the player can advance to a new dungeon with new goals. We think that this is a well suited model for our learning goals at it should be fairly easily scalable and modular, lending itself to random level generation, there are clearly defined win/loss conditions, clear metrics for playability, and the potential for expansion of the complexity of both the world generation and difficulty scaling algorithms.

The minimum deliverable would be a game in which all the goals are exploration based (reaching a given room) on a randomly generated, coherent map (each dungeon room is connected to at least one other with non-trivial paths between them). Our maximum deliverable involves refined difficulty scaling, interactions with the environment, implementation of stats and monsters, and potential for multiple players to play within the same world.

Work is generically divided into two subgroups: map generation (in Python) and general game mechanics and difficulty scaling (using Unity). Currently map generation is being implemented with a text-based outputs and will be transferred into Unity using IronPython. Some graphical assets and preliminary behaviours have been created in Unity.

Model of the game

Two primary classes are being utilized within map generation: Rooms and Maps. Additional classes, such as Players, Monsters, and Items, exist within the game. The diagram below describes these class relationships.



Rooms contain their own width and height as integers. Rooms can also be assigned an (x,y) coordinate on a Map. Maps contain dictionaries, which map a given Room to any adjacent

neighbor. When initiating a map, the constructor is passed a list of rooms, which are randomly assigned coordinates detailing their locations. These Rooms are then connected by paths. The paths no longer have a designated class, but are rather denoted as empty 1x1 Rooms on the Map. This simplifies the general structure of the generation process, since the practical distinction between a room and a path is negligible when actually rendering the map. Items and characters are part of the map.

When a Player spawns, they will be assigned three goals. These goals are randomly chosen with distinct levels of difficulty, which is measured in how far a Player has to travel to complete them. If the Map is viewed as a tree structure where the room in which the Player spawns is the root, each level of the tree denotes a level of difficulty, since the Player has to explore (roughly) exponentially more of the map to uncover the correct location. Goals come in three general forms, find a Room, kill a Monster, and pick up an Item. In each case, the object will carry an attribute identifying it as a goal and will update the number of goals completed by the Player following the appropriate interaction. This scheme reduces the number of global variables that need to be tracked and allows the main game mechanics to be more generalized rather than having a map of functions to indicate when a goal is completed.

Analysis of the Outcome

We met our minimum deliverable, but not our maximum deliverable. We believe that we could have met our maximum deliverable with a little more time, but as this project stands it was pretty appropriately scaled. While some tasks were more difficult than expected (particularly getting our tunnels to generate the way we wanted), overall our expectations matched reality well. Additionally, our minimum and maximum deliverables had enough difference between them that a wide range of performances would have set us between the two goals.

Reflection on Design

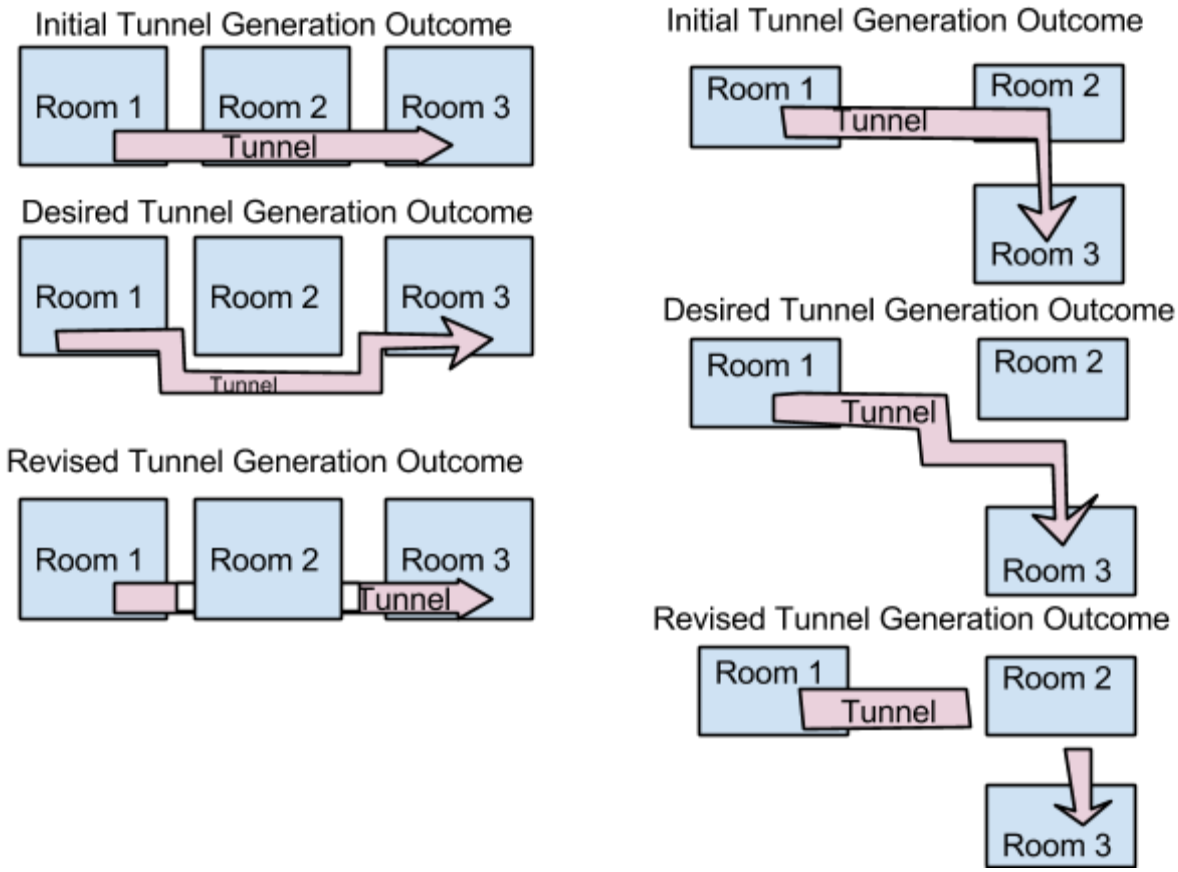
The game that we chose to work on was a dungeon game where rooms are randomly generated and connected in different ways by tunnels each time. This feature allows the map to be different each time to engage the player with a new setting. We are glad that we chose this game, since instead of having a defined set of rules and features, we can choose to make our own settings and add features as we progressed making the game.

Reflection on Division of Labor

The project was divided into two major portions- those done in Unity, and those done in Python. Further divisions were made to write specific classes and methods for those classes, allowing each person to work on a different function or set of functions. We used git throughout the project to allow each other to see the progress being made and allow us to integrate code into a single body so that integration could be done in a piecewise manner, rather than all happening at the end. We feel that this worked very well overall, allowing each team member to work individually on their own time, but also preventing us from having a mismatch in expectations between team members as we were designing our functions and algorithms.

Bug report

Tunnels did not generate correctly for certain room layouts. Our original tunnel algorithm involved taking a list of all rooms in the map and picking two rooms at random to connect until all rooms were connected in a single web. However, we realized that in order to accurately indicate which rooms were directly connected for quest assignment purposes, our tunnels would have to either check for intersections with rooms (or room perimeters) during tunneling, or we would have to keep the tunnels from intersecting with other rooms or tunnels. Unfortunately, we realized that while our tunnels were no longer intersecting unintended rooms and tunnels, they were also not necessarily connecting to the rooms they were supposed to. Below are some illustrations indicating some of the scenarios in which this happened.



The tunnel was, in terms of not intersecting rooms unintentionally, behaving as desired. However, it was not behaving appropriately in that the two rooms that were supposed to be connected at the end were not connected.

To debug, we inserted print statements into our code at regular intervals to try to see where the error was occurring. It became clear that the function was entering the if loop which declared that the next move was illegal, but never entered any of the conditional loops designating what the function should do next. Unfortunately, after about six hours of three different team members trying to figure out what was wrong with our conditionals by testing similar loops in Python, and trying different conditionals, we still couldn't figure out what was wrong with the function. We solved the bug by throwing out our old method and writing a new

one, because it seemed like less work than banging our heads against the strange bug longer. In the future we could possibly avoid bugs like this by writing our code in smaller steps. Because everything up to that point had been very easy to write and debug, we got cocky and started writing longer and longer chunks of code before testing. However, we were able to pinpoint the exact line where the problem was occurring rather quickly. We just couldn't figure out what was wrong with it.

Additional Resources

Link to the github repo which contains the Python portion of the project:
https://github.com/tchoetso/dungeon_generator (has been shared)
Link to the github repo which contains the Unity portion of the project:
https://github.com/cg123/SoftDesGame