Procedural World Generation

Aliesha Garrett, Charles Goddard, Tenzin Choetso, Morgan Zhu

Introduction

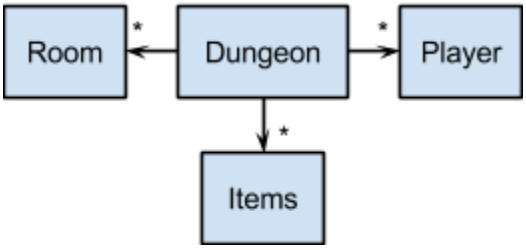
For our Software Design project, we are creating a top-down 2D dungeon crawler with procedurally generated maps. Players will randomly spawn within a room and be assigned a number of randomly generated tasks to complete. Once all of the tasks have been completed, the player can advance to a new dungeon 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 world generation.

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. Two additional classes, Players and Items, also 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 Dungeon. Dungeons extend 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 spaces on the Dungeon map, similar to Rooms. 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. Dungeons also random assign starting positions for Player characters and Items.

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

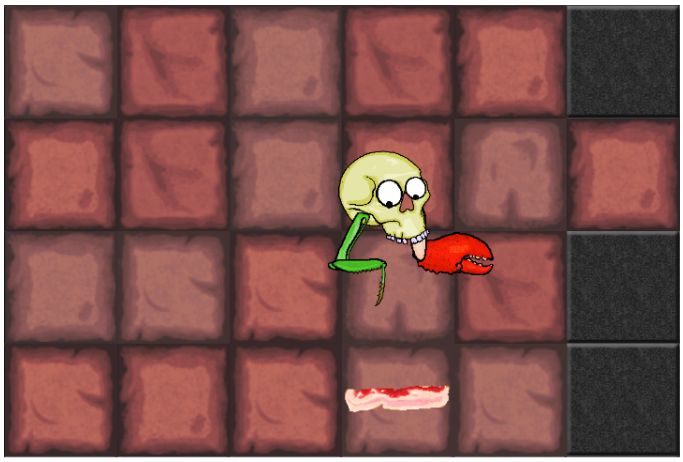
complete them. If the Map is viewed as a tree structure where the room in which the Player spawns is the root, each level of the tree denotes a level of difficulty, since the Player has to explore (roughly) exponentially more of the map to uncover the correct location. Goals require players to pick up items, which restore health. 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. Non-quest items are monsters, which will attack players. 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 and exceeded our minimum deliverable but did not reach the maximum deliverable. In general, difficulty scaling could be more refined; it is currently just a balance of maintaining enough health to complete all three quests. The generation of the dungeon map took longer than expected (see Bug report), which traded off with the number of game features that could be implemented. However, the combat element is reasonably challenging without excessive difficulty and met our expectations for the outcome.

Left: A screenshot from in-game during combat. The enemy (monk, or perhaps jesus- bottom right) is launching projectiles at the player (skull with crab and mantis appendages- top left). The player is upside down because the projectile force has made the player start spinning.

Right: The player approaching a quest item (bacon). The player must collect all quest items in order to win the game.



Left**:** The screen splash indicating that the player has won. Another splash is used to indicate the loss condition (player has 0 life remaining).

Reflection on design

The format of the game (top-down 2D dungeon crawler) was a good candidate for this project, as it allowed lots of flexibility with respect to the ruleset. Thus, we had some space to edit the mechanics we wanted to implement throughout the process while still producing a playable final project. While the procedural generation introduced difficulties during testing, it provides stronger engagements without the burden of producing a lot of additional content. Similarly, it is provides room for scalability by making more complex dungeon generation algorithms. There are a number of additional features that could improve the game, multiple levels, different forms of quests, etc., but they would not require any significant shifts in the design direction. One refinement would be to implement a separate Monster class, though the ruleset we developed did not have enough distinctions between Items and Monsters to justify it.

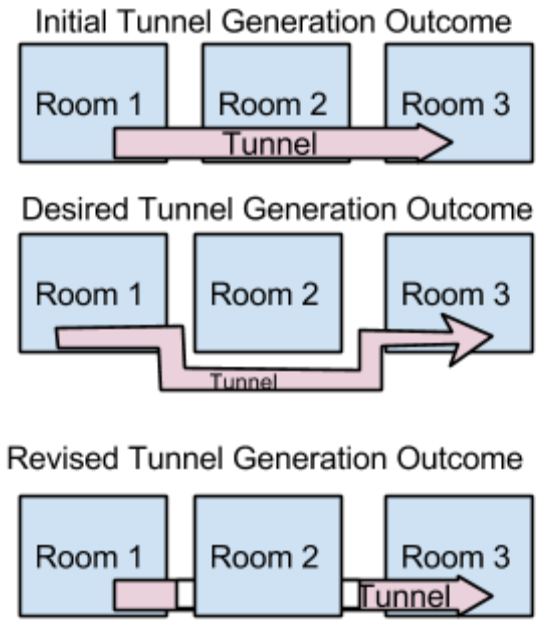
Reflection on division of labor

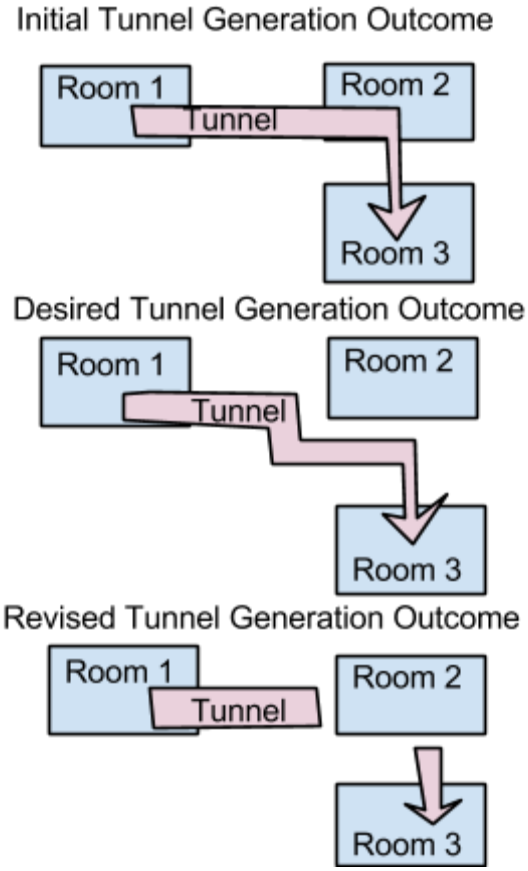
The project was divided into two major portions- game mechanics done in Unity, and dungeon generation done in Python using text outputs. Since the elements of both portions were fairly well-defined from the start, subdivision was relatively simple and outcomes could be assessed very clearly.Using Git as version controlled also allowed progress to be tracked easily and reduced the burden of integration by keeping the separate pieces of the two portions more visible. 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.

Significant effort was made to resolve this issue, which was based in the conditionals that dictated how to re-routed tunnels around rooms. We implemented a Tunnel class to help debug the issue, which was later removed because it was no longer useful. In the end, after several days of attempting to fix the bug, it was easier to simply choose a different algorithm, which placed the rooms and recursively tunneled through the dungeon until it connected all the rooms. This was a cleaner strategy, though less efficient. Additional work was necessary to map the connections as the tunnels were being drawn. The original algorithm was a victim of test-driven development, as additional requirements increased the necessary complexity. The best way to resolve a similar issue in the future would be to start with a more comprehensive test bench rather than taking a more piecewise approach.

Additional Resources

Link to the github repo which contains the Python portion of the project and the Final Paper: https://github.com/tchoetso/dungeon_generator (has been shared)

Link to the github repo which contains the Unity portion of the project and the final game as (Win32build.zip): https://github.com/cg123/SoftDesGame