Luke Sanderson

Tools & Middleware

Random Dungeon Generation

14/01/2017

Introduction

In this assignment, we were tasked in creating a simple dungeon. The dungeon is built of many smaller segments, which connect to each other in order to create a larger level.

The individual pieces were assembled in Blender, which was then converted to an fbx file which was read by our dungeon_generator.py code, creating a final fbx file that could potentially be used in-game.

The team decided to complete a separate part of the project's whole – Nikita and I were to ensure that corridor and room pieces slotted together, respectively, while Jack was in charge of ensuring the two separate types of pieces fit.

The Process

Firstly, we decided on what pieces to use for our dungeon. While an example file was provided, it was clear that some pieces were unnecessary or complete – for example, some stair pieces had no walls and connected to each other backwards. In the end, we decided to stick with large rooms that the player could eventually play in, connected by small corridors.

I started by assigning each of the pieces with a number of children that could be identified by name, for use by the dungeon_generator.py code later. The children acted as 'connectors' and were placed at all available exits of its room. *Fig 1* shows some examples of this – the top-right room can only be accessed from the top, middle and base.

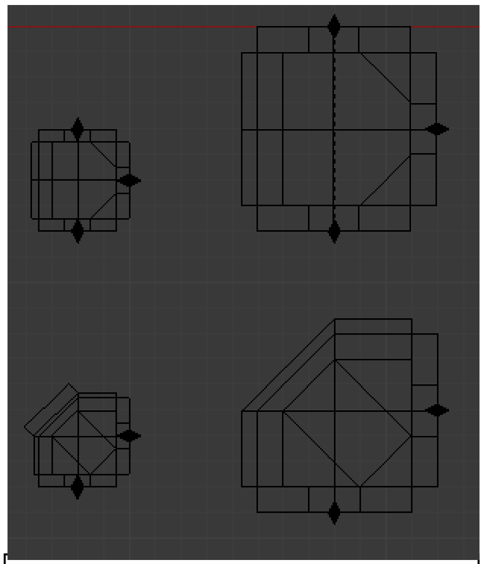


Fig 1

It was important that each room piece and child have the correct name, again for reference by the dungeon_generator script. An example of this is shown below in *Fig 2*, which is the hierarchy system for the top-right room discussed earlier. It must have the word 'room' in the title and its children must be (rather unhelpfully) called *bigflat*. The '<' and '>' pertains to the connector being an entry or exit point for another connector to join.

As well as creating and placing the various pins that connect the pieces together, I was tasked with creating additional rooms that serve a special purpose – for example, I created an ending room, a set of 9 rooms that interconnect with each other and walls for rooms with no exit, so the player can't leave the map.

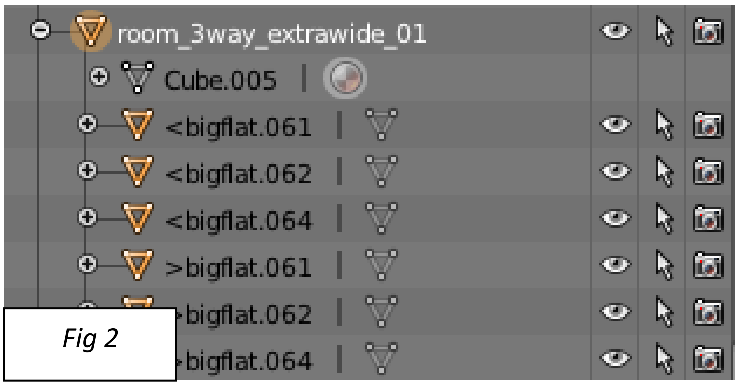
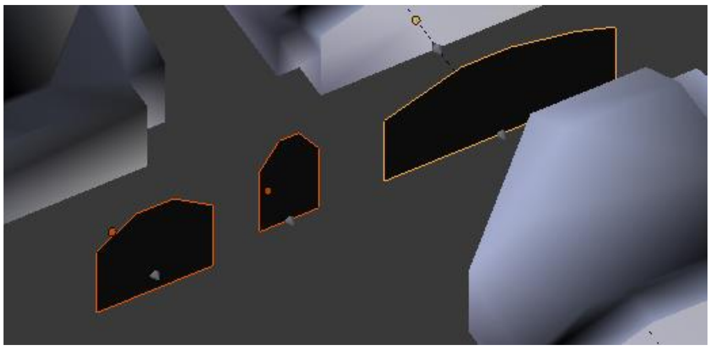
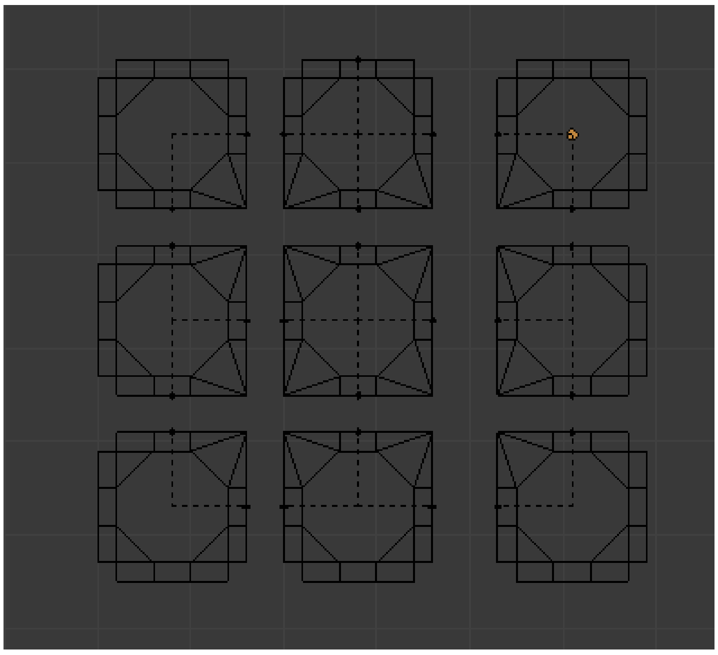

room_3way_extrawide_01
Cube.005
<bigflat.061
<bigflat.062
<bigflat.064
>bigflat.061
bigflat.062
bigflat.064

Fig 2



The Code

I focussed on the rooms_generator_module.py code for this assignment. My task was to create large rooms for the player to roam in. I achieved this by create a prototype set of 9 rooms that could connect to each other through their children's names, being *bigroomflat*. None of them can connect to the outside world aside from one piece found at the very top, which can connect to any objects with the children *todoor,* which connects to a door which theoretically leads to Nikita's wide dungeon pieces. Only one room was given an exit as to ensure that the rooms were given the chance to grow in size to give the impression of one large room.



The room shown on the bottom right is made up of 9 separate pieces shown on the right (the number of rooms created can be changed in the variable *no_of_rooms*). While it's clear from the outside that it's made up of separate rooms, once inside it is hard to distinguish where each piece begins and ends. The usefulness of this technique becomes apparent when you realise you can combine many corner, edge and open room pieces to create a potentially infinite number of rooms.