Steven Smith WGE Report

A number of scripts were used to implement the scene, the first script was the VoxelGenerator script. This script adds a mesh and other components to an object to create a number of voxel faces and, with use of a dictionary, applies a UV Texture to the generated voxel faces. The next script created was the VoxelChunk script, this script handles creating the main voxel chunk by looping through and creating many single voxels and positioning them correctly. It also handles the spawning and despawning of blocks and is connected to the canvas object. The AudioManager script handles all audio output in the scene. The PlayerScript script handles where the player is, what they are looking at and what action they perform. The XMLVoxelFileWriter script handles the encoding and decoding of XML files to be loaded into unity or saved into the project folder. The InventoryItem script is used to create a prefab of an inventory item and perform checks on an inventory item. The InventoryManager script handles displaying the inventory correctly and displaying the amounts of items correctly when an item is placed or picked up. It also handles the sort functions.

The communication between the scripts is used often and was kept to the use of events where possible, the main events are triggered when a block is placed or removed, two events were created in the PlayerScript, one to return what block type the player is looking at and one to handle the removing of the block, both these events triggered a delegate in the VoxelChunk script. When the delegate was triggered the information on the block would be sent to the AudioManager and the removal or placement of the block would be carried out. Events were also used for the VoxelChunk script to communicate with the AudioScript, the event, EventBlockSound would trigger in the SetBlock method and the AudioManager script would listen for the events being triggered. When the event triggered, it would play a sound depending on the information on the block that was passed to it earlier. The VoxelChunk script makes calls to the VoxelGenerator script to update the mesh when the chunk has been changed, the PlayerScript calls the InventoryManager script to update the item numbers when an item is hit by the player and removed.

A software design pattern is a reusable solution to a commonly occurring problem, in our case the software design patterns were used to do a task once or more. The InventoryItem script used this as it had a template for creating an inventory item and when used with the InventoryManager script it was called several times to create a full inventory. Another example is the VoxelGenerator script that was called in a loop by the VoxelChunk script to create the voxel chunk out of singular voxels.

The XML file “AssessmentChunk1” that was loaded into the game had the following structure: It began with the start-tag of VoxelChunk, this began the voxel chunk information. The first two elements after this were the start and end points for scene 2, after this the Voxel elements were placed, when loaded these told the VoxelChunk what Voxel type to load and where. After a 16x16x4 array of voxel elements there was the end-tag of VoxelChunk, signalling the end of the XML file and where the XML reader would stop reading.

Here is an excerpt of a part of the XML file:

<VoxelChunk>

<start x="1" y="4" z="4"/>

<end x="8" y="4" z="5"/>

<Voxel x="0" y="0" z="0">2</Voxel>

<Voxel x="0" y="0" z="1">2</Voxel>

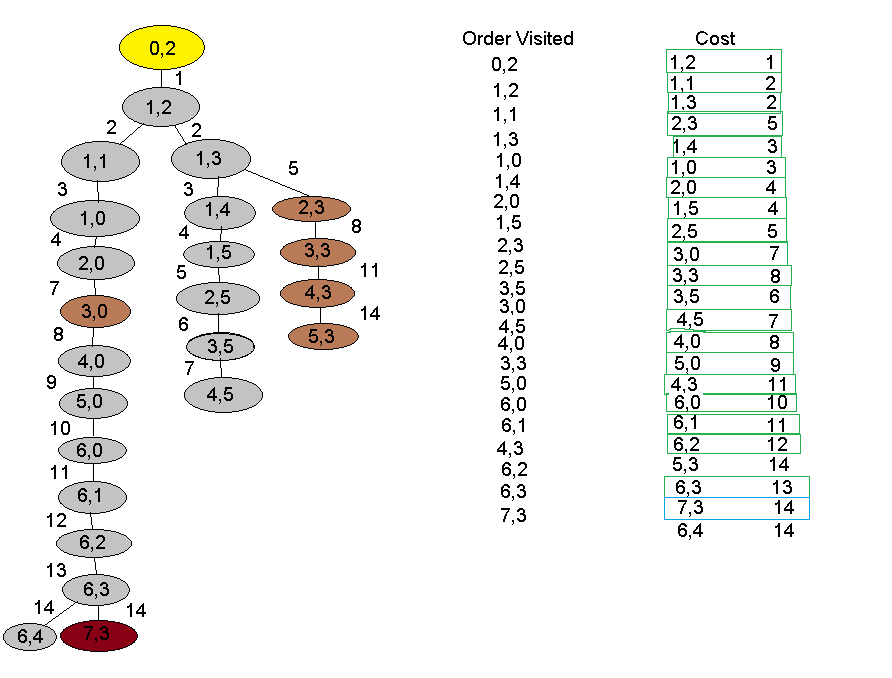
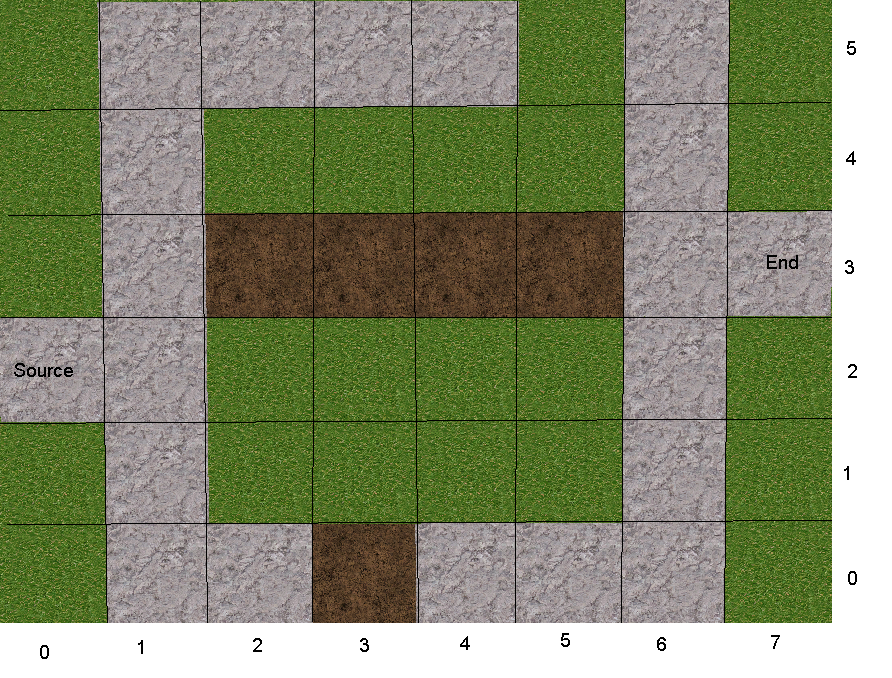
<Voxel x="0" y="0" z="2">2</Voxel>

This is a DOM of the XML File:

C:\Users\Steven\Desktop\XMLDOM.png

Dijkstra’s Algorithm is a path finding algorithm that works on weighted paths. Weighted paths mean the traveling from node to node is not the same; each node travel can carry a different value, meaning the shortest path by distance may not be the shortest by value. For Dijkstra’s Algorithm to work you need a list of unvisited nodes, the current best distance from the source point to each node and a list of parent nodes. First all node distances are set to infinite and all parents are set to null, all nodes will then be added to the unvisited list and the source distance is set to zero to set the start point. While the nodes remain unvisited, the current node with the shortest distance from the source is visited, the distance of all unvisited nodes adjacent to the visited node is then checked. The unvisited nodes are checked again to see which is closest to the source and the cycle loops until the end point is visited. This will mean the end point is always visited in the shortest path possible.

Dijkstra’s Algorithm Diagram:



A master server facilitates matchmaking between clients and hosts. The host registers their game sessions on the master server, allowing the creation of a game server which allows more clients to join the newly created game server and play with the client. The master server can have many game servers on it at once; the game servers can then have many clients on the server. The client must first connect to the master server before they can join one of the game servers on the master server.

In the coursework, the network views were used so game objects knew they were on a network. The objects that had a network view were the voxel chunk and the player cube. The voxel chunk belonged to the host and updated the client when the chunk was changed by the host or client. A player cube spawned when a client joined the server. Each cube belonged to the client that joined when it spawned.

The latency compensation technique used was dead reckoning, this was used as it predicts where the client is going to be ahead of the frames received, instead of other techniques that would merely move the object using frames received such as interpolation. Dead reckoning works by using the rigidbody attached to a game object to predict where the game object will be, ahead of time. It succeeds in doing this by using the line “syncEndPosition = syncPosition + syncVelocity \* syncDelay;”. The end position is updated once every 15 seconds as this is the default send rate across a network in unity. The game object is constantly being moved smoothly by the lerp in the update function which is called once per frame.