**Spike:** 5

**Title:** Game Graphs from Data

**Author:** Ryan Chessum, 102564760

**Goals / deliverables:**

* Code, see /11 – Spike - Game Graphs from Data/Task 11/
* Spike Report

**Technologies, Tools, and Resources used:**

List of information needed by someone trying to reproduce this work

* Visual Studio 2019
* C plus plus reference (<https://www.cplusplus.com/reference/>)
* Zorkish game specifications
* Class and header files from previous spike

**Tasks undertaken:**

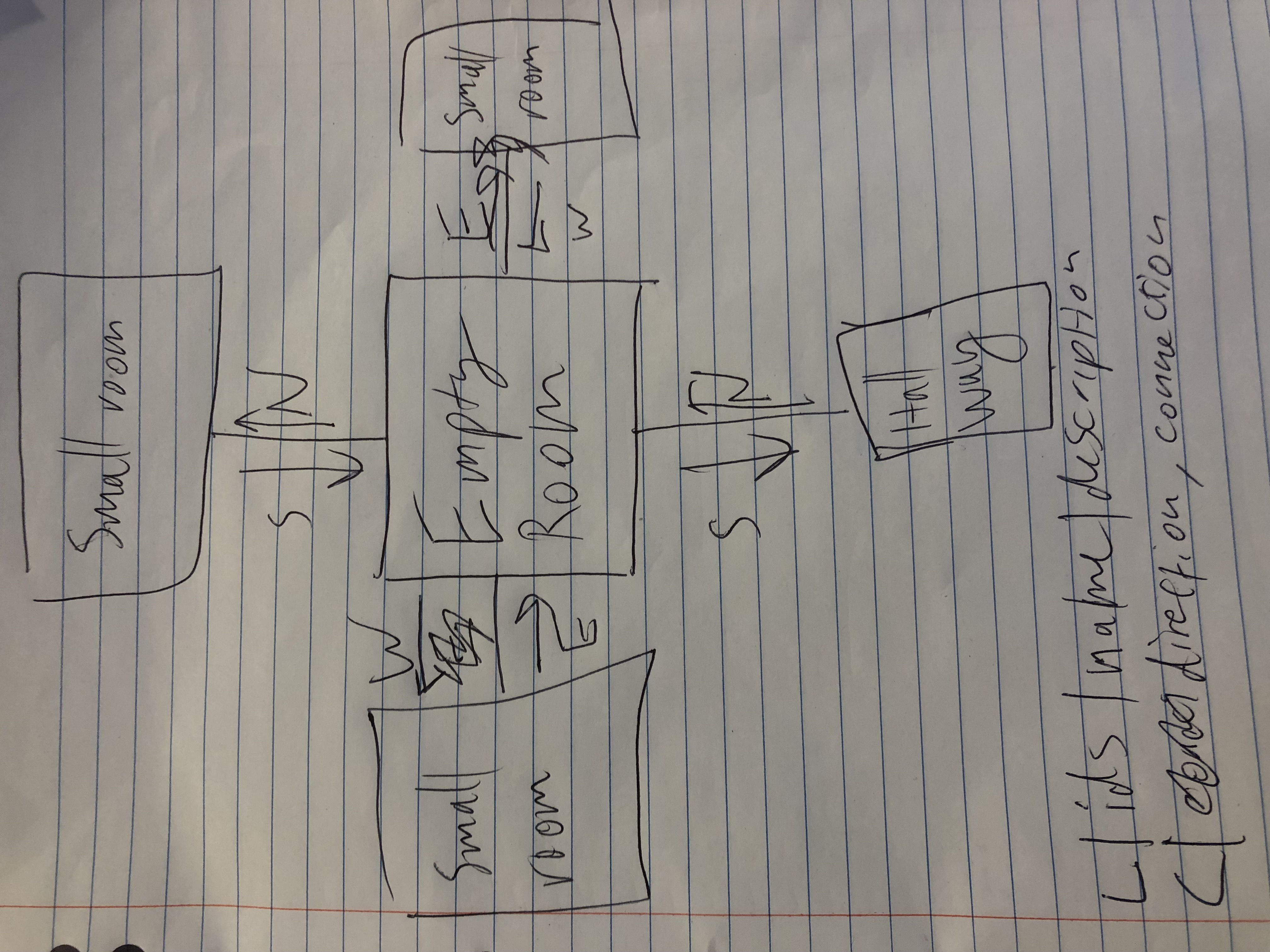
* Download and install Visual Studio
* Create a new C++ project
* Import Identifiable object, game object and player class code
* Create location class
* Create functions to read in world graph data from a file
* Create a test world file and add locations to it

**What we found out:**

A graph is a group of nodes or points connected by vertices. If we look at the Zorkish specifications, there can be any number of locations connected in any direction. At first this may seem difficult to implement but if we approach it as a graph it’s not too bad.

We can implement the locations by having each location in the graph be a node. Then each location node can contain a list of every connection which would be the vertex connecting each location in the graph together.

Here’s how I implemented the location class. Like the items from the previous spike, it inherits from the gameObject class. This lets us get name/description functions and ids.



To implement the location graph, each location is going to use a map to store its connections. This means we can pair each connected location with a direction. Up, down, left, right, north, south, east, west, north-east, a magical portal. Whatever we set the key string as will be the direction the player can move to travel to the paired location.

Text

Description automatically generated

Then there are functions to help us traverse and manage the graph.

GetConnection retrieves a location based on the string passed in by using the string parameter as a key.

Text

Description automatically generated

ConnectionList simply returns a string listing each connection that the location has along with its direction.

Text

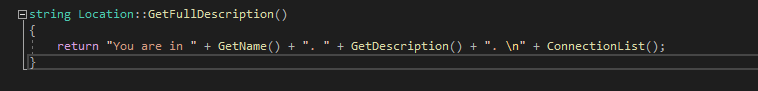
Description automatically generated

Finally AddConnection takes in a location pointer and a direction as parameters and adds them as a pair to the connection map.

Text

Description automatically generated

I also overrode the full description to have the list of connections with the location name and description.



Next, we need a way to traverse the graph. For this we are going to use the player. So, should each location keep track of the players position, or should the player keep track of which location it is currently in?

You could really do either if you tried, but having the player keep track of its location is the better option in this case. There is one player, many locations and the player can only be in 1 location at a time. We can simply add a variable that points to a location to the player and that can give us our current location. Then if we want to move the player, we just change the pointer to look at the next location. This is much easier to implement than having to test each location to make sure the player is in one place at a time, removing the reference to the player then adding it to another location,etc.

So, we can add a pointer variable to the player which points to the current location.

Text

Description automatically generated

The we also have a function which moves the player to the location at a pointer passed in.

Graphical user interface, application

Description automatically generated

Next, I worked on making a playable game world using these classes. To create the location graph, the program reads location and connection data from a graph and adds the locations to a vector. The vector isn’t used to access any locations after all the data has been loaded, it’s just to store the locations and is used in initialisation.

This is the format I used to load in the locations.

Graphical user interface, text, email

Description automatically generated

L signifies a location and C signifies the connections to other location. The file will be read twice. First it will go through and find all the locations. Then it will go through again and add all the connections to each location. The connections need to be underneath the location they are for. The reason for this is because all the locations need to be added first before we can add connections between them. So if the connections are underneath they should be read in at the same position in the vector. Every location Should have at least one connection, if not those locations can just go at the very end of the file.

Each line is also split up into each part that needs to be used by where there is a ‘|’. The locations have their id’s separated by a comma, then their name and lastly their description. The connections have each connection split up by ‘|’ and the direction location pair are split up by commas. Each room must have at least one unique id so that it can be used when specifying connections.

Text

Description automatically generated

When opening the file I am using argv[1]. This is because I am passing in the filename as a command line argument. To do this in visual studio go to properties -> Configuration properties -> Debugging and put any values you want as arguments. Argv[0] will be the filename of the program and the rest will be the values you set here.

Graphical user interface, text, application, email

Description automatically generated

Now we need to add commands to let the player go where they want to go in the graph.

Text

Description automatically generated

We can make a simple command processor in the update function. We can get the players input and split it into each word. If the first word is go then the program tries to get a connected location to the players current connection with the second word. If there is something there, then the players location is updated to the new one.

After implementing everything the program should run like this.

A computer screen capture

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated