**Spike:** Task 11.P

**Title:** Game Graphs from Data

**Author:** Mitchell Wright, 100595153

**Goals / deliverables:**

To demonstrate an understanding of collection types, their strengths and weaknesses, and apply that knowledge in creating a functional inventory system.

Items created during task:

* Code, see: \09 - Spike – Game Data Structures\inventory\
* Report, see: \09 - Spike – Game Data Structures

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

* Visual Studio 2022
* SourceTree
* GitHub
* Lecture 3.2 – Data Structures

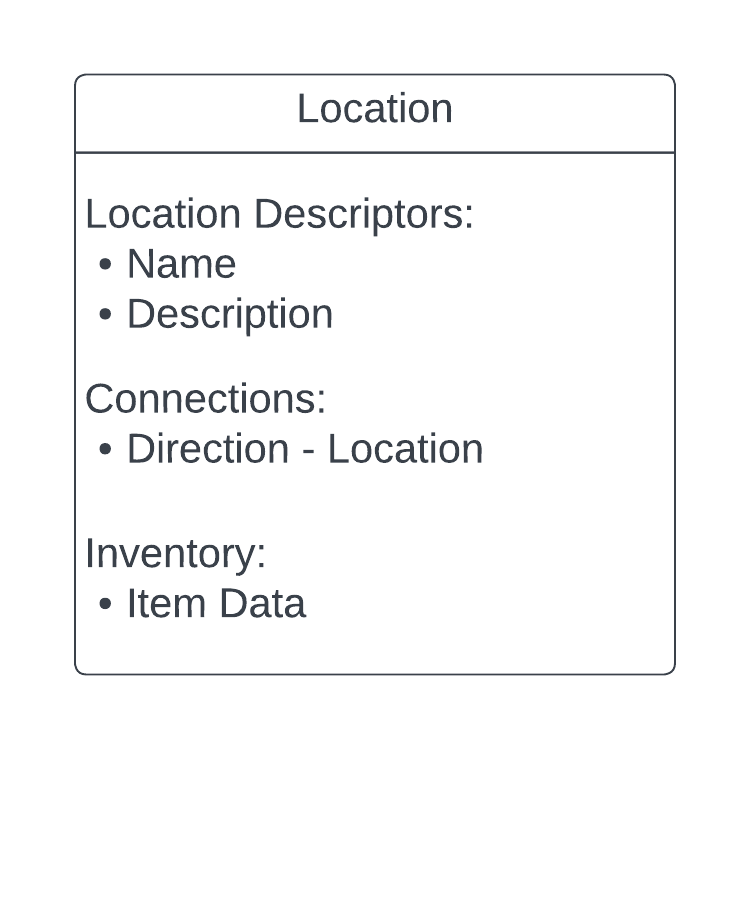
**Tasks undertaken:**

* Design the File Format
* Implementing the Location Class
* Implementing Interfaces
* Testing Locations
* Commit to Git

**What we found out:**

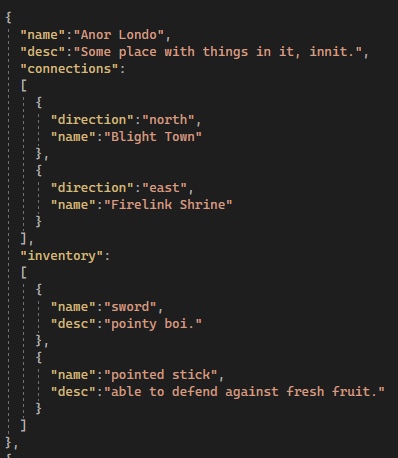
1. Designing the File Format:

I decided to go with JSON for the file format containing the location data. This is mainly due to how simple it was when used in the previous task. The files are broken up as below.



Please note that this is easily able to be added to, with the file currently only containing location data, and not yet containing any data about the specific adventure these locations are part of.

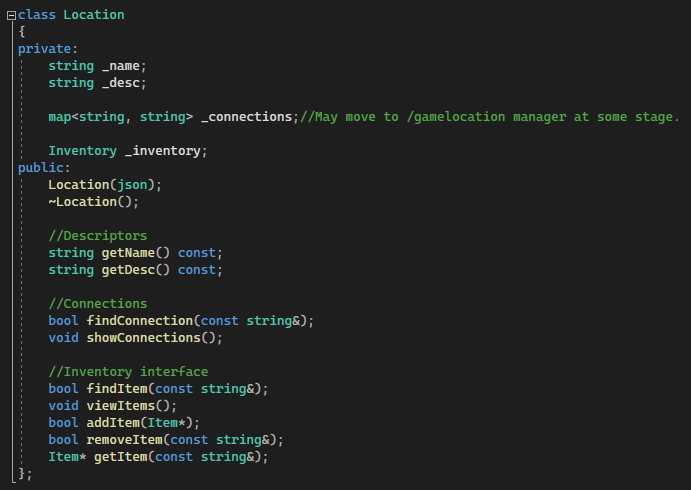
In this current iteration, each location contains a name and description. The connections are pairs of direction and location name. Finally, the inventory is made up of a list of items using name and description pairs (for now).



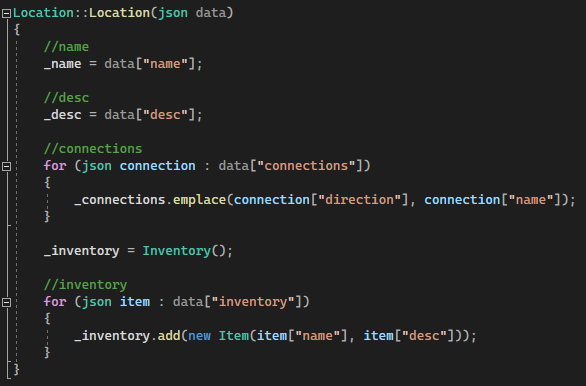
This is what the file ended up looking like.

1. Implementing the Location Class:

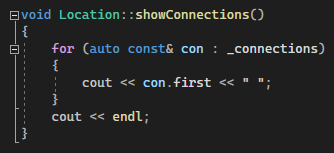
The Location class is primarily made up of a constructor and an interface with the inventory. I was briefly considering storing the connection data within a specific manager, though this idea was dropped as it seemed a bit out of scope for this specific task.



The constructor is the most interesting part of this class, as much of the rest of it are getters and setters, with some interfaces to the Inventory.



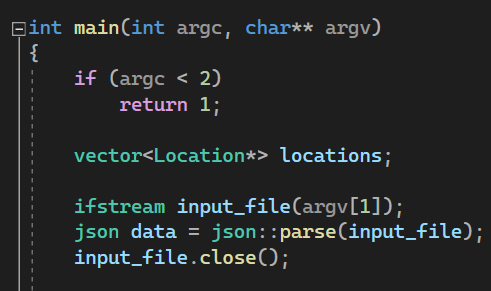
While the name and description assignment are pretty simple, the item and connection lists both involve assignment via a loop. These are done by emplacing the connection pair within a map, using the direction being the key. This however, will mean that “east” can only lead to a single place (Not important right now, but does remove the possibility of some fun stuff later on).



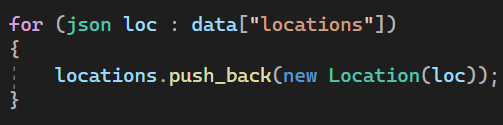
It should be noted that for now, the interfaces for connections and inventory, the functions directly output text to console, rather than an Ostream object to be output by a different part of the program. This is up for change depending on later tasks.

1. Testing Locations:

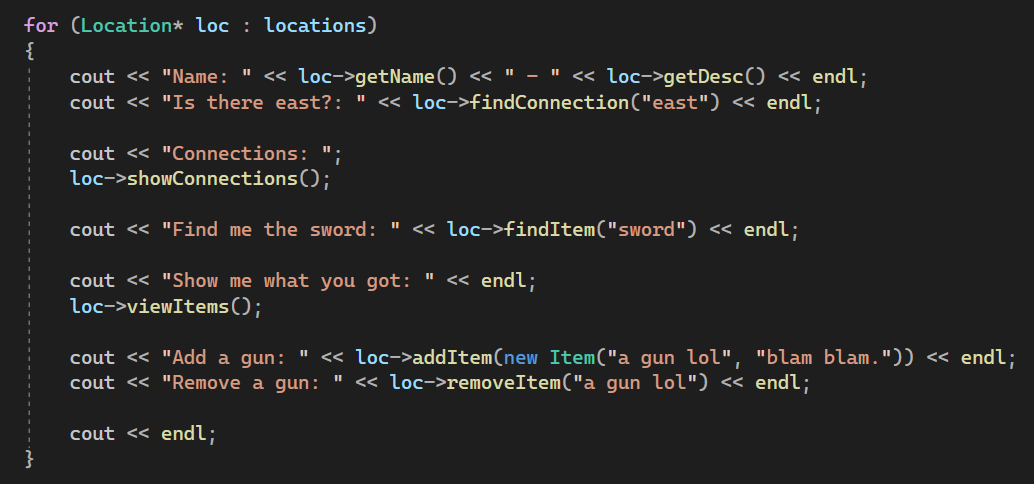
The location testing harness reused a lot of code from the Inventory task, with the file opening code almost entirely unchanged.



The big difference now though, is that the program checks for an argument which is then used to try and open a file. If no arguments are found, the program ends with an error code. For future implementations of this task, another if statement will be necessary, to make sure the file is good to go prior to attempting to parse the JSON data.



The data is then looped through, getting the pushing each location into a vector (temporary for now, will use a proper data structure later on).



Then, finally, each location is put through some checks to make sure that each of the elements are functioning as expected. The testing here is a bit light, as the inventory has already been proven to work previously.

1. Commit to Git: