**Spike:** Task 13.P

**Title:** Composite and Component Patterns

**Author:** Mitchell Wright, 100595153

**Goals / deliverables:**

To demonstrate an understanding of components, composition and interaction between different game objects.

Items created during task:

* Code, see: \13 - Spike – Composite and Component Patterns\zorkish

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

* Visual Studio 2022
* SourceTree
* GitHub
* Lecture 5.1 – Composite Patterns & Components

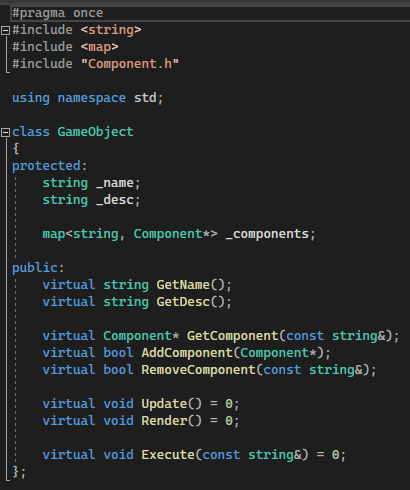
**Tasks undertaken:**

* Composition
* Component Pattern
* Demo
* Commit to Git

**What we found out:**

1. Composition:

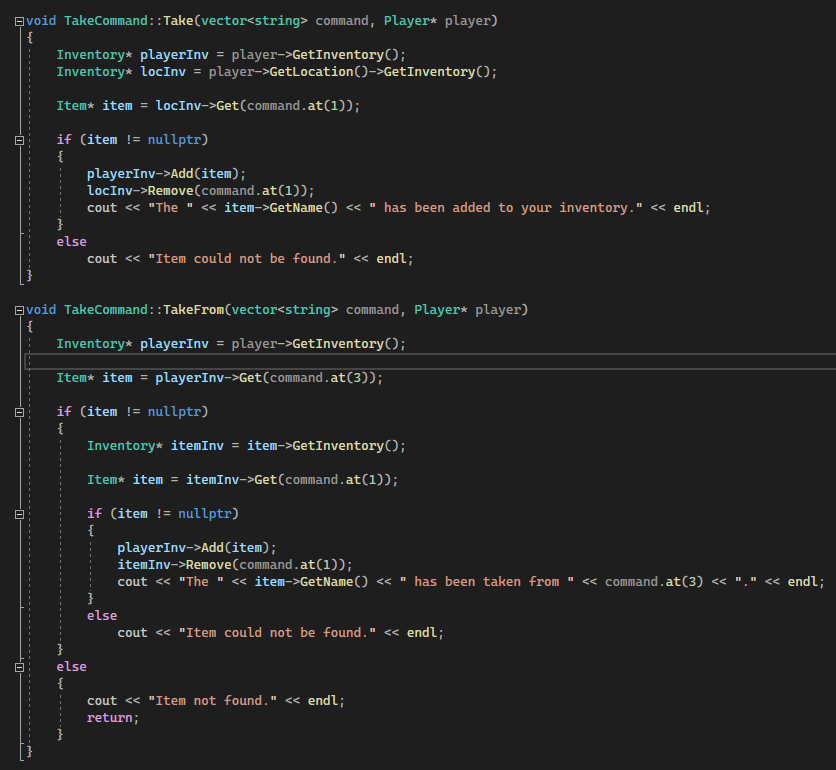
Implementing the composition pattern invovled refactoring a lot of the major classes from previous tasks and creating a super-class for game objects. This is shown below.



A lot of reused functionality has been shifted from Locations, the player, Inventories, etc, and dropped into this class, in order to allow for the implementation of a rough composite pattern. Please note the absense of inventories from this, as attempting to get those working was difficult due to circular dependencies with items, the inventory and game objects.

The implementation of these functions is much the same as in prior tasks, with most returning a bool as a safe guard.

The Take and Put commands were implemented to take advantage of this structure.

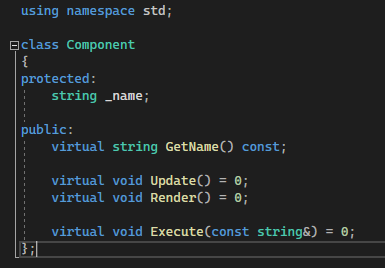


Being able to take an object, get it’s inventory and put an item in, meant that the composite pattern was somewhat working well. Again though, due to the lack of inventory implementation in the pattern, it wasn’t possible to just use game objects rather than specific classes.

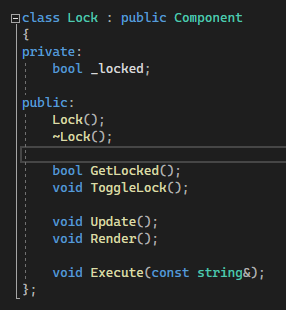
The Put command functions much the same, but in reverse.

1. Component Pattern:

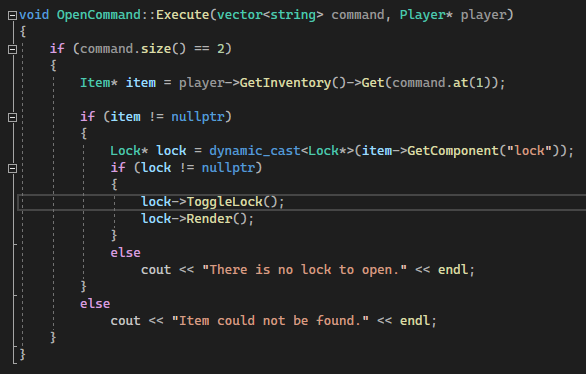
Components were simple to create, but time-consuming to retrofit into the code. I designed the open and lock commands in order to work with a lock component, so that the pattern could be demonstrated.



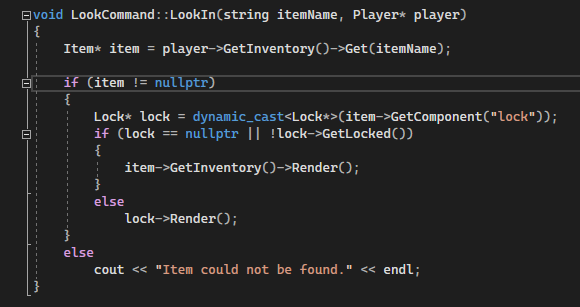
Each component has an Update, Render and Execute. Though, in this case, I’ve just opted to move the lock’s functionality into it’s own function.



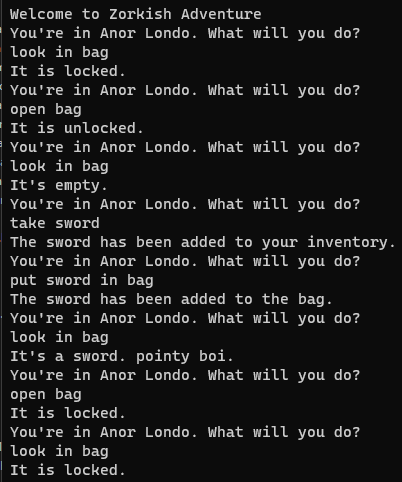
The open command, as shown below, checks the item for the component, then toggles the component’s lock function. A dynamic cast is used to allow the program to treat the generic component as a lock.



The look in command uses a lot of the same structure, though does combine that with the Inventory command.



1. Demo:



1. Commit to Git:

A blue background with white text

Description automatically generated