Spike: 9
Title: Composite Pattern

Author: Parth Madhani , 101901858

Goals / deliverables:

- Code
- To implement the Composite Design and Component Pattern
- To refactor old code and update it to use the new classes created by the implementation of this pattern.
- To expand the Command Processor to include more interactions with game world entities.

Technologies, Tools, and Resources used:

- Visual Studio IDE
- Assorted web sources.
  - YouTube
  - Tutorials

Tasks undertaken:

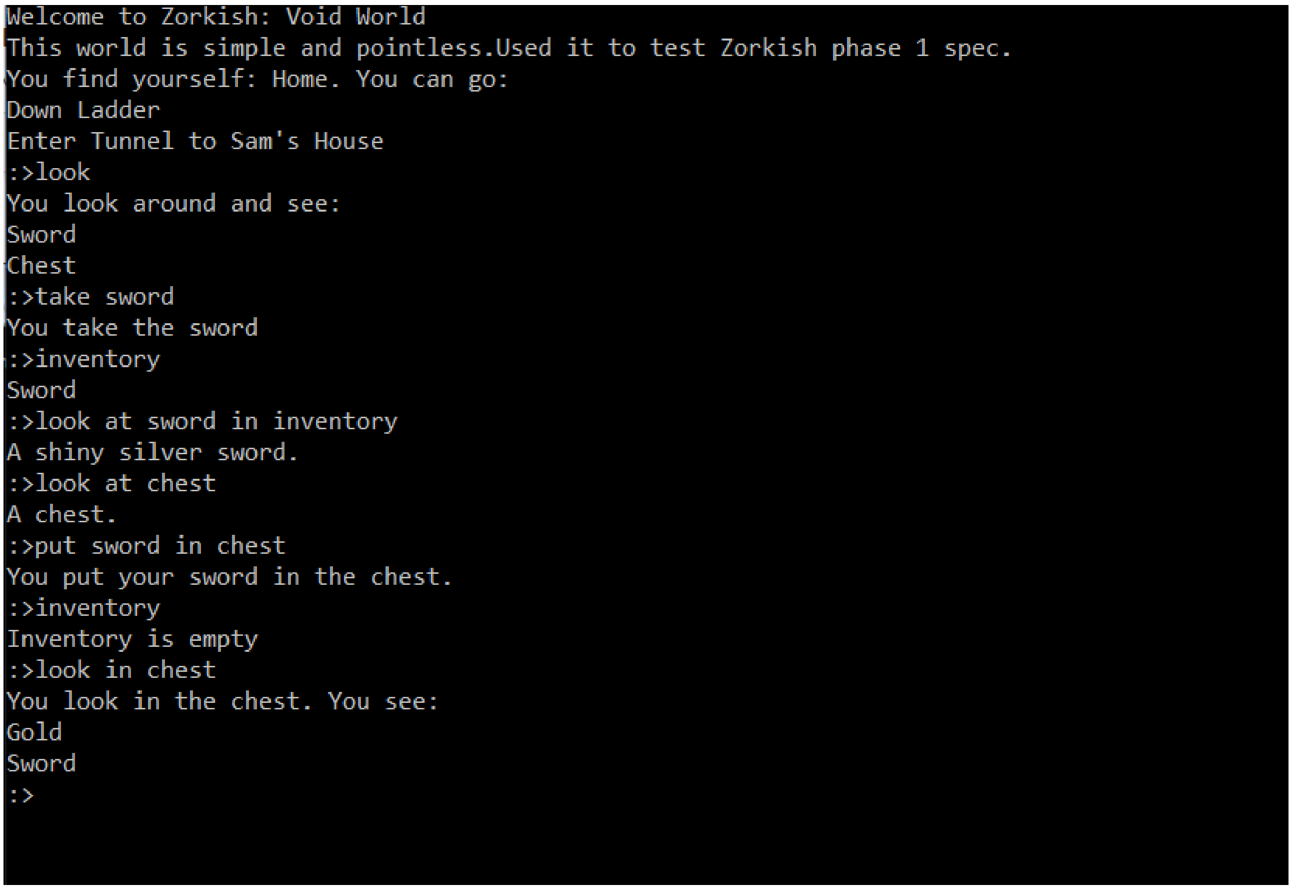
- Research the Composite Pattern and Component Pattern including when, how and why you would implement this pattern in particular scenarios.
- Practice implementation in small programs to understand the fabric of the pattern and how the classes link together.
- Transfer knowledge into the Zorkish game adding the classes required to support the pattern.
- Add additional classes where required to support the new functionality (Component class, Component Manager class).
- Refactor the old code (inventory, world processor, graph etc.) to make use of the new classes and methods implemented.
- Testing code to ensure it all works the same as before.
- Expanding the command processor to add additional functionality.

What we found out:

We found out how to implement the Composite Design pattern and what the advantaged of using this method is. This is allows for objects to be composed of other objects by using two child classes derived from the base class. One is a singular entity, the other is an entity which contains entities.

To implement the Composite entities, we chose to use a list of Entity pointers. This reduces the size of the data being passed around by passing pointers instead of whole data structures (much more efficient).

Through doing this exercise, we found it quite natural to use the pointers, even though it has proven confusing in the past, now we have learned first-hand how useful pointers can be.



```
Welcome to Zorkish: Void World
This world is simple and pointless.Used it to test Zorkish phase 1 spec.
You find yourself: Home. You can go:
Down Ladder
Enter Tunnel to Sam's House
:>look
You look around and see:
Sword
Chest
:>take sword
You take the sword
:>inventory
Sword
:>look at sword in inventory
A shiny silver sword.
:>look at chest
A chest.
:>put sword in chest
You put your sword in the chest.
:>inventory
Inventory is empty
:>look in chest
You look in the chest. You see:
Gold
Sword
:>
```

Component Pattern:

Once we implement component pattern we can dynamically change game entity behaviours at run time very easily.

So I have created a Component Manager which has an instance of each component. Each component then has a register (list) where entities can be registered as having that component.

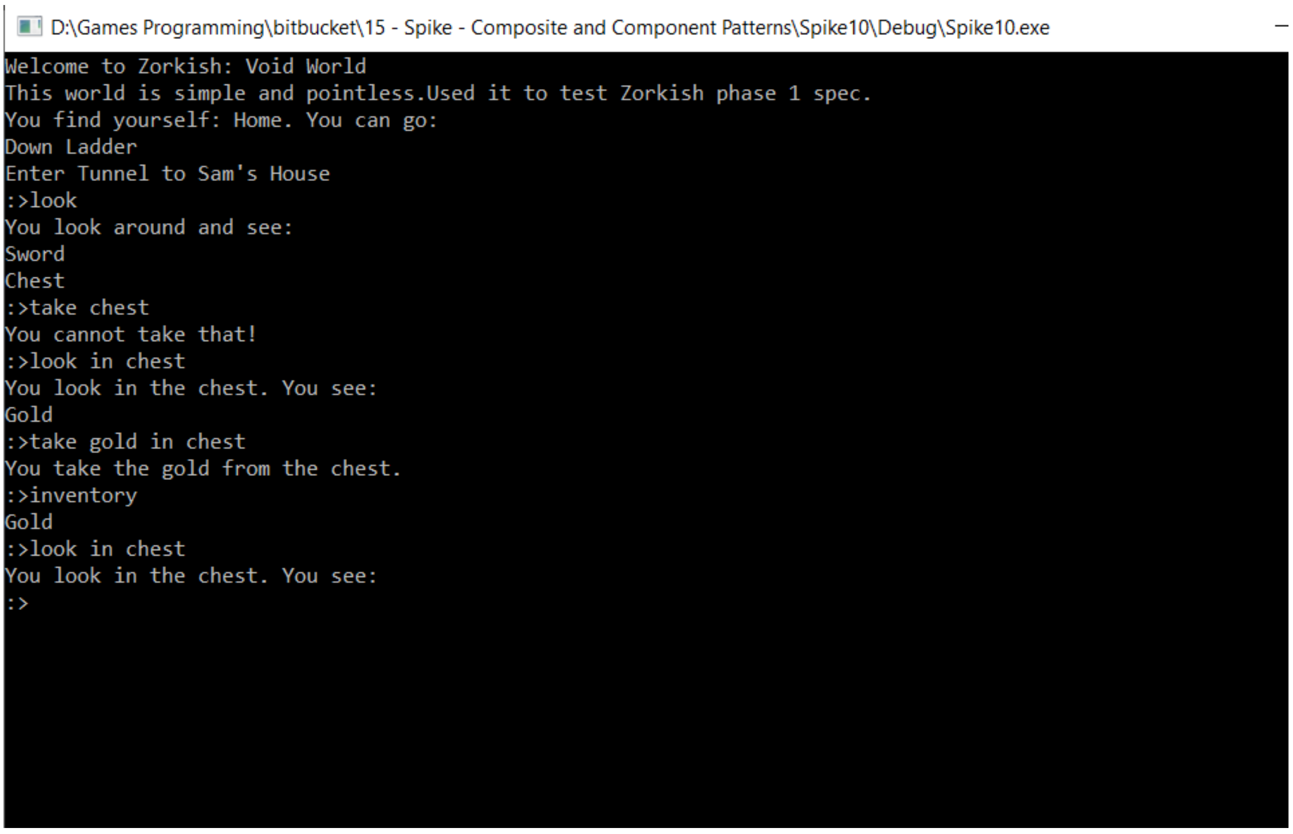
When the player tries to take an object, the Component Manager queries the component class to determine if the object can do that specific thing (if it is registered).

In the game world file, the chest is described like this:

```
<Item>
        <Name>Chest
        <Description>A magical chest.
        <ItemLocation>Home,
        <Container>True,
        <NumItems>1,
                <Name>Gold
                <Description>A bag of Gold.
                <Components>Takeable,
</Item>
```

This means that the gold can be taken from the chest as it has the "Takeable" component, however the chest cannot be taken as it does not have the "Takeable" component.

This can be shown below:



D:\Games Programming\bitbucket\15 - Spike - Composite and Component Patterns\Spike10\Debug\Spike10.exe

```
Welcome to Zorkish: Void World
This world is simple and pointless.Used it to test Zorkish phase 1 spec.
You find yourself: Home. You can go:
Down Ladder
Enter Tunnel to Sam's House
:>look
You look around and see:
Sword
Chest
:>take chest
You cannot take that!
:>look in chest
You look in the chest. You see:
Gold
:>take gold in chest
You take the gold from the chest.
:>inventory
Gold
:>look in chest
You look in the chest. You see:
:>
```