Spike: 10
Title: Component Pattern

Author: Michael Williams, 7668481

Goals / deliverables:

- Code
- To implement the Component Design Pattern
- To refactor old code in order to support the inclusion of entities being composites of their behaviour.
- To expand the file input behaviours of the program to support loading composite aspects of game world entities.

Technologies, Tools, and Resources used:

- Visual Studio IDE
- www.cplusplus.com
- Assorted web sources.
    - YouTube
    - Tutorials

Tasks undertaken:

- Research the Component Pattern including when, how and why you would implement this pattern in particular scenarios.
- Determine on an implementation strategy for the Zorkish game.
- Add additional classes where required to support the new functionality (Component class, Component Manager class).
- Refactor the old code (world processor, command classes etc.) to make use of the new classes and methods implemented.
- Testing code to ensure it all works the same as before.

What we found out:

We found out how the implement the component pattern in order to dynamically change game entity behaviours at run time very easily.

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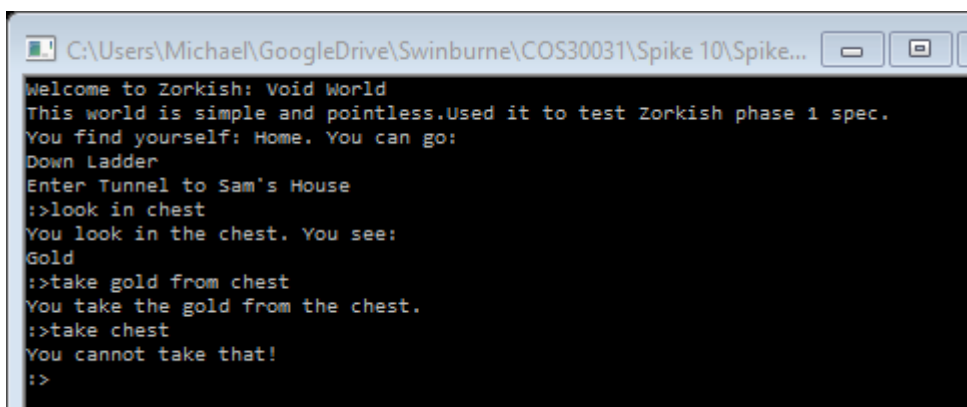
When the player tries to take an object, the Component Manager queries the component class to determine if the object can do that specific thing (if it is registered).

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

```
<Item>
        <Name>Chest
        <Description>A 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: