**Spike:** Task 15

**Title:** Composite and Component Patterns

**Author:** Sam Huffer, 101633177

# Goals / deliverables:

* Composite Pattern:
  + Adventure (world) files that include the specification of game entities, their properties, and any nested entities (composition) they may contain.
  + Players are able to observe and modify entities contents and location, with the Commands “look in”, “take from”, “put in” and “open with”.
* Component Pattern:
  + Game objects that receive attributes (damage, health, flammability, etc.) from component objects rather than inheritance
  + Game objects that receive actions (can be picked up, can be attacked, etc.) form component objects rather than inheritance.

# Technologies, Tools, and Resources used:

* Visual Studio 2019
* Microsoft Word

# Tasks undertaken:

* I copied the “Zorkish Adventure” project and the task 14 spike report into the task folder, stripping out the spike report’s original content and replacing it with goals and resources pertaining to the task at hand.
* I had a look at the task instructions, lecture notes, and examples to get an idea of how the component and composite patterns work, and what could be put together for each.
* Given the notes and examples, I have already completed the following deliverables:
  + A specification of a game world with game objects, their properties, items within containers, etc. (completed for Task 12: Game Graphs From Data)
  + CommandLook, CommandTake, and CommandPut (made for Task 10: Game Data Structures, and adapted to the command pattern properly for Task 14: Command Pattern)

That leaves the following to be completed for this spike:

* + World specification
    - Remove container item specification
    - Add component specification so that components can be added to Game Objects
  + Classes:
    - Component (each has a reference to its GameObject); derived classes:
      * Container
      * Lock (for lockable containers)
      * Movable (to register if an item can be moved)
      * Description
    - GameObject (has components, they can be added and removed); derived classes:
      * Location (has a container and a description component)
      * Player (has a container component)
      * Path (has a description component)
      * Item (has a description component, may have a movable and/or a container component)
    - CommandOpen (to open containers, using a key if they’re locked)
* I put together a UML class diagram of all the classes that would need to be added or have methods or fields swapped out (see below).
* I started implementing the component pattern with the Component base class, Container and Description Components, the GameObject base class, and Item, Player, Path and Location GameObjects, forward declaring in the Component and GameObject classes the other, and the Item class in the Container class.
* I went through all the Commands, finding all the errors created from the change to using Components, and fixed them all such that all the Commands used Components and GameObjects rather than members of their derived classes that no longer existed.
* I went through all the errors in World, correcting those that needed to line up with Components rather than non-existing members of classes, removing the checks for ContainerItem, as that has been removed from the project, and modifying other game object specifications and their formatting error messages to handle an updated text file formatting specification. I then went through the text file, modifying the formatting of game objects where necessary, and adding the details of what the formatting should be in a comment block at the start of the file.

# UML Diagram:

# What we found out:

* Barring CommandOpen, all of the deliverables listed for the composite pattern I seemed to have already implemented for other tasks. ContainerItems containing Items that might also be ContainerItems seemed to perfectly fit the descriptions of the composite pattern linked to in the lecture notes.
* The lecture notes and linked examples seemed to suggest that the component pattern can be easily used with the component pattern, such as with the aforementioned case of Items that might be Containers containing more Items.
* Going through where the component pattern could be implemented for this task in Zorkish, I considered making Paths inherit from Component, but I decided against that as I figured it would be best if GameObjects only had one of any given class of Component, and Locations would often need multiple Paths, not just the one.
* Lock could be a component of Container, but it’s weird for Container to only have a component field for Locks, so it makes more sense for Lock to be a component of item, and Container checks if its GameObject has a Lock component. This method requires Components to have a reference to their GameObjects.