**Spike:** 6

**Title:** Command pattern

**Author:** Ryan Chessum, 102564760

**Goals / deliverables:**

* Code, see /12 – Spike – Command pattern/Task 12/
* Spike Report
* UML Diagram

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

List of information needed by someone trying to reproduce this work

* Visual Studio 2019
* C plus plus reference (<https://www.cplusplus.com/reference/>)
* Zorkish game specifications
* Code from previous spikes

**Tasks undertaken:**

* Download and install Visual Studio
* Create a new C++ project
* Import code from previous spikes
* Implement game into PlayAdventure state
* Add Entities
* Make entities loaded from file
* Create Command class
* Create Command Processor class
* Create Child command classes
* Implement command processor into game

**What we found out:**

In order to process many different types of commands without making a giant if else block, we can make a command processor class that finds the command we are trying to use and executes the corresponding code.

To show this, I needed to create a proper implementation of Zorkish that had a graph world with entities. The first thing I did was import the files for the classes we needed including the state related classes and classes that inherited from the identifiable object class like locations and the player.

Text

Description automatically generated

First, I made a basic class to use for entities. It’s like the item class. I then added a vector of entities in the location class so that locations can store entities within them and a function to list all the entities held in the vector as a string.

Text

Description automatically generated

After we had that I copied the adventure loading code from the previous spike into a new function the PlayAdventure state and called it LoadAdventure, it takes the filename as a parameter.

PlayAdventure also stores a Boolean variable to keep track if an adventure is loaded. If one is loaded, then the Zorkish game will run otherwise the program will run the menu to select an adventure. When an adventure is selected it loads an adventure and then sets the Boolean variable to true. It turns back to false when you quit the game.

Text

Description automatically generated

Then I added some entities to the adventure file. I signify them with an E and are loaded in the same way as a location as their constructors take the same parameters. When the load function loads in an entity it will add it to the last location that it loaded. So, like the connections entities you want to have in a location should be placed underneath the desired location.

Text

Description automatically generated

Now we have a basic version of the game where the player can traverse through the locations filled with entities. From here I Implemented the command processor.

First, I made a basic command class which each command will inherit from. It is a child class of identifiable object. This will let us use Ids as the command sting. So, if the command processor finds a matching id at the start of the players input string it can call the execute function of the corresponding command.

A screenshot of a computer

Description automatically generated with medium confidence

Next, I set up a command processor class. It also has an execute function.

A picture containing graphical user interface

Description automatically generated

Before I set up the rest I went back and created the command subclasses for all the commands I wanted to add.

First is the go command which works pretty much the same way it did before.

Text

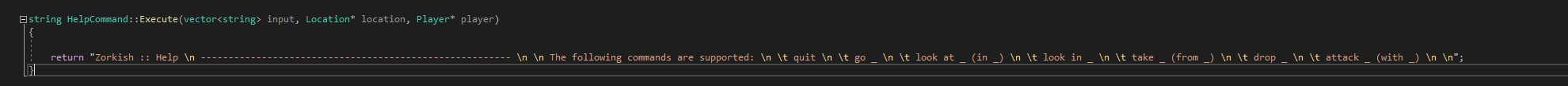
Description automatically generated

Then I added the look command which checks all of the entities int the location to see if any of their ids match the 3rd input string. It also to see if that string is the player.

Text

Description automatically generated

The help command prints the same help message as the one on the main menu.



The debug command is set up for debug related commands. At the moment if the second word is tree it prints off the location details along with the list of entities stored there. As well as each connected location and their entities.

Text

Description automatically generated

The alias command adds ids to commands so other strings can be used as command identifiers. To do this it needs to hold a reference to every command in the command processor so it can check the up to date identifiers on all of them. When the command processor is constructed, it will pass a vector of references in so that they can be used in the command. When the command is executed, it will check all the commands to see if it exists then checks the alias you want to add and makes sure it’s not an already existing command. If a command with that string doesn’t exist it will add that identifier to the specified command.

Text

Description automatically generated

The inventory command simply lists the players inventory.

Text

Description automatically generated

Lastly the quit command gets passed in the loaded variable from the constructor. When the player chooses to quit it will change it back to false and set the state to QUIT.

A screenshot of a computer

Description automatically generated

Now we can go back to the command processor and add a member of each command type as a variable.

Text

Description automatically generated

When the processor is constructed, it will set all of the ids for each command and add a reference of them to the command vector.

Text

Description automatically generated

When a command is executed, the processor checks the first input string to see if it is a command. If so, it will pass the input string vector, location and player pointers into that command’s execute function, executing the command.

Text

Description automatically generated

Then we need to use this in the update function. I added the Command processor as a member of the play adventure state.

Text

Description automatically generated

Then I called the execute function in update.

Text

Description automatically generated

Now if you try playing the game you should be able to use all the commands we added.

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated