Spike: 08
Title: Command Pattern

Author: Ford Killeen, 9731822

Goals / deliverables:
Goals this spike aims to achieve:
- Extend upon Zorkish by adding a command processor
- Load adventure files with partial game entities

Deliverables required:
- Code showcasing the command processor in use
- Spike report

Technologies, Tools, and Resources used:
The following is required to complete this spike:
- Visual Studio 2015
- Zorkish game specification
- Online C++ references
- A text file adventure of your own making
- A thesaurus to aid with choosing aliases

Tasks undertaken:
The list below details the steps taken to complete this spike.
- Firstly I designed the classes that would form my command pattern, ie. the `Command` and `CommandManager` classes. The command manager would hold a list of `string` command/aliases mapped against the `Command` that would handle that command, so that when the manager is given a string of commands it can determine who gets to handle it.

```
class Command
{
public:
    virtual void process(vector<string> cmds) = 0;
    virtual vector<string> getAliases() = 0;
};
```

```
class CommandManager
{
private:
    map<string, Command*> commandMap;
public:
    CommandManager();
    void processCommand(string command);
};
```

- After setting up these two main classes, I started the implementation with the `GoCommand` class. This handles all movement aliases and directions within the game world.
- This was tested and refined.
- I then moved on to add the `LookCommand`, `HelpCommand`, `PickupCommand` and `UseCommand` classes to the game.

What we found out:
By completing this spike we found out the usefulness of the Command Pattern, as it enables us to easily add new command modules to take care of very specific and focussed tasks along with the easy addition of any aliases that go with that command.

Issues:
This is not so much an issue but more that when completing this spike, I didn't add any extra functionality to my adventure loading code as I figured this would be better handled in the future spike. Because of this I kept some of the command classes quite simple for now (it just prints the action you intended on completing), as I'm not sure how I will design these future entities yet. See image below.

```
void PickupCommand::process(vector<string> cmds)
{
    if (cmds.size() > Sizes::_one)
        cout << "You pickup " << cmds[1] << endl;
    else
        cout << "Pickup what?" << endl;
}
```