Spike: Task 9
Title: Game State Management
Author: Sam Huffer, 101633177

Goals / deliverables:

- A simple plan of the code design of phase 1 of the "Zorkish Adventure" game
- A console program that implements phase 1 of the "Zorkish Adventure" game, using a flexible and extensible game state management method of some kind, without gameplay yet. Game stages demonstrable at this stage must include:
    - Main Menu (to select other stages)
    - About (including the student's own details)
    - Help (hard-coded summary of commands)
    - Select Adventure (hard-coded list and the title of the test game)
    - Gameplay (test game only accepting "quit" and "hiscore" commands)
    - New High Score (let user input their name, but doesn't work yet)
    - View Hall of Fame (view hard-coded list of name/score)

Technologies, Tools, and Resources used:

- Visual Studio 2017
- Microsoft Word
- Draw.io

Tasks undertaken:

- Read through the spike instructions and the Zorkish phase 1 specifications to get the gist of what the program requires.
- Created a class diagram in draw.io to outline what, at this stage, I figured the program would require:

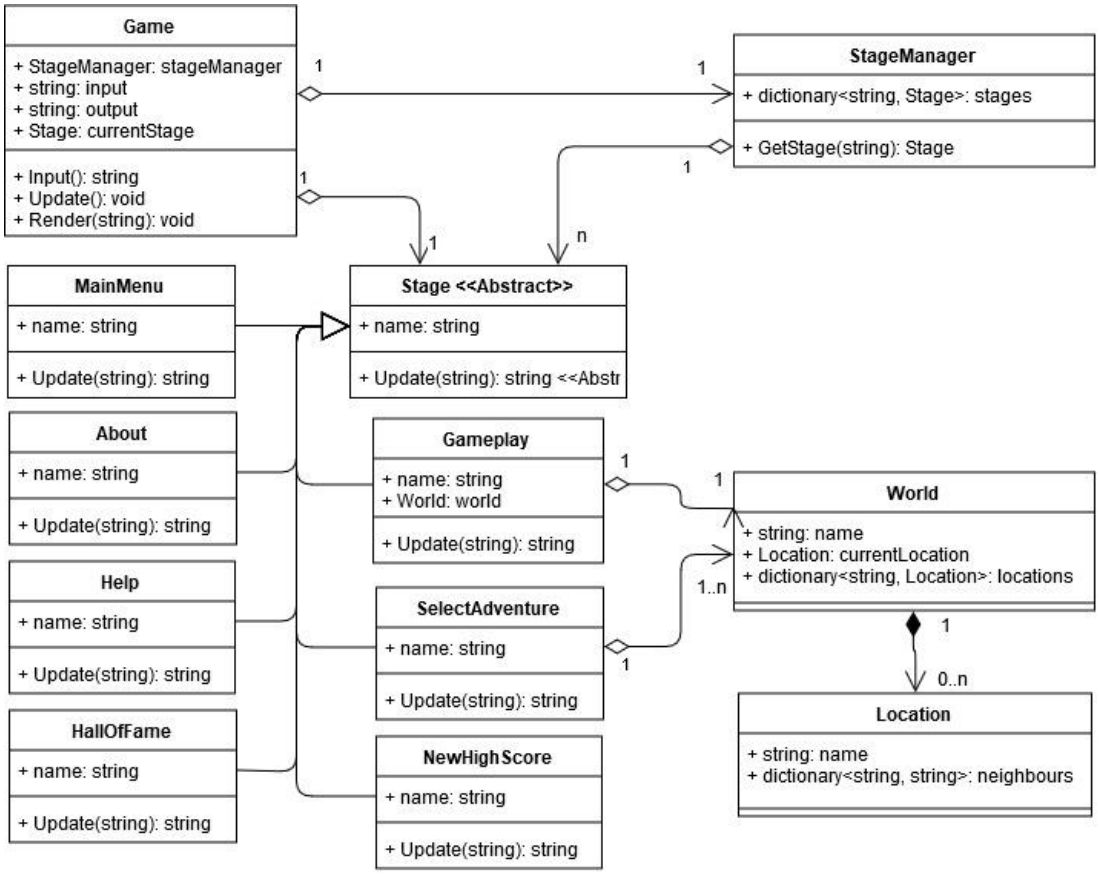

Figure 1: preliminary UML class diagram for phase 1 of the Zorkish Adventure game

- Created a Visual Studio project with .cpp files for each class, set up the main() method to create and run the Game class, and set up the basic Input(), Update(), and Render() loop for the game.
- Fleshed out the Game, Stage, MainMenu, and StageManager classes enough for a very basic test of the game's functionality to ensure I had everything plugged in correctly. Upon running this version, I ran into a number of errors screaming about no or too many includes and definitions of files and classes.
- Reorganised the code to have class definitions in .h files and the method implementations in .cpp files and reworked the #include statements, and ran it again, but it still kept complaining about further errors.
- Did further research to sort out how to organise the files and use #includes and forward declarations properly (http://www.cplusplus.com/forum/articles/10627/), and came across other content giving good practice tips (https://accu.org/index.php/journals/1995). It took a while and some trial and error, but eventually I sorted out the #includes and forward declarations so that the game ran properly. While doing this, I also cleaned up some interactions so that the code displayed output properly, and did some setup for the implementations of Stage.Update() when it would be called for the first time after being assigned to Game.currentStage. Between errors and other classes, that took me from the Tuesday Tutorial until Saturday morning to complete.
- I updated Game to follow the singleton design pattern (https://sourcemaking.com/design_patterns/singleton/cpp/2) so that I could access it at will from other classes where necessary. Once I had that working properly, I updated MainMenu to be able to send the user to About when it was selected; that required some adjustment of Game.Update() to immediately and correctly display the output of About.Setup() rather than rendering and waiting for input needlessly again.
- Once About was displaying its information properly, I noticed Game.Input() wouldn't register the enter key as a distinct input when using "std::cin >> input;" to take input. I looked up how to get around that, and tweaked it accordingly to be "std::getline(std::cin, input);" (https://stackoverflow.com/questions/903221/press-enter-to-continue).
- Next, I added a Quit() method to Game to set finished to true, and modified MainMenu to call it and produce appropriate end of game text.
- I created the Help and HallOfFame classes, deriving their code from About.h and About.cpp, making the appropriate changes to make each its own class providing the appropriate output.
- I created the Gameplay class on the same basis, modifying its code to display options for available worlds, and accept input for choosing a world. I also added the World and Location classes, adding enough implementation that the World could include multiple Locations later on, and that Locations could link to their neighbours and provide a description of the current Location.
- Next, I added functionality to Gameplay.Update() to accept the "quit" command to return to the main menu, and the "hiscore" command to proceed to the NewHighScore stage.
- I created the NewHighScore class, configuring its Setup() method to display the appropriate filler text, and the Update() method to return the player to MainMenu as soon as they entered their name.
- Lastly, I cleaned up some of the text to consistently leave a blank line between the end of the output of one stage and the start of the output of the next stage, and edited Stage.Setup() to not require a string parameter, as none of Stage's child classes were using that parameter in their Setup() methods.

What we found out

- How to organise C++ code across multiple files, splitting definitions into .h files and implementations into .cpp files, and #include and forward declare classes from other files properly such that the program works and the compiler doesn't scream at you about errors you have no idea how you caused them.

- How to store child classes in a list of their base class in C++.
- How to implement the singleton design pattern in C++, and call the created instance of the singleton class.
- Which input-gathering method to use to accept the enter key as input.