Spike: Task 08
Title: Zorkish - 1

Author: Hayden Whiteford, 104001272

Goals / deliverables:
[DESIGN DOC] + [CODE] + [SPIKE REPORT]
You need to create the “Zorkish Adventure” game (Phase 1), as described in the specification document available on the unit website.

Technologies, Tools, and Resources used:

- draw.io for the plan
- Xcode
- https://www.codeproject.com/Articles/1374/State-Pattern-in-C-Applications - state management example

Tasks undertaken:
List key tasks likely to help another developer
This section should resemble a tutorial – the goal is to allow another coder to reproduce your work following these steps.
Eg: (Good)

- Download and install Visual Studio
- Download and install DirectX
- Configure VS Project File to point to the DX lib folder
- Compile sample code
- Using the State Management Example, create a plan for the classes involved
- Create a new Xcode project
- Implement the headers for each class and resolve any dependancy issues
- Implement classes Game, GameManager, and the abstract class GameState
- Make each individual state based on the GameState class and tweak to fit each page/output
- Run!

What we found out:
The State Manager Example already had some interesting challenges to overcome, for starters, many of the classes in the example were rather useless in this context, so I managed to simplify it down to just 3. Furthermore there was a problem in the way that inputs were handled. It became quite apparent that in the example, the input is being checked every “frame” that would occur, and was capable of continuing the game loop if no input is found. For this context I wanted to use a simple std::cin to handle inputs, which wait until an input is given, so I would need to think about that

too. Lastly, I really learnt how to use header files in order to fix dependancy issues. Before this, I had no idea how to fix my GameStateManager and my GameClass needing to know about each other, so I needed to learn about forward declarations.

Open issues/risks:

- The way inputs are handled are not all the same across game states. Some use the get() function for characters, and others use getline() and some std::cin >> fInput, depending on the use case. In future, id like to standardise this as much as I can
- The actual logic for what gamestate comes next is handled within the TakeInput() function of the GameState. I think I would prefer to put this inside the GameStateManager, or create a new function for the GameState altogether
- No enumerators are used. After TakeInput() decides the next game state, it simply returns a new instance of the class to the GameStateManager to set the Current Class.