**Spike:** 3

**Title:** Game State Management

**Author:** Ryan Chessum, 102564760

**Goals / deliverables:**

* Code, see /08 – spike - Game State Management/Task 8 Zorkish States/
* Simple plan/design outline for the program
* Spike Report

**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

**Tasks undertaken:**

* Download and install Visual Studio
* Create a new C++ project
* Create a basic design for the code
* Create a basic game loop structure
* Create a state class
* Create an enum for all the states
* Create a state manager to change the behaviour of the program by calling the update and render functions of different states.
* Create state subclasses for every state and define the behaviour of their update and render functions

**What we found out:**

A game state manager is a great way of managing the states or stages of a game. States can help us determine what a game should be doing. For example is the player in a menu, playing the main game, etc. A class based state manager system is an efficient approach for implementing a state manager.

To determine the current state of the game we can create a state manager class. Which will get the current state and call update and render functions that we need for each state/stage. This also can help us with code organisation and abstraction.

Here is what the main function looks like using a state manager.

Text

Description automatically generated

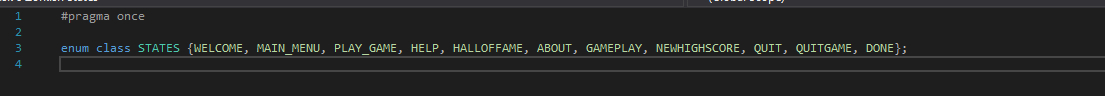
It’s a very simple looking game loop. What’s happening is we have an instance of a state manager which is filled with state objects. Update and render are called by getting the current state of the state manager and calling the member function for update/render of that state.

For this to work we need a base state class. It should be very basic having only update and render functions. We are going to use this as a parent class and the children of this class will have definitions for Update and Render. So, each function should be virtual so they can be overridden in the child states.

Text

Description automatically generated

Next create an enum containing each state we want to add. As well as a global variable of the type of your states enum for the current state. This is what is used to change/get the game state. Note that the state variable has to be defined as an external variable in each file so that each file knows that it exists.



Then we create a child class for each state we would like to have in our game. Override the update and render functions and define the behaviour for each one.

Text

Description automatically generated

As an example, here is the main menu state. Render lists the options the player has on screen while update collects an input from the player and updates the global state based on the input. You can define whatever you want in there but keep in mind you probably want to have a way of going back to the main menu, even if it’s going to a state that can take you there.

Text

Description automatically generated

Once you have all of your states, we need a state manager to hold and manage them all. The state manager class should look something like this. It should have instances of all of your states, a state (class not enum) pointer that points to the current state and a function that can give the current state.

Text

Description automatically generated

The Current function should look like this. Depending on the state set on the global variable it will change the current variable to point at the corresponding state.

A screenshot of a computer screen

Description automatically generated with medium confidence

Now that you have your state manager go back to main and create your gameloop by accessing the current state and calling the update and render function through the pointer.

Text

Description automatically generated

If done correctly everything should work.

Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence