Spike: Spike No.8
Title: Task 8 – Game State Management

Author: Khang Trinh - 102118468

Goals / Deliverables:

The goal of this spike is to explore how implement game states to

Technologies, Tools, and Resources used:

- Visual Studio 2017

Useful Links:

Tasks undertaken:

Step 1: Plan out the transition between states

It's worth drawing out a diagram showing the relationship between the states so that it can help you figure out how to code their functionalities.

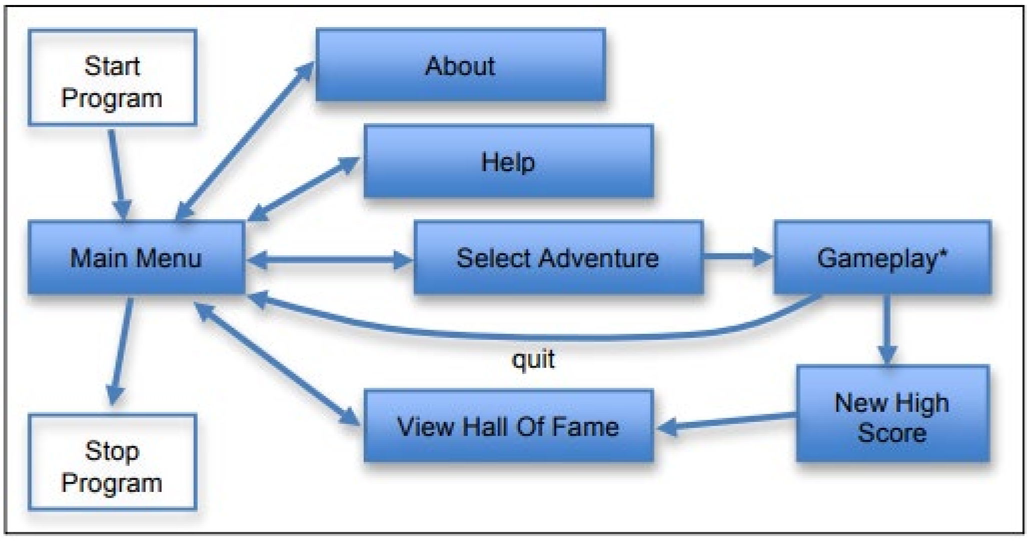
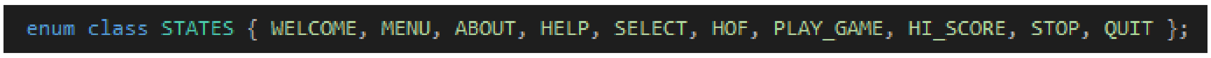


Fig 1. State diagram for Zorkish Adventures

Step 2: Create an enum holding all the possible states in the game

If there's a state in the game that will be doing something, they'll need to be in this enum so that the game can switch to.

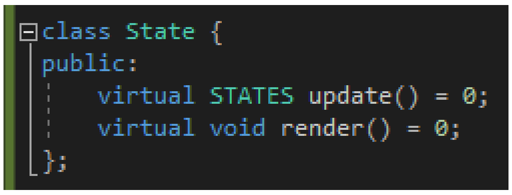


```
enum class STATES { WELCOME, MENU, ABOUT, HELP, SELECT, HOF, PLAY_GAME, HI_SCORE, STOP, QUIT };
```

Fig 2. An enum holding all the states

Step 3: Create a base class for the states

Every state the game changes to, it'll do something different, and as a result would need to look different. Therefore, every state should have an update and render function. To do that the most efficient way, we'd need to create a base class with these functions that the states can inherit from. *Note that the update function has a return type of the enum, it'll be explained.*

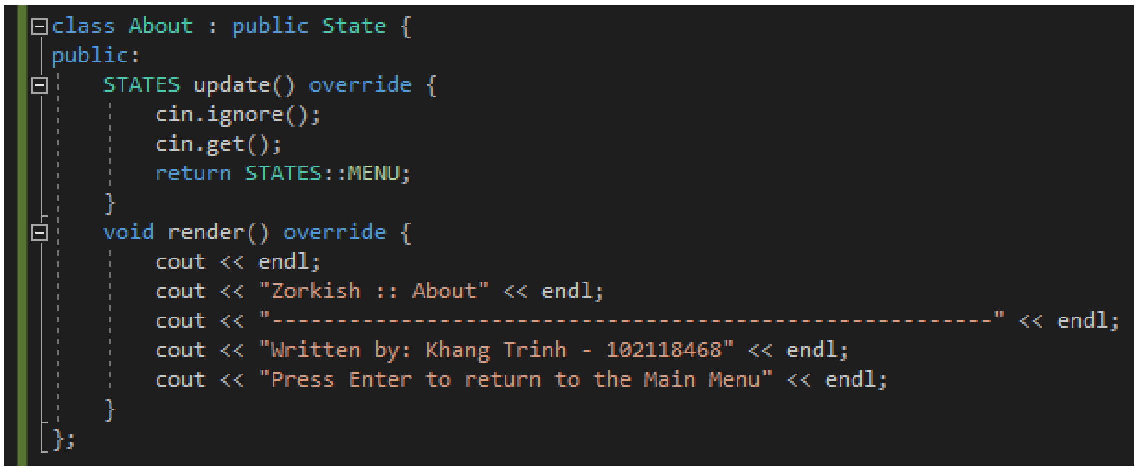


```cpp
class State {
public:
    virtual STATES update() = 0;
    virtual void render() = 0;
};
```

Fig 3. Base class for states

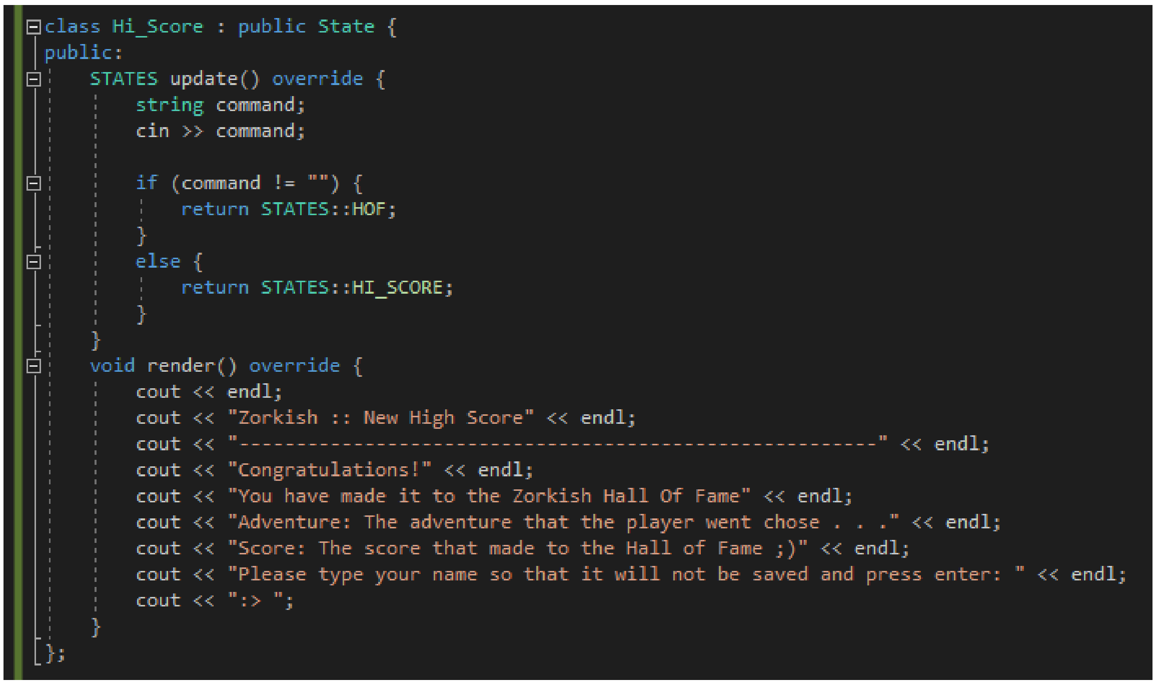
Step 4: Create the states

Now that we have a blueprint to create the states, we can use that to add functionalities to all the states.



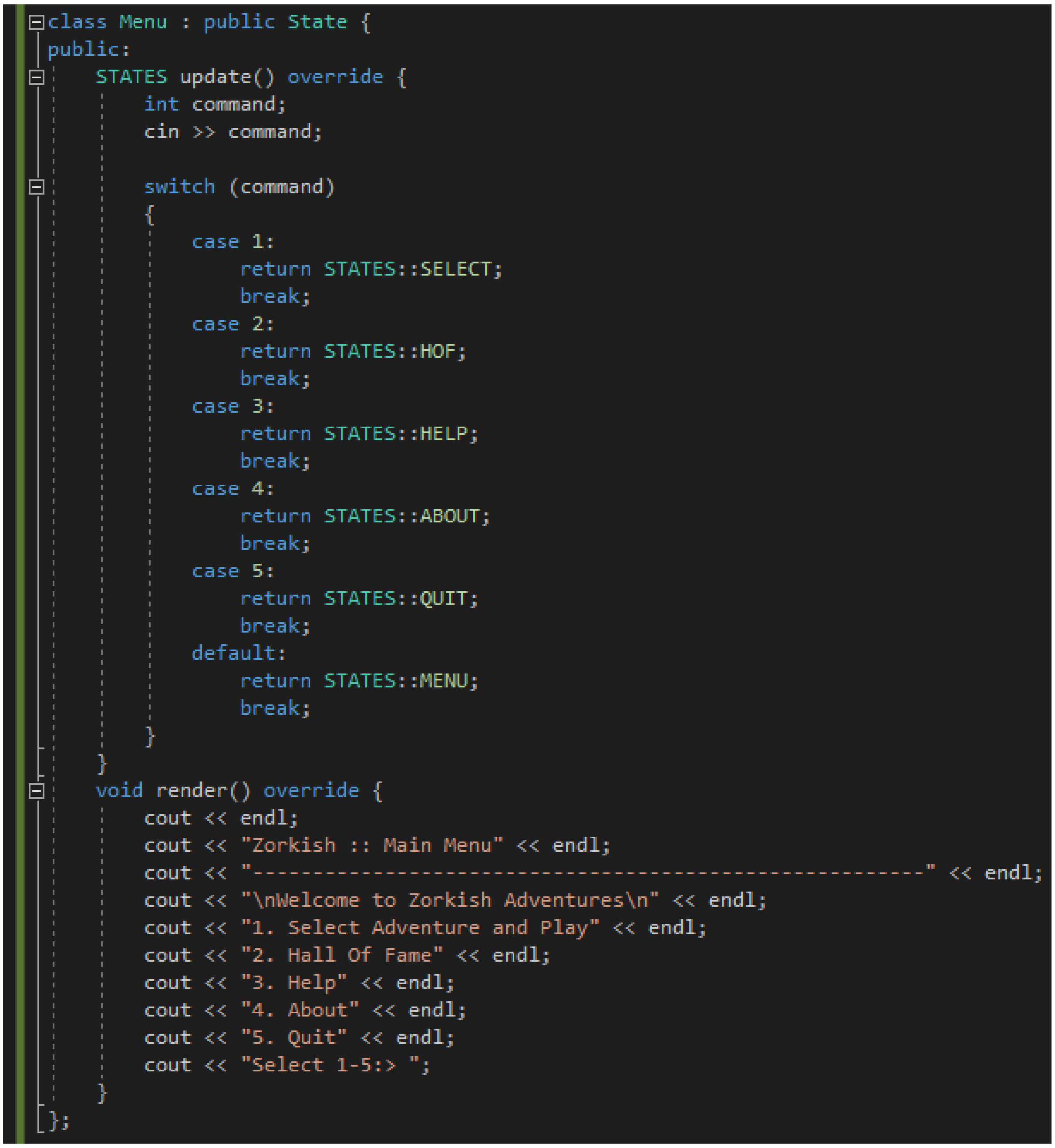
```cpp
class About : public State {
public:
    STATES update() override {
        cin.ignore();
        cin.get();
        return STATES::MENU;
    }
    void render() override {
        cout << endl;
        cout << "Zorkish :: About" << endl;
        cout << "---------------------------------------------------------" << endl;
        cout << "Written by: Khang Trinh - 102118468" << endl;
        cout << "Press Enter to return to the Main Menu" << endl;
    }
};
```

Fig 4. An example of a state that doesn't do much other than display info, and proceeded by hitting Enter



```cpp
class Hi_Score : public State {
public:
    STATES update() override {
        string command;
        cin >> command;

        if (command != "") {
            return STATES::HOF;
        }
        else {
            return STATES::HI_SCORE;
        }
    }
    void render() override {
        cout << endl;
        cout << "Zorkish :: New High Score" << endl;
        cout << "--------------------------------------------------------" << endl;
        cout << "Congratulations!" << endl;
        cout << "You have made it to the Zorkish Hall Of Fame" << endl;
        cout << "Adventure: The adventure that the player went chose . . ." << endl;
        cout << "Score: The score that made to the Hall of Fame ;)" << endl;
        cout << "Please type your name so that it will not be saved and press enter: " << endl;
        cout << ":> ";
    }
};
```

Fig 5. A state that takes any string input

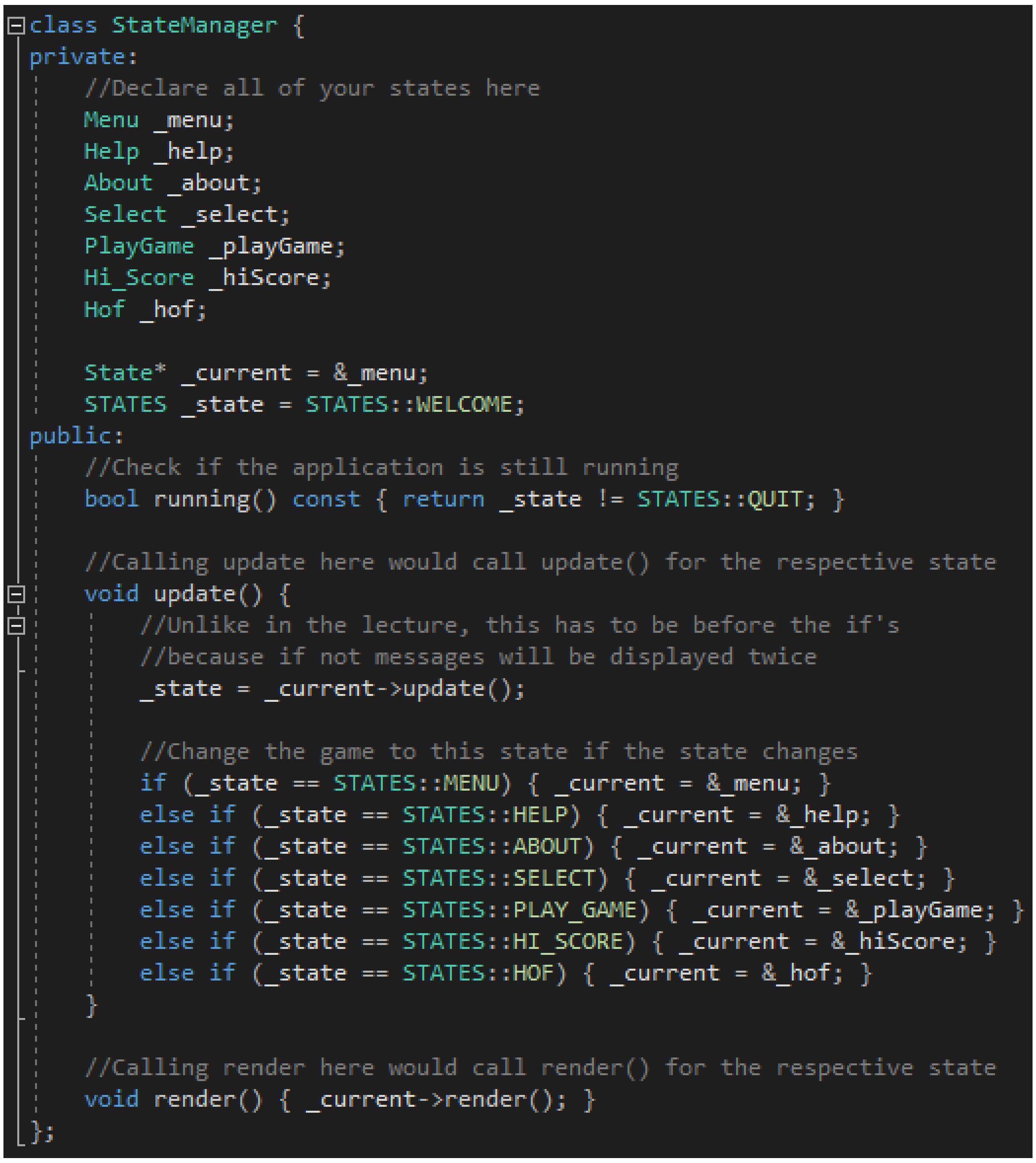


```cpp
class Menu : public State {
public:
    STATES update() override {
        int command;
        cin >> command;

        switch (command)
        {
            case 1:
                return STATES::SELECT;
                break;
            case 2:
                return STATES::HOF;
                break;
            case 3:
                return STATES::HELP;
                break;
            case 4:
                return STATES::ABOUT;
                break;
            case 5:
                return STATES::QUIT;
                break;
            default:
                return STATES::MENU;
                break;
        }
    }
    void render() override {
        cout << endl;
        cout << "Zorkish :: Main Menu" << endl;
        cout << "--------------------------------------------------------" << endl;
        cout << "\nWelcome to Zorkish Adventures\n" << endl;
        cout << "1. Select Adventure and Play" << endl;
        cout << "2. Hall Of Fame" << endl;
        cout << "3. Help" << endl;
        cout << "4. About" << endl;
        cout << "5. Quit" << endl;
        cout << "Select 1-5:> ";
    }
};
```

Fig 6. Sometimes it's worth using a switch statement to save time and effort

Step 5: Create a state manager class

Now that we've created all of the needed states, we need to create a class to manage all of these states (called a manager class) to tell the game when it can switch to what state.

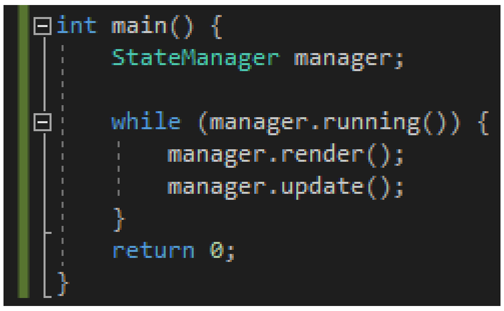


```
class StateManager {
private:
    //Declare all of your states here
    Menu _menu;
    Help _help;
    About _about;
    Select _select;
    PlayGame _playGame;
    Hi_Score _hiScore;
    Hof _hof;

    State* _current = &_menu;
    STATES _state = STATES::WELCOME;
public:
    //Check if the application is still running
    bool running() const { return _state != STATES::QUIT; }

    //Calling update here would call update() for the respective state
    void update() {
        //Unlike in the lecture, this has to be before the if's
        //because if not messages will be displayed twice
        _state = _current->update();

        //Change the game to this state if the state changes
        if (_state == STATES::MENU) { _current = &_menu; }
        else if (_state == STATES::HELP) { _current = &_help; }
        else if (_state == STATES::ABOUT) { _current = &_about; }
        else if (_state == STATES::SELECT) { _current = &_select; }
        else if (_state == STATES::PLAY_GAME) { _current = &_playGame; }
        else if (_state == STATES::HI_SCORE) { _current = &_hiScore; }
        else if (_state == STATES::HOF) { _current = &_hof; }
    }

    //Calling render here would call render() for the respective state
    void render() { _current->render(); }
};
```

Fig 7. An example of a state manager class

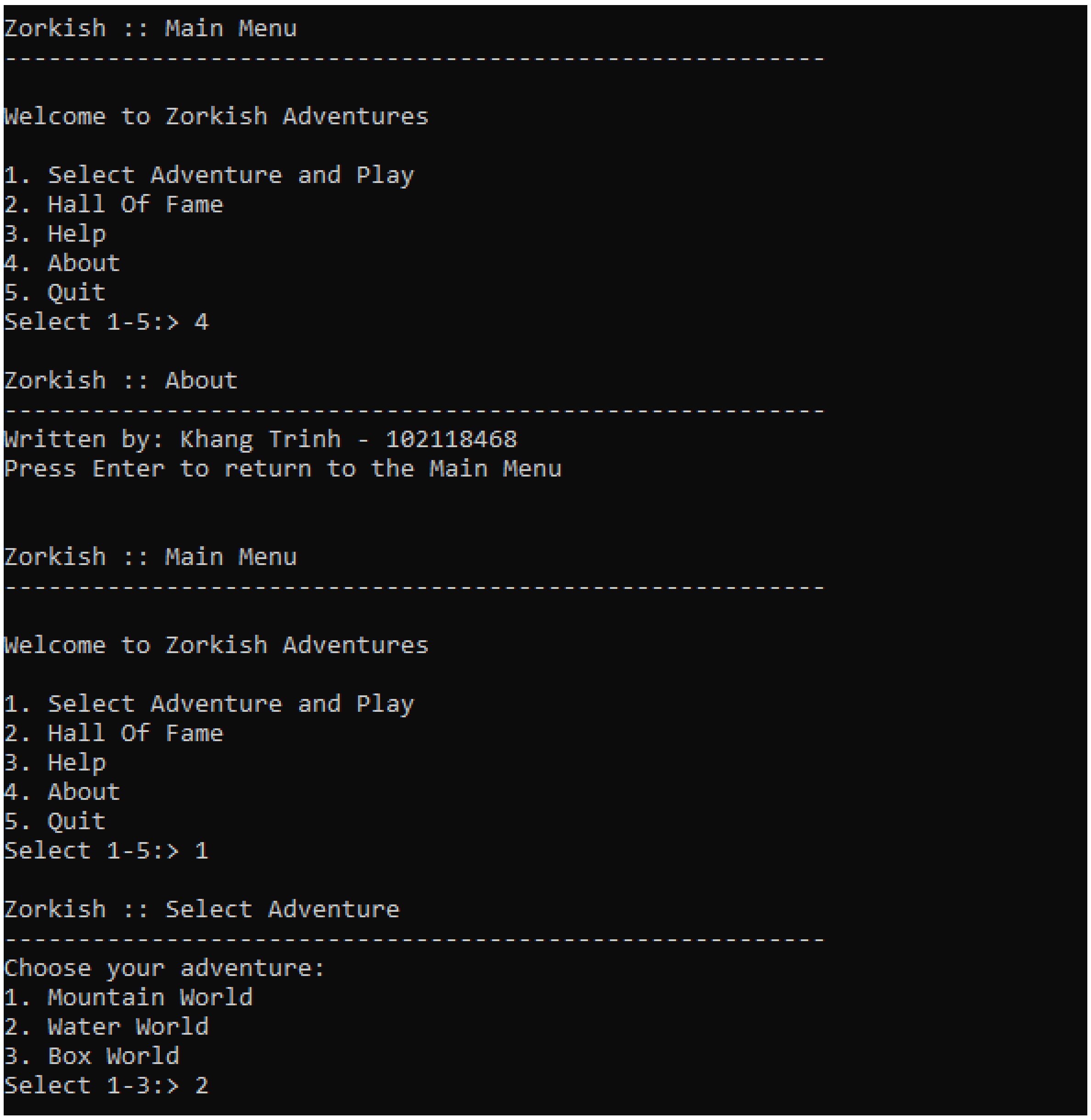
Step 6: Tie it all together

If you've done all the steps above correctly, main should be trivially easy to implement.



```
int main() {
    StateManager manager;

    while (manager.running()) {
        manager.render();
        manager.update();
    }
    return 0;
}
```

Example output



```
Zorkish :: Main Menu
--------------------------------------------------------

Welcome to Zorkish Adventures

1. Select Adventure and Play
2. Hall Of Fame
3. Help
4. About
5. Quit
Select 1-5:> 4

Zorkish :: About
--------------------------------------------------------
Written by: Khang Trinh - 102118468
Press Enter to return to the Main Menu


Zorkish :: Main Menu
--------------------------------------------------------

Welcome to Zorkish Adventures

1. Select Adventure and Play
2. Hall Of Fame
3. Help
4. About
5. Quit
Select 1-5:> 1

Zorkish :: Select Adventure
--------------------------------------------------------
Choose your adventure:
1. Mountain World
2. Water World
3. Box World
Select 1-3:> 2
```

Fig 8. What the output should look like for Zorkish Adventures