STEWART GRANT

101082708@student.swin.edu.au

Games PRogramming Custom Project Distinction Report

Or how I learnt to stop worrying and Love C

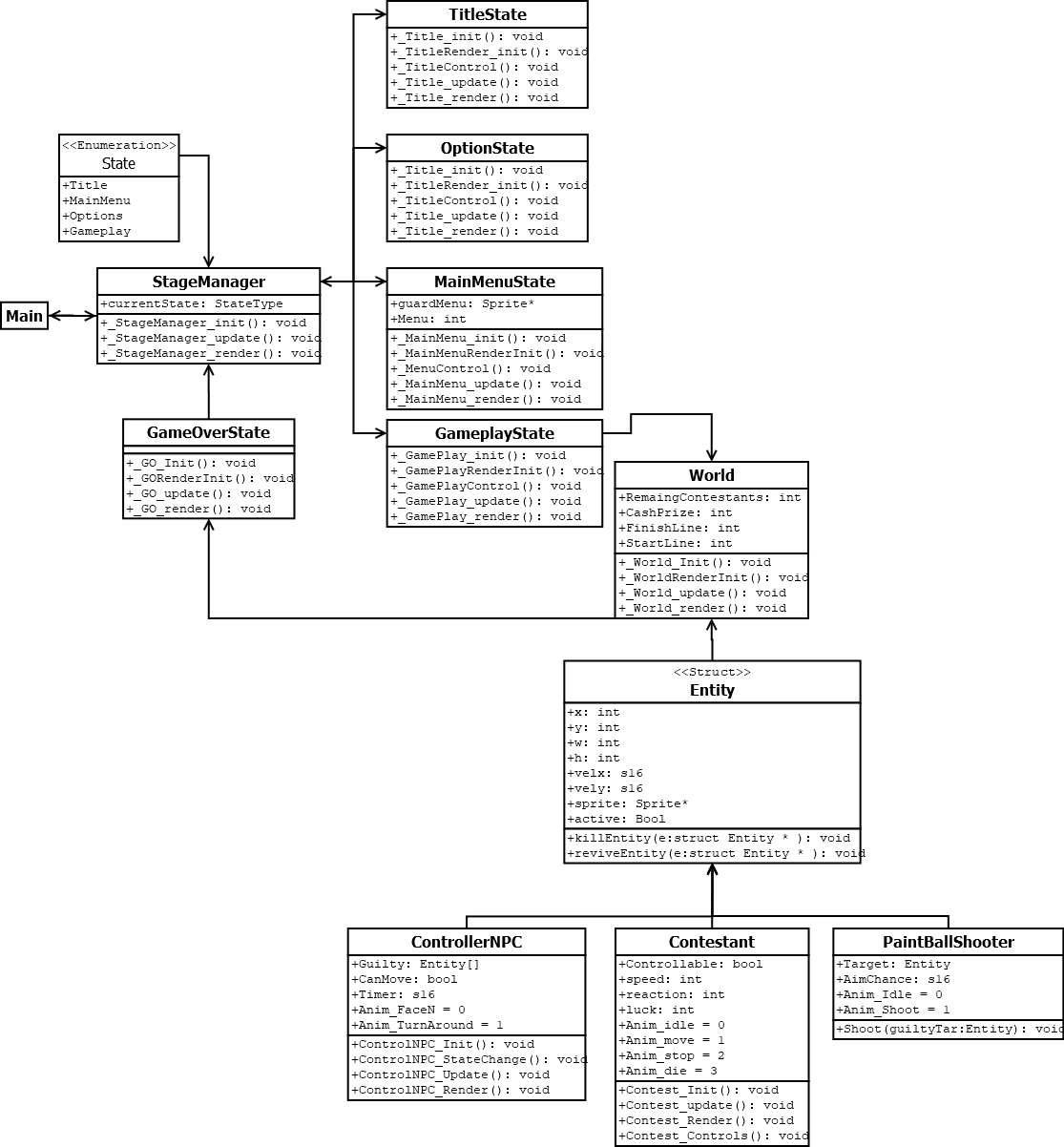
**Technologies, Tools, and Resources used:**

* Visual Studio Code
* Git command Line
* Stack Overflow
* Sega Genesis Development Kit (SGDK)
* SGDK Help Discord

**Tasks undertaken:**

Before attempting any programming, a basic UML is drafted of the design of the project. This UML graph will help with development of the system and ensures that an outline is followed however there will be some deviations from the intended design. Using the UML graph as the design, the steps involved would be to:

1. first get the Main Game Loop running,
2. To get the Game States functional and ensure each C script is interacting with the game loop
3. Get the Non-Gameplay specific states working
4. Get the world and the NPC Controller functional
5. Get the Contestants and Guards working.
6. Get the Messaging System working to send messages across the system.



The next step taken was to get the Game Loop and States working via enums and through various scripts. Since SGDK runs in C, many luxuries of C++ are absent so alternative methods are required. Since Class encapsulation was not an option, I opted to use a function pointer that takes in a Enum variable called current State to select the appropriate Game State.

A screenshot of a computer

Description automatically generated with medium confidence

The Game Loop doesn’t need to quit like earlier pass tasks so it doesn’t need a Boolean to exit the loop. However, SGDK specific functions need to be called beforehand. To get Game States working, an Update and Render class are created in the StageManager script and are called in the main game loop.

The various game states are now assigned their own scripts and are filled with the init, init\_Render, update and render script accordingly. In C, I thought it functions with similar sounding names would not be an issue since they are in their own script however some research and compilier errors taught me otherwise. To solve that, each function is given a unique name related to the script they are a part of.

Before attempting to get the game working, the other game states are implemented. The Menu, Title screen and Options are implemented to ensure the Function pointer in Stage Manager is finding the appropriate function.

**What I found out developing this project:**

When tackling the task, we wanted to expand functionality without the heavy use of inheritance and to achieve that, the composite and component patterns are utilised to achieve that.

**Issues:**

Some of the biggest issues faced developing the custom project was not the limitations of the hardware but adjusting to C’s lack of OOP concepts and having to find Procedural alternatives to the concepts taught from both OOP and this unit.

That’s not to say there were no hurdles with the systems limitations.

Graphical user interface

Description automatically generated with low confidence

To explain what is going on here, the idea was to have over 32 Contestants on screen however, the mega drive can only display 80\* Hardware sprites at a time. I say I had 32 sprites however that’s not the entire truth since the Mega Drive renders sprites via 8x8 tiles which combined create the entire sprite. For example, a sprite of Sonic would consist of 1x2 tiles to create his sprite while a sprite of Dr Eggman would may be 12x12 Tiles. Simply out, I wasn’t trying to draw 32 sprite/tiles, I was attempting to draw 32 sets of 4x4 Tiles so the options were to shrink the sprites or reduce the amount of entities.

**Screenshots (Post Repository creation):**