**SOFT10101: Computer Science Programming**

**“It’s High Noon!” – A Game Project Made Using C++ With the SFML Library**

To show what I have learned using during term two, I have been given the task of creating a project using the C++ language, that must include the use of classes or structs, file handling and arrays or other collections of data items, such as vectors.

To that end, a simple game that has a varying degree of complexity when it comes to implementing some of the features, seems like a perfect choice.

**Description of the Project**

As I have never worked with a graphical library or graphics of any kind, I want the project to be 2D based game, to give me a new challenge. The game itself will be a simple top down view shooting game, based upon an old favourite, Asteroids. I want the player to control the character object using the keyboard, and shoot using mouse clicks with the cursor as the target direction. Each enemy defeated will add to the high score which will be saved to file when the player has lost all the characters ‘lives’. The top high scores will then be viewable from an option in the main menu. These features will allow me to make use of various classes, data structures and calculations.

**Aims and Requirements of the Project**

* The user will be able to move the player object through the game world:
* The player object will be able to move around the game world through the keyboard input commands by the user in real time. Using the SFML library to check event during a loop the game can manoeuvre the object using positional changes, based on the key being pressed. For this purpose the arrow keys can be used, with each arrow corresponding to the same direction the player object takes within the game.
* The user will be able to fire projectiles from the player character at enemies to remove them:
* The user will be able, using the left mouse button, to ‘shoot’ projectiles at enemies on screen, in a 360-degree area around the player object. The calculation for the projectiles direction will be based on where the mouse pointer is in relation to the player object when the left mouse button is pressed. This allows for some more complex calculation to be used within the program. These projectiles will be able to collide with the map terrain and with enemies, removing the foe and projectile from the game in the process.
* The user will be able to navigate from the main menu into a new game, or high score screen, and back to the main menu from both:
* The main menu will be the first screen of the game, allowing the user to start a new game, or view the high scores. The user will be able to exit from both screens back into the main menu using keyboard commands.
* The user will be able to save high scores.
* When the player object has lost all it’s lives, the game will be over and exit to the game over screen, on which the high score will be displayed, and the user prompted to enter their name, to save their score. The score and name will be saved to a text file.
* The user will be able to view high scores.
* The user will be able to view a rank of previous high scores, in descending order, on the high score screen option from within the main menu. These scores will be taken from the high score text file on which the saves are made, sorted into descending order and outputted to the screen.
* The user will be able to exit the game smoothly on command.
* Through the use of the escape key the user will be able to quit the game gracefully from each window and then the main menu.
* The user will be able to restart the game as many times as they like while within the program.
* When the user ends their current game, and exits to the main menu, they will be able to start a new game right away, with everything being reset correctly.
* The game will utilise collision.
* Using bounding box collision and map tile set, the map, player, enemies and projectiles, will all have collision detection. This makes the user choose how to navigate the map, how the enemies themselves must navigate the map, how far the bullets can travel and when each object intersects with another. This will allow for some more complex calculations to be used.

* The game will keep going until the player loses all lives.
* The game will keep going, for as long as it takes the player object to lose all its lives.
* The game will make use of timers to control movement rate and spawn rate of enemies.
* Timers will be used to make the game frame rate independent, to decide when to spawn more enemies, to decided how fast the player can shoot, and to control the damage the player receives from enemies.

These functions represent the main requirements of the project and what I want it to achieve.

Exit Program

Save Score or Exit

Game Over, player has 0 lives remaining.

Game Loop

High scores and names placed into struct objects. Objects sorted based on score. Scores displayed to screen

High Scores Text File

Main loop initiated.

User input checked.

Player, Enemy and projectiles, updated.

Collision checked.

Player lives checked.

All items drawn to screen.

Program Entry Point

Main Menu

Variables initialised, player, enemies, map and projectiles initialised. Map loaded. Collision tiles loaded.

**Design of the Project**

Flow chart overview of the program:

High Scores

The following class diagrams represent the way the classes will work together in the program and how they function. It also includes the text files and image files to show how it will be handled.

Screen Class

(virtual int, run screen) all screens in a vector, main program pushes screens to stack, return values in screens determine next screen in use

Map Collision Layer

Text File

Map Base Layer

Text File

Tile Set

PNG file

Map Struct

(Used in all screens to generate background and in game loop for collision)

Bullet Sprite Sheet

PNG file

Enemy Sprite Sheet

PNG file

Player Sprite Sheet

PNG file

High Score Struct

(contains name + score)

Player Class

(Used in game loop)

Bullet Class

(Used in game loop)

Enemy Class

(Used in game loop)

Screen One

Game Loop

(inherits from Screen Class)

Screen Two

High Scores

(inherits from Screen Class)

Screen Three

Game Over

(inherits from Screen Class)

Screen Zero

Main Menu

(inherits from Screen Class)

High Scores Text File

**Testing the Program**

For the purposes of testing my program, the most appropriate kind would be a form of unit testing. Unit testing uses very small samples of code, normally just functions, but a unit can be defined as something as large a whole module, and then tests its performance individually outside of the main program.

For a small project such as mine, unit testing allows me to go through each function as I create it, and test how it performs to make sure it’s working as intended. Once you know it works how you want it to, you can be confident in using it as part of the main program, as you can show it does what you need it to.

This testing method allows me to catch any problems early and deal with them. However, this will not catch every error in a program, as it cannot evaluate every single execution path taken by the main program. Thus, it is best to test again with larger components of the program.

The way my program is set up, I can test individual screens to make sure they work as intended as well, before putting them as part of the main program. This moves the unit up from individual functions to multi-functional pieces of code and lets me test changes to each one individually, without impacting on the other areas of the program.

**Critique of the Program**

After spending much time running and updating the program, I can say I am at a state where I am happy for it to be handed in. It performs almost all the requirements I set for it, with a few exceptions. I did want the enemy AI to be able to traverse the map like the player, by having to avoid collision to get to them. I struggled for some time trying to implement it but couldn’t quite get it to work. I have since discovered an algorithm which would help me achieve what I want, however I believe time will run out before I can implement it. The algorithm is known as A\* (A star) and is a way for calculating the quickest path to an object from one point to another. Another part I would have wanted to spend more time over is the collision detection. Instead of just bounding box, I would have liked to implement pixel to pixel collision, which would only calculate when the bounding box collision is triggered, to be more efficient with calculations, and provide a better overall feel for the game.

A large amount of my time was spent on drawing the sprites myself, which let me get a feel for how much time goes into single person development of a game, and this was just a small project. Of course, you could always outsource this to an artist or find plenty of free to use sprites online, however, it just felt so much better to see something I had made moving on the screen.

The project helped me to understand how useful it is to keep everything separated into its own distinct header and cpp files. It makes navigating the code extremely easy and allows you to instantly know where a piece of code you wrote for something is, simply because it is grouped with similar items. It also presented a problem I hadn’t encountered before, using forward declarations to help break cyclic definitions.

Overall though, I have thoroughly enjoyed this project. I have learnt many new things about using C++ effectively, for example, vectors and references, along with destructors. The biggest surprise for me was finding out about the ability to have running the screen as a class, and not only that but it could be used to also help pause the game for menus. If I had known about this from the start I would have planned the game around it, as I think it’s such a great concept to add. The variables for all moves would be stored by the class screen itself, rather than the other classes, so they wouldn’t be destroyed when they went out of scope, as the screen is constantly in use in the background, simply pushed back onto the stack. This way all data is retained rather than removed as it is now, so it’s something I plan to work with in the future and hopefully find other interesting ideas from other sources throughout the summer as I continue to learn more about C++.