**Foundations of Computing & Technology – Programming**

**George Hand – N1050297**

**Programming Project – Space Invaders**

**Project Aims**

The aim of the program is to create a fully working Space Invaders style game using Pygame, which is a cross-platform set of Python modules designed for writing video games.

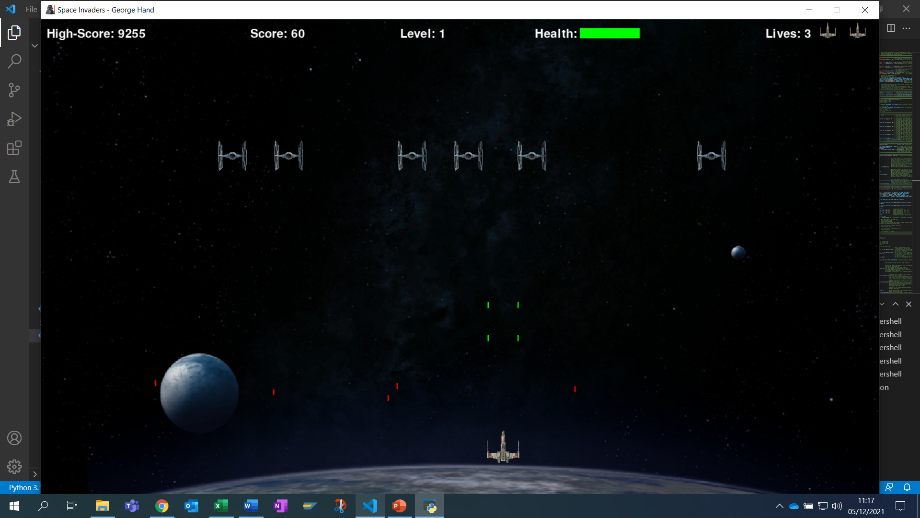
It is intended that the program will have a “Start Screen” displaying the name of the game and giving the user instruction on how to play and start the game.

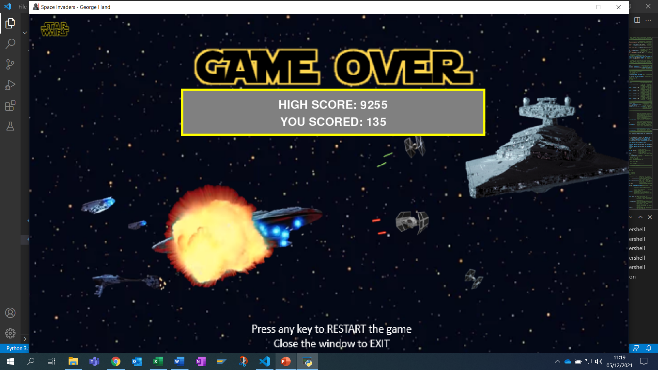
In the game, the player will control a spaceship that can move across the bottom of the screen and fire missiles at waves of enemy spaceships as they fire missiles at the player whilst, simultaneously, moving across and down the screen closer to the player. The player will gain points for destroying enemy spaceships but will lose health and lives if they are hit by either an enemy missile or spaceship.

If the player destroys a full wave of enemy spaceships, then a new level will start and the difficulty will increase by speeding up the enemy spaceships and allowing more missiles to be fired. The player will receive increased points per enemy destroyed as the levels increase. All scores, lives and health data will be displayed on the game window. Sound effects will also be added to improve user experience.

Once the player loses all their lives, a “Game Over” screen will display summary score data and give the user the option to either restart or quit the game. The game will also include the ability to save and restore a high score, which will involve file i/o to an external text file.







**Project Analysis & Design**

When analysing the project, it will be split into four major parts.

* Game Design – not program design
* Pygame
* Programming
* Testing and Validation

**Game Design – not program design**

Before considering the Python and Pygame coding aspects of the project, I firstly needed to think about what the I wanted the actual game to look like and what the main functionality should be. A lot of this has been covered in the previous project aims section. But it was also important to sit down and think in more detail about how the Start, Main Game and Game Over screens may look like. This involved sketching screen layouts and getting background images and game sprite images and sound files ready for use.

**Pygame**

The challenge was to understand and teach myself the basics of the Pygame module. The first requirement of this was to download and install Pygame and ensure it was fully working on my laptop. Once this was done, I consulted online materials and tutorials to gain a basic knowledge of Pygame’s capabilities, base functions and how to approach game programming.

**Programming**

Once I had a clear idea of what the I wanted the game to look and play like, and I’d understood the basic capability of Pygame, I could begin coding the game. This again, was be broken down into smaller very specific tasks and requirements.

1. **Definitions**

The first section at the top of the program is where I will define all modules and functions that need to be imported to be used in the functions of the program.

For example.

Pygame – to use the game functions

os – which supports file paths/dir which will be supported across platforms

load image and sound files

1. **Programming the game window or Surface as Pygame refers to it.**

Here I set up the size of the window, add a title and icon.

Run the first basic program to ensure the game window is displayed as intended.

1. **Main gain loop**

This is the ***main()*** function and heart of the game.

At the start of the main ***while loop***, I check set the speed of the game in FPS

(frames per second) so it will run the same speed on any computer irrespective of its

capability. Then I monitor if the user decides to QUIT the game. Then the main function

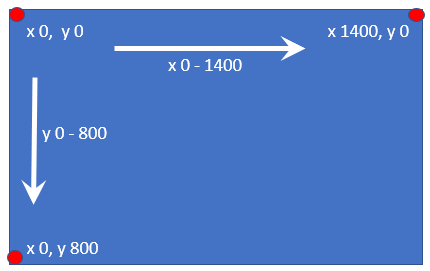
runs the game loop calling all the separate functions and passing the necessary

parameters to execute the game.

1. **Creating a specific draw function to update the display**

As the name suggests the game calls this function each time it needs to update the display screen. The program sets up all the information it wants to display but needs to run specific commands to render this information onto the screen.

*Note: Pygame screen setup starts with 0, 0 in the top left of the screen, as shown below for my 1400, 800 game window setup.*

**

1. **Learning how to get the player images and display them on the screen**

After setting up the basic “shell” of the program in the previous steps, the first game programming task was to understand how to get the basic game characters or sprites displayed on the screen and scaled to an appropriate size. For this I used the player image and only one enemy spaceship, as duplicating enemy ships would be done later.

1. **Move the sprites scaled images, and ensure they could not be moved off the screen**

Once on the screen the Player Spaceship needed to be capable of moving and being controller by the user. For this I use the left and right arrow keys. It is also important that the Player cannot be moved off the screen. The enemy sprites were programmed to move following a set path across and down the screen as per the original Space Invaders.

In all cases Player and Enemy sprites are represented with rectangles, this allows them to be moved using their rectangle co-ordinates.

1. **How to create missiles to be fired by both player and enemy**

Basic rectangles will be used for both sets of missiles, the Player rectangles will be green and the enemy red.

Player missiles will be fired when the user presses the ***Spacebar***, this will generate two green rectangles to be displayed on screen at the position of the player’s guns.

These two missiles (rectangle coordinates) will then be loaded into a ***LIST*** to enable tracking and movement of multiple missiles.

Enemy missiles will be generated randomly but handled in the same way. Missile movement is achieved by increasing/decreasing the ***“y”*** position of each missile in the player and enemy lists.

In both cases the program controls the maximum number of active missiles. This prevents the player continuously firing and allowing the difficulty level to be controlled due to the number of enemy missiles.

1. **Detect and deal with collisions**

Before dealing with missile collisions, the program first needs to be able to remove any

missile that has gone off the screen without hitting any other object.

Then, once at the stage where all the sprites can move and fire missiles, the program needs

to detect and deal with any collision with a missile and between the player and an enemy

spaceship. This is done using the Pygame ***colliderect*** function which detects any collision

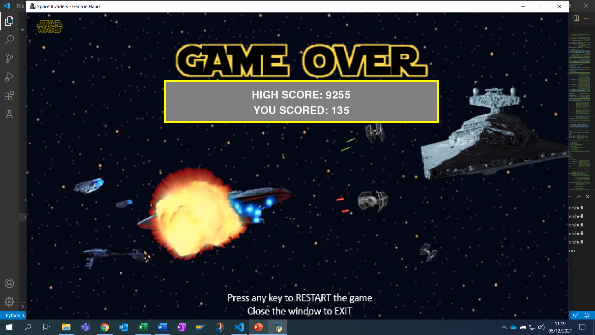
between rectangles. Once a collision is detected we then trigger events such as remove the

missile, remove the enemy, and increment the player score etc.

This completes the basic game programming tasks and requirements.

1. **Start & Game over screens**

Without a Start Screen or Game Over screen the program will just start the game without the user knowing what to do and close once the player has lost all his lives. Added these two screens/functions makes the game look more professional. The Start Screen will display the game name and user play instructions and wait for the user to decide when they are ready to play the game. The Game Over screen will be called once the Player has lost all their lives, display their score and ask the user to either QUIT the game or restart.

1. **Add basic score, health, and lives**

This is the start of the final touches to the game, where the status items are displayed on the screen. Here we will need to select an appropriate font and select where on the screen the information will be displayed.

1. **Additional features levels with increasing difficulty, high score, sound effects**

The difficulty level of the game is increased each time to player destroys a wave of enemy spaceships. The next level will move faster and fire more missiles.

High Score data involves saving and retrieving data from an external file (file I/O)

Otherwise, every time to game program is closed the information will be lost.

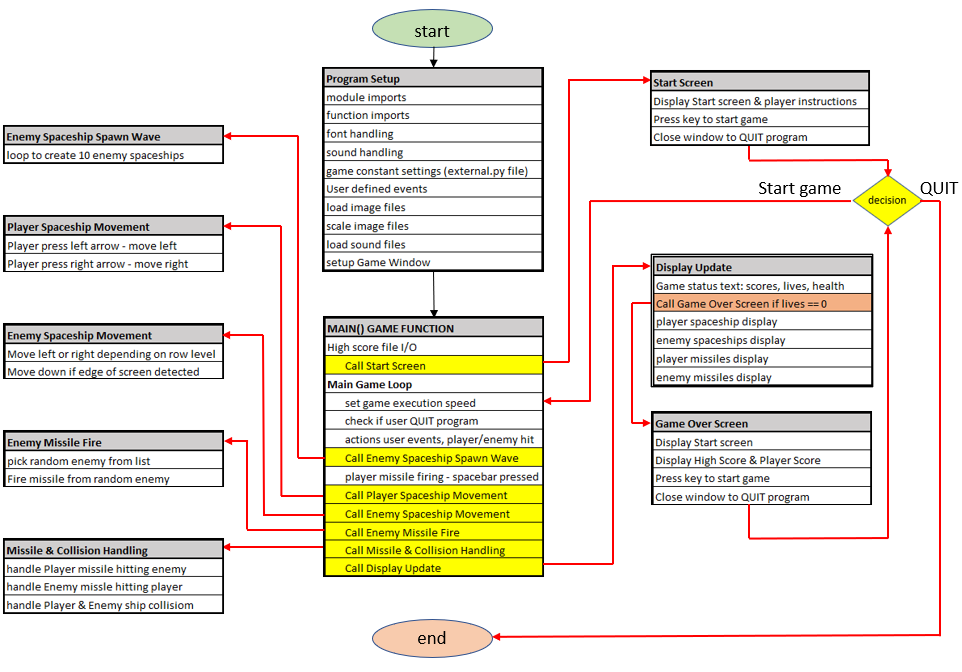
Sound effects are also included to enhance the game.

**Testing and Validation**

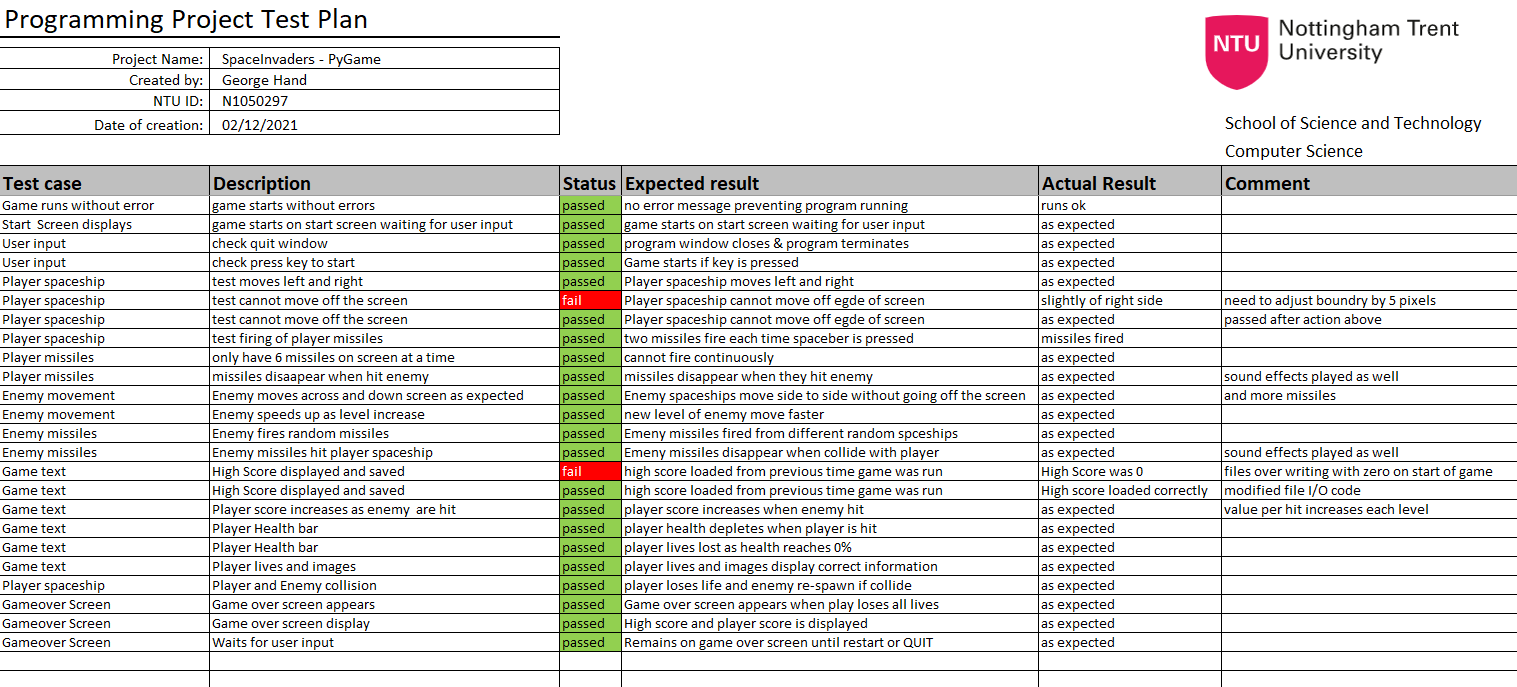
During the programming development, every small change and addition is tested to verify it functions as intended. Some sections of code are tested in smaller stand-alone Python files before being added to the main game file, such as the High Score file I/O code to ensure they don’t introduce unexpected errors in the main game file.

After the game was programmed and thought to be fully working as per design specification. It is important to have a detailed test plan to validate all functions are operating as expected. A full Program Testing Plan can be found in the Program Test section of this document.

**Project Design Flowchart**



**Program Testing**

****

**Project Critique**

Overall, I feel the project was a great success, managing to achieve all the requirement set out in the project specification and design brief.

What worked:

Taking time to properly design and plan the work instead of jumping in and starting to code.

Having a clear step by step approach allowed me to move from one part of the game programming to the next and test most of the functionality before moving onto the next item.

I also used small test.py program files to test individual pieces of code and get it working before adding it to the main program file.

This ensured I did not have any major disasters by adding new code, encountering problems, then this affecting other parts of the game.

Another major success was making sure I organised and commented my code as I developed the game. This became increasingly important as the size of the program increased. The fact that all the code was commented made it much easier to keep control of.

Items that can be improved with additional experience and knowledge:

The Collision detection method I used was the most basic method available of comparing if the areas of two rectangles overlap. Because the images of the sprites aren’t perfect rectangles it often looks like the sprites get hit even though the on the display the missiles don’t contact the sprite image. I know there are more complex methods of overcoming this problem by using a “mask” of the image at pixel level, but this was not known at the start.

The programming code’s efficiency could be vastly improved using more advanced structures such as “Classes”, these would group together an objects attributes better and bundle them into a single object. I presume these would then simply the list of parameters passed between functions.