OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name : <Armaandeep Nagra>

Candidate Number : <8215>

<Institution Name> : <25136>

Title of Project : <INSERT PROJECT TITLE>

H446-03 – Project CONTENTS

Table of Contents

[A. Analysis 5](#_Toc141881568)

[Description Of The Problem 5](#_Toc141881569)

[Stakeholders 5](#_Toc141881570)

[Identification Of My Target Platform 5](#_Toc141881571)

[Impact Of Platform And Target Audience 6](#_Toc141881572)

[PC Platform 6](#_Toc141881573)

[Target Audience 6](#_Toc141881574)

[Research 6](#_Toc141881575)

[Operation Wolf (1987) 6](#_Toc141881576)

[Interface And Graphics 7](#_Toc141881577)

[Control System 9](#_Toc141881578)

[Level/Game Progression 10](#_Toc141881579)

[Game Mechanics/Rules 11](#_Toc141881580)

[Conclusion 12](#_Toc141881581)

[Survey Analysis 13](#_Toc141881582)

[Features Of The Proposed Solution 17](#_Toc141881583)

[Interface 17](#_Toc141881584)

[Gameplay 17](#_Toc141881585)

[Hardware & Software 18](#_Toc141881586)

[Hardware & Software I am using for development 18](#_Toc141881587)

[Hardware & Software Requirements For Deployment 19](#_Toc141881588)

[Computational Methods 19](#_Toc141881589)

[Thinking Ahead and Concurrently 19](#_Toc141881590)

[Abstraction 20](#_Toc141881591)

[Decomposition 20](#_Toc141881592)

[Limitations 20](#_Toc141881593)

[Realistic limits for the proposed game 21](#_Toc141881594)

[Success Criteria 21](#_Toc141881595)

[Must 21](#_Toc141881596)

[Should 22](#_Toc141881597)

[Could 23](#_Toc141881598)

[B. Design 23](#_Toc141881599)

[Systems diagram 23](#_Toc141881600)

[Structure Explanation 24](#_Toc141881601)

[The Main Stages 24](#_Toc141881602)

[Screen Design 25](#_Toc141881603)

[Welcome Screen 25](#_Toc141881604)

[Main Menu 27](#_Toc141881605)

[Level 29](#_Toc141881606)

[Win Screen 30](#_Toc141881607)

[Loss Screen 31](#_Toc141881608)

[Development Plan 31](#_Toc141881609)

[Object Oriented Approach and Class Diagrams 32](#_Toc141881610)

[Benefits of the Object Oriented Approach 32](#_Toc141881611)

[Disadvantages of the Object Oriented Approach 33](#_Toc141881612)

[Classes in my Game 33](#_Toc141881613)

[C. Developing the coded solution (“The development story”) 51](#_Toc141881614)

[D. Evaluation 71](#_Toc141881615)

[Project Appendixes 74](#_Toc141881616)

# A. Analysis

## Description Of The Problem

I have decided to create a shooting game which takes inspiration from the retro game “Operation Wolf” (1987), Operation Wolf is an arcade game viewed from first-person perspective with the screen scrolling horizontally through the landscape. The player must save 5 hostages who are being held captive by enemies, to do this the player is given a physical replica uzi gun at the arcade machine and he must shoot the attackers.

In my version of the game, the player will be given various weapons such as a sniper or rifle. The player must save hostages and kill the enemies, the game is viewed from first person perspective with the screen automatically scrolling. If all enemies are killed and no hostages are shot, then the player has completed the level. What makes the game difficult is that the enemies will shoot back at you which will decrease your health and because of the element of randomness in the game, it may be hard for the player to differentiate between an enemy and a hostage.

I will need a health bar so that health is deduced when enemies shoot at the player or to add health when the player kills the enemy. I will also add an ammo count so that the player has to reload and cannot shoot continuously as that would give them an advantage. I will have a progress bar such as a percentage of how much of the level the player has completed. As soon as the player kills the hostage the game is over. The user will use their mouse to move the scope and to shoot.

## Stakeholders

I have identified my stakeholders to be gamers who like retro style shooting games, they will usually be between 12 and 18 year old, however, due to Operation Wolf being a retro game from 1987 there may be some older gamers who may be interested in my game due to nostalgia.

Although my game does not contain profanity, it does contain violence as the player has to shoot the enemies and gets shot himself. Therefore, if I was to look at the PEGI ratings, I would classify my game as PEGI-12, this means anyone over the age of 12 should be able to play this game without it affecting them negatively. However, due to having a high PEGI rating that means my market is smaller.

Stakeholders may find it hard to adapt to the different commands and interface of the game, so it is vital that I add standardization throughout the project, e.g. A bar out of 100 with a heart on the side, a scope to shoot with etc... This will make it easy for players to get used to my game.

## Identification Of My Target Platform

My game will be designed to be run on a PC and it will be created using Python and the Pygame library. As a result, I could adapt the game to be played on other platforms that have a python interpreter and the correct libraries installed.

As the game is being developed for a windows PC, I am going to assume the following:

* The user has access to a keyboard and mouse
* The user will have access to a suitable monitor or screen that will be able to output the game
* There will be a way for the sound to be outputted to the player, computer may have built-in speakers or external speakers.
* The computer is less than 10 years old and has sufficient CPU power and memory to run the game.

Because I am only developing the game for a PC, I won’t be required to implement:

* The use of motion controls
* The use of touch screens
* The use of microphones or webcams

The target platform will be important for the development process of the game, as I design the interface and functionality of the game.

## Impact Of Platform And Target Audience

As I am creating a PC shooter game, these decisions will have a major impact on the features and design of the game.

### PC Platform

My game is meant for a Windows PC. Therefore, it won’t be suitable for the player to play the game on the go e.g. on the bus or when outside. Therefore, the player will probably play for long periods of time which means I should make the game engaging as it is inaccessible for mobile users.

### Target Audience

My target Audience is players above the age of 12, so they would usually have prior gaming experience and be familiar with the standardised features. Adding different levels and difficulties would be suitable for my target audience as it means the players are more engaged.

## Research

### Operation Wolf (1987)

The game Operation Wolf is a first-person perspective 2D light gun shooter game that automatically scrolls through the horizontal plane. It was developed by the Japanese company “Taito” in 1987. It was originally developed to be an arcade game and it became one of the highest-grossing arcade games of 1998. However, it was quickly developed for 8-bit computers.

### Interface And Graphics

In operation wolf, a new player can be a little daunted by the HUD because the right-hand side contains a lot of information, however it does not take long to get used to. This is what the right-hand side of the screen means:A picture containing text, screenshot, pc game, cartoon

Description automatically generated

Below is an explanation of the different sections of the screen:

A picture containing text, screenshot, pc game, cartoon

Description automatically generated

Here you have the score, this increases as you destroy tanks and helicopters, when you kill enemies, when you rescue the prisoners and when you complete a mission/level.

This contains information about your weapon, the top shows how many magazines you have, and the bullets show how many bullets are in the current magazine that you are using. In this case there are 6 magazines and 30 bullets in the current magazines, which means that in total there are 240 bullets. You also have access to grenades, in this case there are 5 grenades you can use.

A picture containing text, screenshot, pc game, cartoon

Description automatically generated

A picture containing text, screenshot, pc game, cartoon

Description automatically generated

To complete the mission, you must eliminate the enemies, in this photo the player must eliminate 50 enemies, 5 tanks and 4 helicopters, as the player kills/eliminates these, the counter will decrease.

This box is not used until later in the game, once you reach the concentration camp mission (if you defend the hostages), the hostage number increases to 5. Then, the hostage number decreases to 0 one by one at the airport when you evacuate each hostage.

A picture containing text, screenshot, pc game, cartoon

Description automatically generated

This thin section is the player’s health bar, the blue shows your health and whenever you get injured or when you shoot an innocent person you lose health and it shown because the blue bar decreases, and the red bars increase.

A picture containing text, screenshot, pc game, cartoon

Description automatically generated

This is the reticle used in the operation wolf dos emulator, in the original arcade game there was no reticle as you had to use a physical gun to aim and shoot, however with emulators you can aim with your mouse.

A picture containing text, screenshot, pc game, cartoon

Description automatically generated

As the game is first shooter and the screen automatically scrolls, you cannot see your own character.

### Control System

The controls for Operation Wolf are simple, a physical uzi gun is used to aim and shoot, and it looks like this:



You have a front and a back sight, and you must align them together to aim and shoot. There is no reticle on the screen of the arcade machine, however there is a reticle on the emulator version of operation wolf because you have to aim with a mouse.

There are two red buttons on the uzi gun, the red circle on the bottom front of the uzi gun is to start the game however you can also start the game by pressing the trigger or firing. The other red button on the left side of the uzi gun is to switch between mortar rockets and bullets.

To make the game realistic there is an electric motor and piston mechanism inside the gun, when the trigger is pulled it activates this special mechanism which sends the piston back and forth and this creates a vibration effect which mimics real uzi guns. It looks like this:

A picture containing auto part, car, automotive air manifold, automotive fuel system

Description automatically generated

### Level/Game Progression

A screenshot of a video game

Description automatically generated

This screen shows all 6 different missions that the player has to complete in order to finish the game. All players have to start at communication setup and then the arrows above show the order in which different missions have to be completed.

To complete each mission you have to eliminate the amount of soldiers, armoured cars, helicopters or gun boats specified, e.g. you must eliminate 40 soldiers, 4 helicopters and 5 armoured cars in the communication setup mission.

Of course, as you complete a new mission the game gets progressively harder, for example, hostages will be mixed in with the enemies in the later levels so you have to be careful not to kill the hostages and also to kill a soldier with a knife that chases the hostages. During the last missions there will be “bosses” that appear during the mission and they have certain weaknesses you need to exploit in order to kill them.

### Game Mechanics/Rules

The main rules of the game are quite simple, the player must complete all 6 missions successfully in order to save the hostages and finish the game. To complete each mission, the following conditions need to be met:

* The player should have eliminated all of the soldiers, armoured cars, helicopters or gun boats that can be seen on the left side of the HUD during a mission.
* The player must have at least one bar of health
* Rescue at least one prisoner/hostage

There are some items in the game which the user can shoot to gain an advantage, examples include:

A picture containing text, sketch, drawing, design

Description automatically generated

A picture containing text, screenshot, font, sketch

Description automatically generated

The maximum number of rocket grenades and bullet magazines a player can have during the game is nine.

### Conclusion

**Things I like about the game:**

* I like the simplicity of the controls; with modern console or PC games, it can be hard sometimes for new gamers who are not used to standardised controls to understand which buttons to input to get their desired output. However, with the arcade version of operation wolf, an uzi gun is used and this is something that even non-gamers know how to operate because they may have seen it in movies or on tv shows.
* I like the attention to detail that the developers used, for example when the enemies throw a grenade or a knife you can shoot it whilst it is still in the air so that it does not cause harm to you, also, when you shoot the enemies on different parts of their body they fall differently, e.g. when you shoot a soldier in the head they fall backwards but when you shoot them in the legs they fall towards you.
* I like the sound effects the game has to offer, for example the main menu music and the sound effects when you shoot and when you eliminate enemies.

**Thinks I dislike about the game:**

* I dislike the HUD on the right-hand side of the screen when you are in a mission, as that blocks a lot of room and does not look good, instead the sections from it should be blended it in with the gameplay and should be at the bottom or top of the screen.
* Operation wolf is known to be a difficult game, so this is something I don’t like as it requires expertise and a lot of practice just to complete the game.

## Survey Analysis

A screenshot of a computer

Description automatically generated

From the stakeholders who I surveyed, who were all between 12 and 18 years old, the majority only play games for 0-5 hours a week, this means I need to make the length in which my game finishes quite short as most people won’t play for too long.

A screenshot of a computer

Description automatically generated

The majority of my stakeholders say that they prefer games which have a final objective, Operation Wolf has a final objective which is to rescue the hostages, therefore, I will try to implement a solution which requires the player to finish the game with an objective being accomplished.

A screenshot of a graph

Description automatically generated

The main reason my stakeholders play video games is for entertainment, therefore I have to create a game which keeps players entertained, again, Operation Wolf is a great example of that due to the excitement building up as you get closer to rescuing the hostages. Other factors such as socialising, improving gaming skills and stimulating focus and creativity weren’t as important so I don’t need to make the game multiplayer and I should make it so that finishing the game is possible after practising for a few hours.

A screenshot of a chat

Description automatically generated

Almost every single stakeholder I surveyed prefers games which can be paused midway, as Operation Wolf is a game which cannot be simply paused, I will make sure to add a way in which a player can pause the game just in case they need to do something else.

A screenshot of a computer

Description automatically generated

A screenshot of a graph

Description automatically generated

Most of my stakeholders agreed that they want good sound effects e.g. when they shoot at the enemies or when they pick up an item etc… Although Operation Wolf is known for its background music the stakeholders getting surveyed said the background music is not as important so I will try to focus more on getting the sound effects right.

A screenshot of a chat

Description automatically generated

The startup menu for Operation Wolf is very graphical and looks really good, however, after surveying my stakeholders I realised that the startup menu is not a very important aspect of the game development, therefore, I will be spending more time on other aspects of the game.

A screenshot of a computer screen

Description automatically generated

The stakeholders I surveyed would clearly like for the game to have good graphics, therefore I will be spending more time on making sure the sprites and background looks realistic as that is something that Operation Wolf could improve on due to it being quite an old game.

## Features Of The Proposed Solution

### Interface

Display –

* Fullscreen Display.
* Side on view.
* Dynamic Screen - Screen automatically scrolls from left to right.
* Menu on the side of the screen to inform player with statistics.

Controls –

* Mouse for aiming.
* Mouse right click for grenade, left click for shooting rounds.
* Keyboard: space bar for starting game or pausing. Esc to exit.

Colour and style:

* Simple military-style level map.
* Simple enemy sprites, helicopter, armoured cars and more.
* I won’t use bright colours such as bright pink etc… Instead I will try to use the original colours used in operation wolf such as different shades of: green, brown, grey, blue and more.

### Gameplay

Start –

* When you run the game, you are greeted by a screen asking you to enter name, after entering name you enter the main menu where you can start the game or view the leaderboard and your personal score.

Progress –

* Player has to go through one level to finish game.
* Level ends when player dies, when he runs out of ammo, when he kills a civilian or when he reaches the end.
* As the level progresses the player can shoot at different power-ups or tools that can support him.

Scoring –

* Player gets score based upon the time in which he completes the game
* The overall score is then deducted according to the amount of ammunition and health left.
* Top scores will be saved onto a leaderboard/scoreboard.

Enemies –

* Difficulty will increase as player progresses throughout the level.
* There will be both human and vehicles shooting at player.

End Game –

* Game ends once the level is completed successfully.

Data Requirements –

* Scoreboard will be saved in an external file.
* Music and sound effects will also be saved in an external file.
* Map may also be stored in an external file.

## Hardware & Software

### Hardware & Software I am using for development

These are not the minimum specifications needed to develop this program, however, they are what I will be using as that is what is accessible to me.

#### Hardware Used for Development

I will be using a desktop PC with the following spec:

* CPU: AMD Ryzen 5 3600 6-core processor
* GPU: Nvidia GeForce RTX 2060
* Memory: 16GB
* Storage: 1TB SSD

The storage will be more than enough as I will probably only need to store 1 or 2 GB of stuff such as images, sound, text and source code.

I have a GPU, however, pygame doesn’t automatically make use of this, I can configure it to make use of it, however if I do that, I may find it difficult to develop a game which someone without a GPU can play without lagging. Because I have a higher amount of RAM and higher clock speed than is needed to play the game, I will have to make sure to keep my game simple to make sure that anyone with the specs I outline below can play.

#### Software Used for Development

* Operating system: Windows 11
* Additional Software: Python IDLE

### Hardware & Software Requirements For Deployment

#### Hardware requirements for deployment

* Monitor to display the game.
* Mouse to aim and shoot.
* Speakers/headphones to hear the music and sound effects.
* Keyboard to start the game and to enter your name into the leader board.
* Processor: Intel Pentium or above (64-bit)
* Memory: 2GB or better.
* Graphics: Integrated Graphics.
* Network: This game does not require an internet connection to be played however it is needed to download python and the used libraries.
* Storage: 1 GB available space.

Explanation and justification – The requirements above are typical for an entry level computer, low amounts of ram and an entry level processor are needed to run the game. If the processor can run the operating system it should be able to run my game. External graphics cards won’t be needed as my game will not require high graphical processing power.

#### Software Requirements For Deployment

* Operating System: Windows, MacOS, Linux.
* Additional Software: Python and libraries such as pygame are required.

Explanation and justification – The player needs a “standard” Operating systems such as Windows 7 or later, MacOS X 10.7 or later or a suitable Linux system this is because Python and pygame are only available for these. Because Python is an interpreted language, the python interpreter and additional libraries such as Pygame also need to be installed as I will need to use them during development and for them to interpret the game when the user is playing it.

## Computational Methods

As I am completing my project I will need to use a range of computational thinking methods, that will help me plan and complete my project. The following techniques will be discussed and I will explain how I can demonstrate them.

### Thinking Ahead and Concurrently

Thinking ahead is the process of thorough planning of your code to ensure that an efficient program is produced. When we approach a problem such as coding a game, it is important for the developer to plan the steps they are going to take because if they don’t, then they may be wrongly solving a problem, which will be a waste of time and resources.

I will aim to think ahead by identifying a plan to create the different stages of my game, I will also use a modular approach to solving my problem, this is so that it is easier to make changes to my program during development if I change my mind or if I get new feedback. This is a great technique as I could create classes for certain sections of my solution, and if I need to change a certain class it won’t affect the overall problem.

### Abstraction

Abstraction is the removal of unnecessary detail from a problem to make solving it easier. It may be used to make a game playable on low-spec computers as you could remove unnecessary graphics or high-resolution images.

During this project I will be using abstraction to simplify and remove unnecessary details from my game, this will make my game playable by users who play on low-spec devices. Abstraction will also be used to make the game easier for me to develop as I am just a student doing his A-level computer science project, so it will make my project more likely to be successful.

For example, I won’t be using all of my resources to perfect the way the enemy falls when the player shoots them as that is an unnoticeable detail, which does not require much consideration for this project.

### Decomposition

Decomposition is breaking a problem down so that it becomes more manageable and easier to solve. A solution can be found for each subfield of that problem which can make it easier to “connect” each subfield together to produce a solution to the problem as a whole.

In my project, using decomposition would be a vital process because it would allow me to focus on working on one part of the game at a time, this means the game can be broken down in terms of time stamps so that I know which part I should be working on according to the day of the week.

For example because I will be using Pygame, I can make use of modules so that I can develop one section of the game such as the sprite class and then import it later to the main file.

## Limitations

### Realistic limits for the proposed game

Because this project is being completed as part of my A-level studies, I have to be realistic in what I can achieve in the timeframe I am working with.

#### My limitations

* I have a limited time to develop this game as I need to meet deadlines, and also have to focus on other subjects in school.
* Because I have never completed a project at this scale I have little experience on how to manage it.
* I have limited programming skills, therefore, it will take me longer to understand concepts and find bugs in the code.
* I am coding the game all by myself, if this was to be done in a team, more complex projects could be undertaken in this timeframe.

#### Technical Limitations

* **Graphics:** I won’t be spending too much time on creating the graphics in my game, firstly, creating high quality graphics will strip away a lot of my time and I want to use that time to actual create a game which can be played smoothly, secondly, I do not possess the expertise to create such graphics and I can’t learn that within this timeframe, thirdly, pygame requires simple 2D graphics.
* **Scale:** I won’t be creating a game with many levels, although it will have a final objective, it will be more of a prototype so that the stakeholders can know what to expect for the final game.
* **Features:** my game will only contain one level and I may not add additional features such as recoil if they’re too challenging.
* **Connectivity:** unfortunately, the whole idea of the game stops it from being multiplayer, however I can add a leaderboard so that people can compete with one another based on their score.

## Success Criteria

It is essential to have a clear list of SMART criteria by which I can evaluate my project. My success criteria will consist of must, should and could features. Most of these features will be standard convention for the game however some features will be added due to feedback from my survey and further research.

### Must

|  |  |  |
| --- | --- | --- |
| No | Criteria | Justification |
| 1 | Player must be able to use mouse to use weapon | This is the primary element in my game, without shooting the player cannot win the game. |
| 2 | Player must be able to use keyboard to start the game or pause game (if I include the option to pause the game) | This is a primary element in my game as the player would be playing on keyboard and mouse, therefore to start the game or to pause the game, the player will need keyboard inputs |
| 3 | Game must include a level in which the player has to kill enemies. Game must also get harder as level goes on. | This is a key part of the game as it allows player to receive entertainment from playing. |
| 4 | Player must be able to see:  -score  -health  -ammo  -gun type  -Number of enemies and vehicles left  Whilst in-game | This is a fundamental part of the game as without seeing this information, the player cannot make suitable decisions. |
| 5 | Enemies must show some intelligent way of attacking | There must be some type of human-like behaviour in the enemies so that their reaction times and attacking tactic is humane |
| 6 | Game must load and run without crashing or errors | This is the most important part, if the game does not run properly then the player cannot play |
| 7 | There must be a simple start-up menu, main menu, winning screen, game over screen, leaderboard and pause menu | This is important as it allows to keep track of player scores, play the actual game, provide feedback to the player and to let player know how his score compares to the other scores. |
| 8 | The game must include sound effects and background music | This was a request by my stakeholders and I think it is a fundamental because it informs the player about what is going on. |
| 9 | The screen must scroll in the horizontal plane | This is the core part of the game which gives it its “mission-like” feeling. |

### Should

|  |  |  |
| --- | --- | --- |
| No | Criteria | Justification |
| 1 | Game should have a pause option | By adding a pause option, it would make the game slightly better as the player can pause midway if they have to go somewhere |
| 2 | Game should have background music when in the menu | This should make the game more entertaining |

### Could

|  |  |  |
| --- | --- | --- |
| No | Criteria | Justification |
| 1 | The game could have a leaderboard of an individual’s score | This would be a nice touch to add to the game as the players can see how good their score is compared to others. |
| 2 | The enemies could all behave differently so that they don’t perform the same actions | This would make the game seem more professional |
| 3 | I could add the option for the player to customise his controls | This is to ensure that the player is comfortable when playing the game. Players may have played similar games in the past so they could use the controls they are used to |

# B. Design

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

## Systems diagram

## Structure Explanation

I have used decomposition to break down my problem into smaller, manageable sections. This will help a lot as I can work on each individual part and then connect each subsection together to complete the final game. This diagram also helps me to keep track of how I should manage this project in terms of deadlines and when I should have a part completed by.

### The Main Stages

There are 4 main parts of my game: The welcome screen, the main menu, the level itself and the end of the game.

#### The Welcome Screen

This part of the game will present the user with a simple graphical picture with the name of the game and background music, after the user presses the spacebar to continue, another screen will be loaded which contains instructions and controls to the game and prompts the user for their name, so that score can be stored in the leaderboard.

#### Main Menu

After pressing space on the welcome screen, the main menu will be loaded along with the background music, high score, name, leaderboard and the option to play the level. The user can then press a key on the keyboard to start the level or to exit the game.

#### Level

After the player has started the level, my program will load the map and loop to create a horizontal movement. After this the enemies/vehicles, the sound effects, crosshair and HUD loads. In this part the user can press the spacebar to pause the game and then press spacebar again to resume or ESC to exit. When the player shoots at something there will be a visual effect which will be displayed. Messages will also be displayed onto the screen if something such as ammo or health is low to warn player. Throughout this, I will have a score variable which can keep track of the score of the player. Most importantly I have to keep track of the enemies targets so that I can register whether they have been shot.

If the player does not finish the level due to dying or killing hostages, then a losing screen is loaded and the player has the option to be redirected to the level progression menu to play again or to exit the game.

#### Win Screen

If the player has finished the level, he is redirected to a page which congratulates him and outputs the player’s score, he is then prompted to either play again and go back to the main menu where his level is reset or to exit.

## Screen Design

Down below I will be explaining what each of the displays the player will see looks like, this is so that I can visualise how I should go about programming this. The colours used don’t mean anything as I’ll change them later on and I have used abstraction to simplify the drawing, so that it is not time consuming at this stage.

### Welcome Screen

#### Welcome Image

Digital Art, background

Title of Game

“PRESS SPACE TO CONTINUE”

The welcome image will contain some digital art as the background, it will have the title of the game and it will have a message which will inform the user they can press space to go to the next part.

#### Name Prompt

Instructions & Controls

Enter Name

SUBMIT

The name prompt menu will be a simple menu which contains a text box with the instructions on how to win and the controls for things such as shooting, pausing etc… There will also be a prompt for the user to enter their name so that their score can be added to the leaderboard, once they’ve entered their name, they will need to click submit to go onto the next part.

### Main Menu

Game Logo/title

Name: (name)

High score: (h\_score)

CLICK FOR LEADERBOARD

PLAY

Press to exit game.

This menu acts almost like a “middle-man” between the player and the level. The player will come here after inputting his name. He can see his name and high score, there is an option to click the leaderboard which will get updated with the top scores. There is a button to play the actual level and an instruction/button at the bottom right informing the player how to exit if they wish to.

#### Leaderboard

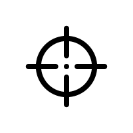
LEADERBOARD OF TOP 3 SCORES

1. Player1
2. Player2
3. Player3
4. Player4
5. Player5

Go back to main menu

This leaderboard can be accessed from the main menu and it will provide the player with a list of high scores, I will have to store the scores locally so that they can be retrieved back and overwritten if the player has already played but he receives a better score. I have a button at the bottom which redirects the player back to the main menu if he wished to go there.

### Level



Amount of ammo and grenades

Gun type

Number of enemies left to kill.

In the level, the background will be moving from one side to the other, creating a feeling like the player is moving, at the top we can see the health bar which will have bars in it to let the player know how much health he has, on the bottom right is the weapon section which tells the player which gun they are using and how much ammo they have left for that gun and how many grenades they have. On the bottom left is the number of enemies left to kill in order to finish the level. The green rectangles in the image and the car represent the enemies which the player has to eliminate. The player will be able to use their mouse to move the reticle/crosshair to shoot at the enemies.

#### Pause Screen

You’ve paused the game.

Press ESC to exit the game

Press Space to resume

### Win Screen

Congratulations (name),

Your score was: (score)

Press SPACE to play again and be taken to the main menu

Press ESC to exit the game

### Loss Screen

Game Over

Press ESC to exit the game

Press Space to play again and be taken to the main menu

The pause screen is loaded and displayed when the player presses the SPACE BAR whilst in the middle of the level, it will be listening for keyboard outputs which will be the SPACE bar and ESC.

The win screen will be outputted when the player successfully completes the level, it will contain a suitable message to congratulate the player and the score they achieved, it will be listening for keyboard outputs which will be the SPACE bar and ESC.

The player will be greeted by the loss screen if they don’t successfully complete the third level. It is the same as the win screen, however, a suitable message for this scenario will be displayed, and the player won’t get a score.

## Development Plan

During the development of my game I will try to develop the feature in this order, this is because it would make my project more manageable as I am able to follow a roadmap/plan.

1. Create welcome screen
2. Create screen which prompts the player for their name
3. Create main menu
4. **Prototype 1**
5. Create leaderboard display
6. Create level:

* Create the map and implement it onto a new display, map should loop horizontally so that it feels like the screen is moving.
* Add the health bar on top of the display.
* Create the score variable and keep track of the score.
* Add the reticle/crosshair onto display, create logic for when the player shoots, create amount of ammo/grenades variable and sound effects + visuals for when the player shoots or blows something up.
* **Prototype 2**
* Create the enemy sprite class including both humans and vehicles which allows the enemy to move on the map and “go towards” the player and shoot him.
* Add the number of enemies left on bottom left of display.
* Add the number of ammo and type of gun on bottom right of display.
* Create logic to distinguish whether player has killed the enemy or not.
* Add the Power Ups

1. **Prototype 3**
2. Create pause menu.
3. Create the loss display.
4. Create the win display.
5. **Prototype 4**

## Object Oriented Approach and Class Diagrams

I will be taking an object oriented approach to program my solution. This is so that I can use decomposition to break down my problem by making classes for parts of my game. I am able to do this because python allows users to use object oriented paradigms and because I am using pygame which is a python library it is recommended to do so.

### Benefits of the Object Oriented Approach

One of the main benefits of having an OO approach is that you can develop each section on its own. This can be great as all of the attributes and methods of the class are contained in itself which allow for easier testing and it’s easier to change any parts of the code.

One of the benefits of this approach is inheritance because once a class is defined it can be used as a template for a new sub-class. This could be very helpful for me as I could use it for the weapon class to create a parent class for “weapon” and child classes for the type of weapon.

### Disadvantages of the Object Oriented Approach

One of the main disadvantage of using this approach is that I have minimalistic experience in using this hence making it harder for me to grasp concepts which could lead in this project taking longer to develop.

Another disadvantage is to do with the performance, in some cases OOP can decrease the overall performance of the game because pygame has to deal with creating and managing multiple objects. Therefore, it is essential to organise the code efficiently and to not create unnecessary objects.

### Classes in my Game

I aim to separate most major components of my game into their own classes, here is a list of them, however, as my game develops these classes may change.

Player()

Weapon()

Enemy()

PowerUps()

#### Class – Player()

This player class will mainly be used to store information related to a player, as my game is in first person, it will not require me to draw the player onto the display.

|  |
| --- |
| Class Player() |
| **String** name  **Integer** score  **Integer** high\_score  **Integer** xpos  **Integer** ypos  **Integer** health |
| **Procedure** get\_name  **Procedure** change\_score  **Function** get\_pos |

##### Attribute Description

|  |  |
| --- | --- |
| Attribute | Purpose |
| name | A string which stores name of the player |
| score | An integer which allows a player to be given a score based upon their performance in the level |
| high\_score | An integer which contains the highest score achieved by the player if they have played before |
| xpos | The x position of the cursor on the screen |
| ypos | The y position of the cursor on the screen |
| health | This contains the health of the player in a level, it is between 0 and 100. This valued will be checked throughout the loop to see if the game is over |

##### Method Description

|  |  |
| --- | --- |
| get\_name | This procedure gets the name of the player |
| change\_high\_score | This procedure allows the high score attribute of the player to be changed |
| get\_pos | This function returns the position of the cursor |

#### Class – Weapon()

This class will act as a superclass to create child classes for the rifle and the grenade, which will inherit the attributes and methods below.

|  |
| --- |
| Class Weapon() |
| **Integer** ammo  **Integer** damage |
| **Function** get\_ammo\_amount  **Function** check\_collision  **Procedure** change\_ammo |

##### Attribute Description

|  |  |
| --- | --- |
| ammo | This integer stores the amount of ammo left |
| damage | This integer stores the amount of damage the weapon does. |

##### Method Description

|  |  |
| --- | --- |
| get\_ammo\_amount | This function return the amount of ammo a weapon has. |
| check\_collision | This function checks to see if there was a collision between the bullet from the weapon and enemy, then returns a Boolean value |

#### Class – Enemy()

This class will act as a superclass for the human enemies and the vehicle enemies which will inherit it.

|  |
| --- |
| Class Enemy() |
| **Integer** xpos  **Integer** ypos  **Integer** health  **Boolean** onscreen |
| **Procedure** draw()  **Procedure** change\_health()  **Procedure** move() |

##### Attribute Description

|  |  |
| --- | --- |
| xpos | This variable stores the x position of the enemy, according to the most centre point of it. |
| ypos | This variable stores the y position of the enemy, according to the most centre point of it. |
| health | This variable stores how much health the enemy has |
| onscreen | This variable stores whether the enemy is on screen or not, it only stores True or False |

##### Method Description

|  |  |
| --- | --- |
| draw() | This procedure is responsible for drawing the enemies onto the screen |
| change\_health() | This procedure changes the health of the enemies when they are shot |
| move() | This procedure makes the enemy move |

#### Class – Powerups()

The class Powerups will contain the different attributes and methods required to allow the player to pick up these rewards/powerups.

|  |
| --- |
| Class PowerUps() |
| **String** powername  **Integer** xpos  **Integer** ypos |
| **Procedure** draw()  **Procedure** change\_pos()  **Function** getpos() |

##### Attribute Description

|  |  |
| --- | --- |
| powername | This variable contains a string which contains the name of the powerup/reward the player will receive |
| xpos | This contains the position of the powerup on the display |
| ypos | This contains the position of the powerup on the display |

##### Method Description

|  |  |
| --- | --- |
| draw() | This procedure draws the powerup onto the display |
| change\_pos | This procedure changes the position of the xpos and ypos attributes |
| get\_pos() | This function returns the position of the powerup |

## Algorithms

These are the main algorithms which I will need to implement into my game.

### Main Game Loop

This is the main loop that my game will be running on, the loop will be checking for what state the game is on and provide appropriate responses.

While gamestate != “exit”:

#Check events

If gamestate == “start”:

Play\_bg\_music()

Display\_art()

If space\_pressed(events):

gamestate = “name”

else if gamestate == “name”:

Display\_instructions()

Name = Name\_prompt()

If valid\_name(name):

gamestate = “menu”

else if gamestate == “menu”:

if is\_exit(events) == True:

gamestate = “exit”

else if is\_leaderboard(events):

gamestate == “leaderboard”

else if is\_play(events):

gamestate == “play”

else if gamestate == “leaderboard”:

display\_leaderboard()

if leaderboard\_to\_menu(events):

gamestate == “menu”

else if gamestate == “play”:

update\_map()

player.update()

enemy.update()

weapon.update()

if check\_pause():

pause\_level()

if check\_game\_over():

gamestate == “loss”

else if check\_game\_win():

gamestate == “win”

The main loop is a condition controlled while loop, which loops until gamestate is equal to exit.

gamestate is a variable used for storing what state the game is at.

Throughout the algorithm, if statements are used to compare gamestate with a string.

If the gamestate is equal to “start”, the initial Welcome Image is loaded, and we wait for the player to press space.

If the gamestate is equal to “name”, then the screen which asks the user for their name is displayed, and it waits for a valid username, once it is valid, it goes to the next screen.

If the gamestate is equal to “menu”, the player is shown the menu screen and here he has 3 options, he can exit the game, view the leaderboard or start the game. For each of those events a suitable condition is checked and appropriate gamestate is assigned.

If the gamestate is equal to “leaderboard”, the leaderboard is displayed and I have a condition in place which assigns gamestate the value “menu” when they press the appropriate button. This redirects them to the main menu.

If the gamestate is equal to “play”, the map, player, enemy and weapon get updated. A condition checks if the player has paused the game, if he has I call the pause\_level() function, finally we check if the user has won the game or if he has lost.

If the player loses the game, a suitable screen informing that the player has lost is displayed. The key events are then checked to see if player wants to exit or if he wants to play again.

If the player wins the game, a process very similar to the loss screen happens, instead a winning screen is displayed to the user.

Finally at the end, the display is updated.

else If gamestate == “loss”:

display\_loss()

if loss\_to\_menu(events):

gamestate = “menu”

else if exit(events):

gamestate == “exit”

else if gamestate == “win”:

display\_win()

if loss\_to\_menu(events):

gamestate = “menu”

else if is\_exit(events):

gamestate == “exit”

updatedisplay()

### Player Algorithm

Because our field of view is first person, the player algorithm won’t be very complex as we don’t need to draw anything, however a lot of statistics need to be stored and calculated here.

The main methods in my player class will be these.

Firstly, we calculate the score of the player based upon 3 factors: time to complete level, ammunition and health left before level ended.

The next procedure is change\_score, this procedure will change the score if the current score is greater than the high score of the player. If not I will use a placeholder to just ignore the function.

#player class

Procedure calculate\_score(time, ammo, health):

this\_score = 10000-(time\*10)

this\_score += (ammo + health) \* 1.5

self.current\_score = this\_score

Procedure change\_score(current\_score):

If current\_score>self.high\_score:

self.high\_score = current\_score

else:

pass

The procedure change\_pos will change the position of the x and y variables in the player class based upon where the user’s cursor is on the screen.

The function get\_pos returns the current position of the user’s cursor.

The procedure change\_health changes the player’s health when he either gets shot or when he picks up a power up

Procedure change\_pos(x,y):

self.xpos = x

self.ypos = y

Function getpos():

return self.xpos, self.ypos

procedure change\_health(amount):

self.health+= damage

### Enemy AI Algorithms

This algorithm will process how the enemy moves and shoots at the player, although I have vehicles and human enemies, the below algorithm will stay the same for both as I can make sub-classes for the two to inherit these attributes and methods

#### Enemy Targeting

This algorithm will be implemented to act as a connection between the enemy and the weapon targeting.

The changehealth procedure takes in the damage done by the weapon and applies this to the health of enemy, if the enemy dies, an animation is played which shows the enemy dying.

The function getpos() simply returns the position of the enemy, it will be used to determine whether the player is aiming at the enemy.

#Enemy class

Procedure changehealth(damage):

self.health -= damage

if self.health <= 0:

play\_animation()

self.kill()

Function getpos()

return self.xpos, self.ypos

#### Enemy AI Movement

The enemy will be faster than the speed of the scrolling screen, the vehicles and humans will both be moving from one side to the other, they will stop a certain amount of times to shoot at the player which will deduct health and then they will keep moving.

The procedure movement type is a method inside the class used to randomly choose values which will be used to aid the movement of the enemy. Self.stops is a variable which gets updated with a random integer from 0 to 2, this signifies how many times the enemy should stop to shoot at the player whilst it is moving from one side to the other. Direction is a variable which gets created to signify whether the enemy moves from left to right or right to left.

The procedure move starts off by receiving the direction in which the enemy will move (right to left or left to right).The enemy is spawned to whichever side it will be coming from. It loops until the enemy has exited the screen. The variable x is assigned a random variable between 100 and 300 to signify how much to move and the animation to do. If the stops variable is bigger than 0, the shoot\_player procedure is run which will stop the player and let it shoot and animate this. If the player has exited the screen exited-screen becomes true.

Procedure movement\_type():

self.stops = random.randint(0,2)

direction = random.choice([‘-1, ‘1’])

Procedure move(direction):

If direction == -1:

Self.x = screenwidth+self.w

If direction == 1:

Self.x = 0-self.w

While not exited\_screen:  
 x = random.randint(100,300)

self.x += x\*direction

move\_animation(direction)

If self.stops > 0:

self.stops -= 1

shoot\_player()

else:

if self.x >= screenwidth:

exited\_screen = True

if self.x<= 0-self.w:

exited\_screen = True

### Weapon Algorithm

This algorithm will be developed so that the player can shoot his weapon whilst appropriate graphics are displayed, appropriate sounds are heard and so that the reticle feature is precise in killing the enemies. Note: The weapon class will act as a parent class for the grenade and gun

The shoot function will be called from an external module and will only be run when the user right clicks or left clicks the mouse button. Once he clicks it ammo is subtracted and a sound effect is played, if the cursor is on target with the enemy, the changehealth method is called from the enemy class and the enemy’s health changes respective to amount of damage occurred. If the cursor was not on target a visual effect is also played but no enemies are affected.

Function shoot():

self.ammo -= 1

Soundeffect()

If ontarget():

Enemy.changehealth(damage)

else:

play\_visualeffect()

Function ontarget():

enemy\_pos = enemy.getpos()

if self.x > (enemy\_pos[0]) and self.x <(enemy\_pos[0]+enemy.w):

if self.y> (enemy\_pos[1]) and self.y <(enemy\_pos[1]+enemy.h):

return True

else:

return False

e

The ontarget function gets the position of the enemy, it looks to see if the cursor is within the enemy hitbox on the x-axis, if it is, the function checks if the cursor is within the hitbox on the y-axis. If both are true and the cursor is within the hitbox of the enemy, True is returned, if not, False is returned. This algorithm is designed mainly for the gun as a gun has to hit the enemy straight in order to kill it. When I make the child class for the grenade which inherits these methods, I will change this method a little bit because a grenade can be thrown indirectly to the enemy and still kill it.

### Power-Ups Algorithm

The powerups algorithm will be responsible for randomly choosing a powerup, dropping it, allowing player to shoot at it and receive its benefits.

Procedure move(powerups):

#let’s presume the player gets given 100 ammo for his gun (this value may change later).

first\_drop = random.randint(70,90)

second\_drop = random.randint(45,65)

if weapon.ammo == first\_drop or weapon.ammo== second\_drop:

powerup = drop\_powerup(powerups)

while self.y<100 and self.kill == False:

draw\_powerup(pos, powerup)

use(powerup)

self.y+=1

while self.x <screen\_w+self.w and self kill == False:

draw\_powerup(pos, powerup)

use(powerup)

self.x +=1

if self.x>= screen\_w+self.w:

self.kill()

else:

pass

function drop\_powerup(powerups):

self.x = random.randint(0,screen\_w-power\_w)

self.y = 0-power\_h

powerup = random.choice(powerups)

return powerup

The procedure move takes a list of all of the powerups, it chooses 2 random points in the game when these powerups will be dropped, these powerups will drop when the user has x amount of ammo. If the ammo is equal to one of those 2 drops, drop\_powerup() is run and a while loop is run until self.y is less than 100 or if the powerup was shot, in this loop, powerup is drawn, use() procedure is run and the powerup moves down by 1. Once the powerup is on the ground, the next while loop is run which draws the powerup and calls the use procedure, however, the x value of the powerup increases by 1, which means the powerup moves to the right, once the powerup has left the display it is removed.

The function drop\_powerup, takes in the list of powerups as a parameter, it randomly chooses a value to place the powerup on the x-axis, it changes the y value to be 0 – the height of the powerup, this means the powerup won’t initially be visible. It also randomly chooses a powerup and returns it.

This procedure, takes the powerup randomly chosen in the drop\_powerup function, it checks if the shot was aimed at the powerup, if it was it then checks what type of powerup it was.

If powerup was “health”, the player received 25 health.

If powerup was “ammo”, player received 30 ammo.

If powerup was “damage”, player’s gun’s damage increases by 20 for a certain amount of seconds.

Finally once the powerup has given the player an advantage, it is killed therefore disappears.

Procedure use(powerup):

If shot\_onpowerup():

If powerup == “health”:

player.change\_health(25)

else if powerup == “ammo”:

weapon.change\_ammo(30)

else if powerup == “damage”:

weapon.change\_damage(20)

self.kill()

else:

pass

## Key Variables

For my game to function properly, it will require the storage of numerous key variables. Although I have named and described most of my variables used within classes, there will be external variables within the main loop.

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Justification |
| screen\_w | Integer | This variable will be storing the width of the screen in pixels. It will be needed when I create the display, it is good practice to create a variable because I will be using this value a lot throughout my game, therefore I can change it dynamically. |
| screen\_h | Integer | This variable will be storing the height of the screen in pixels. It will be needed when I create the display, it is good practice to create a variable because I will be using this value a lot throughout my game, therefore I can change it dynamically. |
| gamestate | String | This variable will store a string which will represent the current state the game is at. Throughout my main loop, I will use this variable to determine which condition to run. For example, when the gamestate is “menu” I will display the menu to the user and listen for events. |
| events | List | This list will contain the integers 1 or 0 based upon whether a key on the keyboard is pressed or if the left or right side of the mouse is clicked. This will be updated in my main loop as many other parts of the game will depend upon this. |
| background | Surface | This variable will contain the background image of the level, which will need to scroll throughout the loop. |
| font | Font | This variable will contain the primary font that I will be using throughout the game to inform the player of something. |

## Test Plan

It is vital that I have a robust test plan to ensure that my game works to a good standard. I will carry out these tests and if I cannot pass a test I will make improvements. This type of testing will save me time as I can make small changes during the development stage without leaving errors until it is too late. The test plan will be split into components I will be building, it will have a test description, how I will be conducting this test and what the expected outcome should be.

### Stage 1 – Creating the Welcome Screen

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will test be conducted | Expected outcome |
| 1.1 | Check that game window opens. | Visual test, run the code to open a window. | PyGame window should open at correct resolution. |
| 1.2 | Check that welcome picture is displayed correctly. | Visual test, run the code to open a window. | The welcome picture should fit into the game window. |
| 1.3 | Check that the spacebar control works. | Click the space bar on the keyboard. | The gamestate should change and/or the next screen should be displayed. |
| 1.4 | Make sure background music works. | Put headphones on or connect speakers then run code. | Background music should play with no interruption. |

### Stage 2 - Creating the Name Prompt Screen

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will test be conducted | Expected outcome |
| 2.1 | Check that story of game and tips are on the screen. | Visual test. | This information should be on the screen and should not disappear after user enters name. |
| 2.2 | Check that user can click to type their name into the input box. | Left-click on the input box and enter name. | Should be able to manually input name and this should be visualized. |
| 2.3 | Check that user cannot enter an empty name into the input box. | Left-click on input box and remove any characters in it, then left-click submit. | An error message should be displayed. |
| 2.4 | Check that the submit button works | Left-click the submit button. | The gamestate should change and/or the next screen should be displayed |
| 2.5 | Check that the player’s name gets sent to an external file | Enter a name and click submit. | The external file which contains all names and high scores should be updated with this name. |

### Stage 3 – Creating the Main Menu

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will test be conducted | Expected Outcome |
| 3.1 | Check that all aspects of the display are visible. | Visual Test. | There should be a background with the title on top, 3 buttons, the name of the player and high score. |
| 3.2 | Check that the name and high score screen displays correct values. | Visual test. | The name and high score screen should contain the correct name and should contain the user’s high score, if they don’t have one, 0 should be displayed. |
| 3.3 | Check that the buttons work appropriately. | Left-click on every button on the screen. | After left-clicking on the buttons, the player should be redirected to a new display, however at this stage of development that would not be possible, therefore, the gamestate should appropriately change to the correct value. Gamestate should equal: “leaderboard” after clicking leaderboard and “play” after clicking play |

### Stage 4 – Creating the Leaderboard

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will test be conducted | Expected outcome |
| 4.1 | Check that correct names and high scores are displayed. | Visual test – left-click on the leaderboard button from main menu and check external file. | The names and high scores which were on the external file should be displayed on this display. |
| 4.2 | Check that the list of top 5 high scores is in the correct order. | Visual test – left-click on the leaderboard button from main menu and check external file. | Check which scores are the highest and check whether they are in the same order as that on the display. |
| 4.3 | Make sure Display doesn’t crash when there’s no scores or less than 5 scores. | Visual test – Click on the leaderboard button from the main menu. | The display should show the names and high scores without crashing or the game closing unexpectedly. |
| 4.4 | Make sure back to main menu button is visible and working. | Visual and clicking test. Click on the leaderboard button from main menu and then click it. | The button should be in the bottom right, after clicking it the gamestate should change to “menu” and the display should switch to the main menu. |

### Stage 5 – Creating the Level

#### Stage 5.1 – Creating the Background map

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Test Description | How will the test be conducted | Expected outcome |
| 5.1.1 | Check that the map is displayed behind the HUD and scrolls from right to left. | Visual test | The background image of the map should scroll from right to left behind all other objects; this creates an effect that makes the player think he is moving. |
| 5.1.2 | Check that the map is transitioning smoothly when scrolling | Visual test | The background image should transition smoothly because the image won’t be very long therefore, I will make it identical from one end to the other. |

#### Stage 5.2 – Creating the Player Class

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will test be conducted | Expected outcome |
| 5.2.1 | Check that the class is created properly. | Check that there are no errors or bugs in the creation of this. | There shouldn’t be errors during runtime for the level. |
| 5.2.2 | Check that the method of the player class are created properly | Run these methods from the main loop. | The procedures to change information should be accurate, therefore I should print out the attributes before and after running the procedure. The function should also work to return the position of the cursor. |
| 5.2.3 | Check that the health bar is visible and working. | Check if health bar is visible then manually change the health to 100 and then to 50 and then look at the health bar. | The health bar should be at 100% at first, then, after closing the game and manually changing the health it should be at 50% |

#### Stage 5.3 – Creating the Weapon Class

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will test be conducted | Expected outcome |
| 5.3.1 | Check that the reticle is displayed and is precise in accordance with the cursor position. | go to the top left of the pygame window with the cursor, print out the position of cursor, (0,0) should be printed | The position of the cursor should be the middle of the reticle/crosshair. Therefore, if the middle of the cursor is at the top left it should read (0,0) in console |
| 5.3.2 | Check that the appropriate sound effect is heard after shooting or throwing a grenade. | Plug headphones into PC and left-click, the sound of a uzi gun should be heard. After right-clicking, the sound of a grenade should be heard. | When left-clicking, a bullet should be shot and ammo count for gun should decrement by 1 on the HUD whilst creating a visual and sound effect. When right-clicking a grenade should be thrown, decrementing grenade count by 1 on the HUD and creating a much louder and visual effect. |
| 5.3.3 | Check that the appropriate visual effect is shown after shooting or throwing a grenade. | Plug headphones into PC and left-click, the visual effect of a uzi gun should be seen. After right-clicking, the visual effect of a grenade should be seen. | Only one of the two events should happen, either a grenade is thrown, or a bullet is shot. Both events should not happen. |

#### Stage 5.4 – Creating the Enemy Class

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will the test be conducted | Expected outcome |
| 5.4.1 | Check that the enemy is visible and is moving randomly in a non-pattern way. | Visual test | The enemy should be visible on the screen. It should randomly go from one side to the other and randomly stop between 1 to 2 times to shoot at the player. |
| 5.4.2 | Check that the enemy is creating a visual effect when it shoots. | Visual test. | The enemy should walk to the screen, when it stops it should flash. |
| 5.4.3 | Check that the enemy is doing damage to the player. | Print the player’s health on the console and visually check health bar. | The player’s health should decrease when the enemy shoots the player, we can know if the enemy is shooting because it’ll flash. |
| 5.4.4 | Check HUD to check number of enemies left. | Visual test, without shooting at the enemy look at the bottom left of level display. | The number of enemies left should match what was specified in the enemy class. |
| 5.4.5 | Check that enemy dies if you shoot at it. | Create a temporary test in the main loop to print “dead” in the console log if the enemy is killed. | The enemy should disappear from the screen and be removed from the list of enemies once killed. The number of enemies left should also decrement by 1 on the HUD. |

#### Stage 5.5 – Creating the Power Ups Class

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Test Description | How will the test be conducted | Expected outcome |
| 5.5.1 | Check that the powerups are being displayed properly on the map and “flowing” smoothly. | Visual test. | The powerups should fall down onto the map from the sky, they should then land at a certain point and stop falling vertically, they should then move with the same speed as the map in the x-axis, creating an effect that the powerup is moving along the map. |
| 5.5.2 | Check that the power up can be shot at and is removed after. | Whilst in the level, wait for the power up to spawn and then shoot at it. | After shooting the power up, it should disappear. |
| 5.5.3 | Check that the power up is having an effect systemically. | Create a temporary procedure in the main loop to print out the player’s health, ammo and damage. Do a visual test to see difference in value on HUD. | The health, ammo or damage of weapon should increase from the initial value, this difference can be seen in the console or on the HUD |

### Stage 6 – Creating the Pause Menu

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will the test be conducted | Expected outcome |
| 6.1 | Check that the player can pause the level. | Press spacebar on the keyboard. | The pause menu should be displayed. |
| 6.2 | Check that the game is not continuing in the background and that the time taken to complete the level is paused. | Create a procedure to print out all major values such as the time taken to complete level and health. This procedure should print out these values to console. | The values should all remain the same to ensure that nothing is happening in the background whilst the player thinks the game is paused. |
| 6.3 | Check that the keyboard inputs work. | When on the pause menu, press enter to resume. Go to the pause menu again and press ESC. | After pressing the spacebar, the level should resume from where the player left off, after pressing ESC the game should close. |

### Stage 7 – Creating the loss display

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will the test be conducted | Expected outcome |
| 7.1 | Check that this display is only displayed on a loss. | Lose a game by not shooting at anyone. | The loss display should be displayed. |
| 7.2 | Check that the keyboard inputs work. | When on this screen, press Space.  Go onto this screen again and press ESC instead. | On the first time, the player should be redirected to the main menu. On the second time, the game should quit. |

### Stage 8 – Creating the Win Display

|  |  |  |  |
| --- | --- | --- | --- |
| Test  No | Test Description | How will the test be conducted | Expected outcome |
| 8.1 | Check that this display is only displayed on a win. | Win a game. | The win display should be displayed. |
| 8.2 | Check that the keyboard inputs work. | When on this screen, press Space.  Go onto this screen again and press ESC instead. | On the first time, the player should be redirected to the main menu. On the second time, the game should quit. |
| 8.3 | Check that the right score is displayed on this screen. | Lose a game and count how many seconds it took you to lose the game, count your health and amount of ammo. | Use the formula: Score = (10000-(time\*10))+  ((ammo+health)\*1.5)  This score should match with the score shown on the winning display along with the name shown. |

# C. Developing the coded solution (“The development story”)

## Using Python

I will be using the Pygame library in python to create my game. Pygame is a free and open-source Python library that provides a platform for creating 2D video games and multimedia applications. This would be the best option for me because I have some experience with python, therefore, I’ll be able to produce a better game as I won’t have to learn new languages or engines such as Unity and C#.

To make use of Pygame within my program, python needs to be installed, this is usually pre-installed on Apple devices which comes with MacOS, however, windows and linux users need to manually install it. Another prerequisite is that the pygame library also needs to be installed and needs to get imported into my program. The most common features of this library are as follows:

* Screen initialisation (creating the display windows), drawing onto it and updating through the main loop using pygame.display().
* Event handling using pygame.event.get()

## My Code Structure

In order to keep my code organised and easy to maintain, I will be making use of decomposition by separating parts of my code into separate files.

At this stage those files will be:

* Main.py – contains the main game loop which calls other functions.
* Player.py – contains the code for player class.
* ADD MORE AS I WRITE CODE

## Stage 1 – Creating the Welcome Screen

### Initialising Pygame and the Display

The first part of creating the welcome screen is initialising the pygame library and the display.

A white background with black text

Description automatically generated

This function is used to initialize the pygame library and the display, the variables are made global so that they can be accessed outside the function, the pygame library is initialised, the width and height of the display are created as variables, the gameDisplay variable contains the details about creating the display which is 1000x1000 pixels in size. The clock variable is created and this keeps track of time.

### Event handling

In the code below I will be handling all events, this means when the player presses a button or clicks on their mouse, I should be storing that input somewhere so that I can use it in the game.

A computer code with text

Description automatically generated with medium confidence

I start by creating a dictionary which stores keys which are the buttons the user will press/click and I assign the value 0 to signify the button is not being clicked and 1 to signify that it is being clicked. Next I am creating a function called get\_events which makes use of some pygame syntax to handle these events. I firstly check if the player has clicked on the big X on the top right of the display screen. If they have I assign the key “quit” the integer 1. I then check if a keyboard key is pressed down, if it is I look for which key is being pressed down and if it is space or the escape button their respective dictionary keys also get assigned 1. I then have to check if the player has released their finger of the key and if they have I check the 2 keys from before and assign the integer 0 to their respective key in the dictionary. The same principle happens for the next few lines, however the mouse clicks are checked instead. At the end of the function I return the events so that I can refer to them in the main loop.

### Image and Music Loading



The syntax above is used to load the main picture which the user is greeted with and the background music which loops.

### Colours

Throughout my game I will need to change the colours of many objects, backgrounds etc… Therefore, I have created a list of variables which store RGB values as tuples.

A math equations with numbers

Description automatically generated with medium confidence

### Functions

I created a separate file in which I create functions, I am doing this to minimise writing in the main file, this is so that it is easier to read for outsiders. Below are the main functions I have created so far which will help me to create the welcome screen:

#### Can\_proceed

A close up of text

Description automatically generated

This function “can\_proceed” asks for the events dictionary to be passed in when it is called. If the spacebar is pressed, the string “name” is returned, if not, the string “start” is returned. I have created this function because I can’t change the values of a variable in another python file therefore

#### Play\_music

A close up of words

Description automatically generated

The function “play\_music” takes the music which needs to be played in the background of the welcome screen. The function checks if any music is currently being played, if not then the background music gets played. In music.play(), .play() takes in a parameter which indicates how many times the sound needs to be played, -1 signifies the sound will be played an infinite number of times until it is stopped.

#### Draw\_cover



The draw\_cover function simply draws the background picture which is passed in as a parameter.

### A close-up of text Description automatically generatedMain Loop

This is the main loop in the game so far, gamestate is set to “start” as default before the loop starts. This loop will keep looping until the user wants to exit. I create a variable called events which receives the dictionary after running the function I mention above called get\_events(). If the user wants to quit the game by pressing escape or hitting the cross on the top right of the display gamestate is changed to “end”, if gamestate is “start” (which it is by default), play\_music is run to play the background music, draw\_cover is run to display the cover on the screen and I then check to see if the user has pressed spacebar to proceed. I have not started writing the code for when gamestart = “name” as that is for stage 2. The pygame.display.update() is a prebuilt pygame function which updates the display, this is necessary as if a change is made which affects the display, the display needs to also change. The 2 lines at the end simply quit pygame and close the window.

### Progress Visualised

A monkey holding a banana

Description automatically generated

This is the program so far, when the user presses space, that is registered and what happens next needs to be programmed in the next stage.

### Stage 1 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | Data and Type | Outcome |
| 1.1 | Check that game window opens. | 6 | Visual test, run the code to open a window. | Game window opens with no issues, it also closed with no issues. |
| 1.2 | Check that welcome picture is displayed correctly. | Partially 6&7 | Visual test, run the code to open a window. | Welcome picture is displayed correctly, it does not flicker and is aligned correctly. |
| 1.3 | Check that the spacebar control works. | 2 | Click the space bar on the keyboard. | I printed something to check if user can press space key and move on, it worked |
| 1.4 | Check that background music works. | 8 | Put headphones on or connect speakers then run code. | I check if the music works by putting headphones on, and waiting for a few minutes to see if it is looping and it is. |

### Stage 1 – Problems and Solutions

When developing this welcoming screen, these issues were discovered:

When I was coding the function to handle events, I created the variable which had the dictionary containing the key-value pairs for the inputs inside the function, this was a problem as pygame registered a single press when I help a keyboard key. It is good I found this small bug early because it would’ve affected the level, for example, if the player wanted to hold left-click on the mouse to shoot the automatic gun, it wouldn’t have allowed him to do that as it would only shoot once and wait for the player to lift their finger and then click again to shoot 1 shot. I fixed this issue by creating the variable outside the function.

Another small issue was finding the syntax to play the background music, I overcame this issue by finding a similar question to mine on stack overflow.

Apart from that there weren’t any major problems, there were of course some minor problems such as missing certain characters from the syntax, or using == instead of =.

## Stage 2 – Creating the Name Prompt Screen

Below is what the name prompt screen looks like at the end of this stage. I have a title at the top and a background story and tips for the player on the left, the user can input their name into an automatically resizable box and then press the submit button which is a class I created as I will need to add buttons later on. Once the player presses submit and the name is valid, the name is added into a JSON file if the player has never played before. After that, the player is directed to the main menu.

A screenshot of a phone

Description automatically generated

A screenshot of a computer

Description automatically generatedI have added some validation so that the user cannot proceed without providing a name.

An error message in red is displayed to the player to let them know that they cannot proceed with an invalid name. Overall, the infrastructure and functionality work, however, the graphics are not really that good. My main concern for now is to make the game work and be efficient so that the user can play, I can work on the graphics once the game works systemically as that won’t be too hard to implement.

### Main Loop

A black text on a white background

Description automatically generated

This is the addition I have added to my main loop after the last stage, the if statement checks if the gamestate is equal to “name” (the gamestate is only changed to “name” when the player presses the spacebar on the welcome screen). If it is, the function “draw\_message\_and\_title” is quite self-explanatory, it draws the title of the game and the 2 green messages as shown above. The function ask\_name creates the box where the user can input their name, once user has inputted their name the functions returns this to the variable name in the main loop. The function display\_text, displays the “Enter Name:” message. I have created an instance of the class button, this is the submit button under the input box in the next line I run a method of the class which draws the button and updates it to display if name is not valid or to add the name into the JSON file.

### Drawing the Messages and Title

A close up of a computer code

Description automatically generated

This function which is run in the main loop firstly fills the display to be all white, it then creates a render of the title “OPERATION MONKEY”, the False means we are not using antialiasing and this text will be in black. The title is then blitted/drawn onto the screen, the co-ordinates are calculated using the width of the screen and title, this ensures the title is in the middle. After that, the two messages are drawn onto the screen at the co-ordinates specified. Message1 and message2 are variable assigned in my settings.py file with the images of the text messages in green. Here is the syntax for that:



Pygame.image.load() is pygame syntax which allows images to be loaded into the game so that they can be assigned to variables.

### Creating the name input box

A screenshot of a computer code

Description automatically generated

This function takes in the game display as a parameter so that it can draw to it later on. Firstly, some variables are made global inside the function, this is so that if I make any changes to them inside the function, they will be affected outside it as well. Next I check if the player’s cursor has clicked the input box, if it has it triggers the inside\_input\_box variable to change to True. I then create the logic for the user to use their keyboard to enter their name, if they have clicked on the input box then I check the events to add or subtract the characters from their name. If they press backspace I remove a character from the name\_placeholder if they press another key I add that to the name\_placeholder variable. I also have syntax to change the design of the input box when the player clicks on it. At the end I draw the input box itself on the display and draw the text the user inputs onto it as well. Because some people may have names which are longer than the input box, the input box changes length after 300 pixels in width to adjust with the name. At the end of the function the variable name\_placeholder is returned which contains what the player has typed.

### Displaying the “Enter Name” Text

A close up of a text

Description automatically generated

To display the “Enter Name:” text on the left of the input box, this function is created. It takes in the colour, x co-ordinate, y co-ordinate, message and font as parameters. It first renders the message and then displays it onto the display.

### Button Class

A close-up of a computer screen

Description automatically generated

I have a variable before the class constructor called clicked which represents whether the button is clicked or not. In the constructor the following variables are taken as parameters: x and y co-ordinates of button, height and width of button, gamedisplay, colour of button, where the button would lead to when pressed, text to be shown on button, font-size of that text, colour of the text and the events dictionary. The part below def \_\_init\_\_ is called attribute initialization. It is used so that variables that were passed into the constructor can be used in the methods of the class.

#### Class Methods – Draw()

A text on a white background

Description automatically generated

In the method draw(), the rectangle is drawn onto the display, using line 2 and 3, then the text to be drawn (“SUBMIT”) is rendered. I then get the rectangle of the text to be drawn and finally I draw that text onto the display on top of the button.

#### Class Methods - Update()

A computer code with text

Description automatically generated

In this method, I run the draw() function which I mentioned previously and then get the position of the cursor. I then use some logic to check whether the user has clicked on the button and if they have the self.clicked variable is set to True.

#### Class Methods – Name()

A computer code with text

Description automatically generated with medium confidence

In the method name() I run the update() function mentioned previously, I check if the player is clicking on the button. If they are I check if the name is valid, if the player is clicking submit with a name with no characters, they get an appropriate warning. If they’ve entered a name, I load the JSON file I created and check whether that name exists or not, if it does not I add the username into the JSON file and return “menu” to allow the gamestate in the main menu to be changed to allow the next screen to be displayed. I return an appropriate string at each stage to allow gamestate to be changed if user clicks “submit” and name is valid. This would redirect the player to the next screen.

### Stage 2 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | Data and Type | Outcome |
| 2.1 | Check that story of game and tips are on the screen. | 6 | Visual test. | This works, the story of game and tips are on the screen in the format of a text message. |
| 2.2 | Check that user can click to type their name into the input box. | 6 | Left-click on the input box and enter name. | This works, the user can click on the box to type their name |
| 2.3 | Check that user cannot enter an empty name into the input box. | 6 | Left-click on input box and remove any characters in it, then left-click submit. | This works, the user cannot press submit with an empty name because an error message will be displayed |
| 2.4 | Check that the submit button works | 6 | Left-click the submit button. | The submit button works as when you click it, the gamestate changes (if name is valid) |
| 2.5 | Check that the player’s name gets sent to an external file | 6 | Enter a name and click submit. | This works because when the player enters the name, it is instantly stored in a JSON file. |

### Stage 2 – Problems and Solutions

When developing the name prompt screen I discovered more problems than the previous stage. These are the main ones:

* I encountered a problem where I was trying to save the name the user inputs onto the input box. The player would type a letter into the input box and that would update the name variable instead of adding the letter onto the existing characters in the string. There was a very simple fix to this which I already knew about but didn’t release, I wasn’t declaring the variable in the function as a global variable, doing that fixed the issue.
* I encountered a problem with adding the names and scores into the JSON file. When I clicked submit with a new username, my function would add duplicates into the JSON file. I found out that the issue was caused by the conditional statements in my loop. I would loop through every object in the file however my logic in the statements meant this: If the name the player entered is in the JSON file then pass, else add the new object. To fix this issue I created new logic and added a flag called “user\_exists”.

## Stage 3 – Creating the Main Menu

At this stage I have created the next screen the user is directed to which is the main menu. The title is at the top, the player can see their name and high score on the left, they can click on the right where it says “leaderboard” to go to the leaderboard, clicking on play will take them to the actual level and pressing the escape key allows them to exit the game.

A screenshot of a game

Description automatically generated

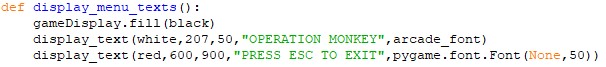
### Main Loop

A close up of text

Description automatically generated

This is the addition I have added to my main loop. When gamestate changes to “menu”, the title and writing at the bottom right are blitted to the screen using the display\_menu\_texts() function, the display\_name\_score function displays the name of the player and their high score. I then create 2 classes for the play and leaderboard button and run the precedence function which returns the correct gamestate.

### Displaying the Menu Texts and Title



This function does not take any parameters, it starts off by filling the display background black, it then displays the Title “OPERATION MONKEY” in white at the co-ordinates (207,50) with the arcade font, the next line is very similar but it displays the text at the bottom right of the screen to inform the user how to exit the game.

### Displaying the Name and High Score of the Player

A computer code with green and blue text

Description automatically generated

To display the name and high score of the player this function is used, it takes in the name of the player which was a variable assigned in the previous if statement in the main loop. It loads the JSON file using the load\_name\_score() function, it then loops through each object in that file and finds the high score of the player and assigns it to a variable “high\_score”. Finally using the display\_text() function, the name and high score are displayed on the left side of the screen appropriately.

### Redirecting After Pressing a Button

To create my 2 buttons I have used the button class I created in the previous stage, however, I added a new method called redirect which runs the update() function and then checks if the button is clicked, if it is, the name of the gamestate destination string is returned, if not, the current gamestate is returned.

A computer code with black and orange text

Description automatically generated

### Deciding Which Button is Clicked

A close up of text

Description automatically generated

The precedence function takes in the gamestate returned by the first button, the gamestate returned by the second button and the current state. If the first button is being pressed then the gamestate of the first button is returned. If the second button is pressed then the gamestate of the second button is returned. If neither are pressed then the current gamestate is returned. This function was built because I couldn’t return 2 gamestates separately due to the fact that the interpreter would take precedence over the gamestate which was at the bottom of the other one.

### Stage 3 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | Data & Type | Outcome |
| 3.1 | Check that all aspects of the display are visible. | 6 | Visual Test. | This is working, all components which I initially added in the screen design are implemented. |
| 3.2 | Check that the name and high score screen displays correct values. | 6 | Visual test. | This is working, after checking the JSON file against the screen both values and names add up. |
| 3.3 | Check that the buttons work appropriately. | 6 | Left-click on every button on the screen. | This is working, after clicking the play and leaderboard button, a response is given and gamestate changes appropriately. |

### Stage 3 – Problems and Solutions

I didn’t encounter as many problems as the last stage, however I had some small problems which wasted a lot of my time. These are the main ones:

* Indentation: When I was creating the logic to display the name and high score of the player, I manually changed the score of a test player in the JSON file, I then “logged-in” as him to see whether the right score was shown and it was not, 0 was shown instead of 1000. I then looked in the JSON file and a duplicate of the player was being made with a score of 0. Therefore, I looked at the syntax of the name method in the button class and realised my indentation was wrong as I was executing an if statement in a for loop by accident. Therefore, I simply changed the indentation and it worked again. This would have been a major problem for test 3.2 if I did not.
* Circular Imports: When creating the instances of the button class for the play and leaderboard buttons, I was creating them in the functions.py file whilst importing buttons.py which contained the button class, however, buttons.py was importing the functions from functions.py, therefore, there was a case of circular importation which is usually not a good idea, in my case it simply did not allow me to proceed with the programming. Therefore, I created the instances in the main.py file instead and did not import buttons.py to functions.py. If I did not change anything, test 3.3 would have been failed.

## Prototype 1

### Analysis of Feedback from Stakeholders

I have asked stakeholders to provide some feedback for the first prototype, I am going to summarise this into the positives and negatives as follows:

Positives:

* TO DO

Negatives:

* There’s a bug in the menu where the player enters their name, the keyboard inputs are very unresponsive.

### How did I fix this negative feedback?

I ran my game file and did some testing with the name input box to check whether there was any pattern. The bug was as follows, I had to type in a letter a few times for it to actually register and appear on the screen, after doing some testing, I realised that there were two more issues with responsiveness. When I clicked on the submit button, it took a few clicks for it to register, it also took the close window button on the pygame window a few clicks for it to register to close the window. This gave me the biggest clue to find out what was wrong, the issue I was encountering was to do with my event handling. I had two for loops to check for the events, one was the get\_events() function and the other one was within the ask\_name() function. This is very bad programming practice in pygame because only one of the for loops will register the events and the other one will simply not register it. Because the main game while loop was iterates countless time per given time, it was almost impossible to predict when the event will be registered. As a result, I added some more keys in the events dictionary to take binary values for specific keys to check whether they were being clicked/pressed. I then used this in the ask\_name function instead of having another for loop to check events as A screen shot of a computer code

Description automatically generatedfollows:

I pass in the events dictionary into this function and access the values from each key from the dictionary.

I encountered another issue from this, when I inputted a character into the input box, multiple characters appeared on the screen. This was because the main loop iterates a lot of times per second, therefore, it will register a character being pressed a lot of times, even if the player only pressed it for half a second, this means multiple characters appear on the screen, likewise if the user presses backspace, multiple characters are removed from the screen. To fix this issue, I added a cooldown using pygame.time.get\_ticks() so that a character is only registered every 115 milliseconds.

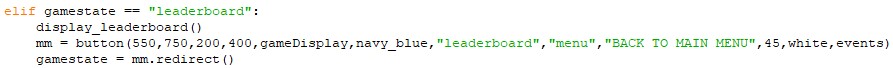
## Stage 4 – Creating the Leaderboard

A screenshot of a computer

Description automatically generated

This is the finished leaderboard, there is a title for the leaderboard on top, the top 3 scores and the names of the playes and there is a button which takes you back to the main menu.

### Main Loop



The code above is what I have added into the main loop. Firstly, the logic checks if the gamestate is equal to the string “leaderboard”, if it is then the display\_leaderboard() function is run which displays the leaderboard and title. I then create an instance of the button class to create the button to go to main menu, I then run mm.redirect() which returns the gamestate to check whether the user has clicked on the button or not.

### Displaying the Leaderboard

A computer code with different colored lines

Description automatically generated with medium confidence

When the display\_leaderboard() function is run from the main loop, the background colour is changed to black and “Leaderboard” title is displayed. I run the calculate\_positions() function to give me the positions of the players or the string “invalid” if there are not enough results. I check if the position is invalid if that is the case I display an error message. If the positions are not invalid I display the string “name” and “score” using the display\_text() function which I previously explained. I then use a for loop to loop 3 times (because I am displaying the top 3 scores), through this I am able to display the individual names and high\_scores. I use a for loop because I can display the text without repeating myself. E.g. for the y-axis parameter I use 350+(100\*x), which means that the function can start blitting the text at 350 pixels down then go 100 pixels down each iteration.

### Calculating the Position of the High Scores

To calculate the position of the high scores I create a function called calculate\_positions(). This starts off by loading the JSON file, it then creates 3 variables which are assigned a dictionary with values which act as placeholders. I check whether there are less than 3 scores and names in the JSON file, if there are I return “invalid”. If there are 3 or more scores I create some logic to order the high scores into order. I loop through each object in the JSON file dictionary, if the high score of the object x is bigger than the high score of the current first object then the first object becomes x and the second object becomes a copy of the first object and the third object becomes a copy of the second object. If the high score of x is not bigger than the first object but it is bigger than the second object, the second object takes the value of x and the third object becomes a copy of the second object. If x is not bigger than the first or the second object but it is bigger than the third object, the third object becomes x. Once it is finished with this for loop, the function returns the first, second and third object.

A screenshot of a computer code

Description automatically generated

### Stage 4 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | Data & Type | Outcome |
| 4.1 | Check that correct names and high scores are displayed. |  | Visual test – left-click on the leaderboard button from main menu and check external file. | The correct names and scores are being displayed, I have checked the external file and inputted values manually to make sure. |
| 4.2 | Check that the list of top 3 high scores is in the correct order. |  | Visual test – left-click on the leaderboard button from main menu and check external file. | The list is in correct order, I have checked this by looking at the external file and comparing. |
| 4.3 | Make sure Display doesn’t crash when there’s no scores or less than 3 scores. |  | Visual test – Click on the leaderboard button from the main menu. | When there are no scores or less than 3 scores the screen does not crash, an error is displayed. |
| 4.4 | Make sure back to main menu button is visible and working. |  | Visual and clicking test. Click on the leaderboard button from main menu and then click it. | The button which redirects player to main menu is visible and it is working because when I click on it, it takes me to main menu and gamestate changes to “menu” |

### Stage 4 – Problems and Solutions

When I was coding this part of my game, I only had one issue and that was finding ways to output the player’s high scores in the right order. To overcome this I watched some tutorials on YouTube and searched online on websites such as Stack Overflow. Subsequently I translated the logic into code.

## Stage 5 – Creating the Level

### Stage 5.1 Creating the background map

For this stage I created the scrolling background map, I used photoshop to create a static image such as the one below, it’s 2000x1000 and when it “scrolls”, the player cannot tell it’s the same image due to the symmetry

#### Scroll Background Function

A screen shot of a computer

Description automatically generated

I take in the game display as a parameter, I then create a global variable for the x co-ordinate value of the background image with respect to the display. I decrease the value of this variable by “scroll\_speed” each cycle. Once the value of the x co-ordinate of the background image reaches -2000, I change the x co-ordinate to 0. I then display the photo from above using the blit() function onto the display at x c-ordinates “background\_x” and “background\_x + 2000”.

#### Stage 5.1 Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | Test Description | Success Criteria Reference | How will the test be conducted | Outcome |
| 5.1.1 | Check that the map is displayed behind the HUD and scrolls from right to left. |  | Visual test | This test has passed because the map does indeed scroll from right to left and it is behind the HUD |
| 5.1.2 | Check that the map is transitioning smoothly when scrolling |  | Visual test | This test has passed because I used symmetry in the image |

### Stage 5.2 – Creating the Player Class

At this stage, I created my player class which allows the creation of an object which contains the details about the player itself.

#### Constructor Method

A white background with black text

Description automatically generated

I take in the current score, high score and health as parameters. I then assign them to their corresponding variables.

#### Change\_Health Method



This method takes in as a parameter how much the health should change by. This is then used to change the health.

#### Health\_Bar Method

A close-up of a white background

Description automatically generated

This method only takes in the game display as a parameter. After calculating a ratio of the health of the player compared to the “full” health. This number is used to draw only a certain percentage of the green rectangle below as the health changes.

A green rectangle with red squares

Description automatically generated

This health bar is showing the health at 63% because only 63% of the box is filled in green. This is a nice way to show the player their health because it is very minimalistic so it does not take up too much space on the screen.

#### Change\_Name Method



This is a fairly self-explanatory method, it changes the name of the player. I have had to use this because I create the player class outside the loop, however the player enters their name inside the loop so I need a way of storing this information. Therefore I run this method after the user enters their name.

#### Stage 5.2 Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | Data & Type | Outcome |
| 5.2.1 | Check that the code runs / no syntax errors. |  | Check that you don’t get any syntax errors by executing the file. | There are no syntax errors therefore test has passed |
| 5.2.2 | Check that the method of the player class are created properly |  | Run these methods from the main loop. | This test passed as the methods are all working as expected. |
| 5.2.3 | Check that the health bar is visible and working. |  | Check if health bar is visible then manually change the health to 100 and then to 50 and then look at the health bar. | This test passed as the health bar is working and the corresponding health is displayed at 50 and 100. |

#### Stage 5.2 Problems and Solutions

The only difficulty was setting up the infrastructure to create the health bar, I had to think twice about how I will create this so that it is dynamically changing.

### Stage 5.3 – Creating the Weapon Class

During this stage I created everything related to the user’s weapon. Here are some of the main features implemented/coded:

* The hitbox
* Weapons and shooting effects
* Displaying the ammunition left onto the screen

#### Constructor Method

A math equation with black text

Description automatically generated

The weapon class’ constructor contains the following parameters:

* Type of weapon
* Amount of bullets
* Delay between shooting

#### Creating the Hitbox

I have made a method in the weapon class called draw\_hitbox, here is what it looks like:

A screen shot of a computer program

Description automatically generated

I pass in the game display and colour of the hitbox/reticle as a parameter. I then remove the cursor from the screen once the user hovers over the display using the built in pygame function. The next 4 lines of code create 4 rectangles which surround the invisible cursor like this: 

#### Weapons and Shooting Effects

A computer screen shot of a program code

Description automatically generated

The method shoot\_effects is inside the weapon class. It takes in the events for the mouse buttons, sound, game display, x and y co-ordinate of the cursor as parameters. The x and y co-ordinates are then saved onto their respective class attributes.

The next if statement checks if the mouse is being clicked, and if the number of bullets left is bigger than zero. I then create the variable “now” to keep track of the time. There is another if statement below this which I created to delay the shooting, this makes sure that when the player left-clicks or right-clicks on the mouse, the bullets don’t decrease too quickly and that the sound and visual effects are in sync. At the end, I create the self.last\_shot attribute which takes in the time and this is then used in the if statement.

#### Displaying the weapon HUD

A screen shot of a computer code

Description automatically generated

The method display\_HUD is also in the weapon class and takes in these parameters:

* Image of the weapon
* Game display
* X and Y values of the image
* X and Y values of the text which shows the ammo

I then display the image and the text which contains the bullet number onto the screen, and this is what it looks like:

A screenshot of a game

Description automatically generated

#### Stage 5.3 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | Data & Type | Outcome |
| 5.3.1 | Check that the reticle is displayed and is precise in accordance with the cursor position. |  | go to the top left of the pygame window with the cursor, print out the position of cursor, (0,0) should be printed | The correct output is printed implying that the reticle/hitbox is synced with the position of the mouse cursor. |
| 5.3.2 | Check that the appropriate sound effect is heard after shooting or throwing a grenade. |  | Plug headphones into PC and left-click, the sound of a uzi gun should be heard. After right-clicking, the sound of a grenade should be heard. | The test has passed because the appropriate sound is heard after clicking on both buttons on the mouse. The sound is also synced in with the ammo count. |
| 5.3.3 | Check that the appropriate visual effect is shown after shooting or throwing a grenade. |  | Plug headphones into PC and left-click, the visual effect of a uzi gun should be seen. After right-clicking, the visual effect of a grenade should be seen. | The test has passed because the appropriate visual effect is displayed after clicking on both buttons on the mouse. The visual effects is synced with the ammo count. |

#### Stage 5.3 – Problems and Solutions

During this stage I encountered a lot of problems and finding a solution to this took a very long time. One of the main problems was:

* When I left clicked on the mouse to shoot, the bullet count would go to 99 and then go back up to 100, it would not go below 99. This took me a long time to solve but the solution was very simple. I was creating an object of the weapon class inside the main loop, that meant the logic inside of the loop allowed me to shoot, however when the loop ran again, the ammo count was reset to 100 because the ammo count was one of the parameters in the class constructor.

### Prototype 2

A screenshot of a video game

Description automatically generated

#### Analysis of feedback from stakeholders

I asked stakeholders for feedback on this prototype, below is the summary:

Positives:

* The health bar is really good.
* The shooting is very responsive and synchronized with the sound and visual effects.
* The location of the elements such as the ammo count is really good.

Negatives:

* The background scrolling is slow.
* The visual effect after shooting is not very good.

As a result of this consultation with my stakeholders I have successfully made the scrolling speed faster by changing the value of the variable scroll\_speed to 5 and have created new designs for the shooting effects.

Below is my shooting effect for the uzi gun, it is simply an X which pops up every time a bullet leaves the gun:

A black background with a black square

Description automatically generated with medium confidence

Here is the visual effect seen when the person throws a grenade:

A yellow and red explosion

Description automatically generated

### Stage 5.4 Creating The Enemy Class

#### Constructor Method

A screen shot of a computer screen

Description automatically generated

This is the constructor method from my enemy class, it is used to create both soldier enemies and tanks. Attributes are variables specific to an object, these can be local values or they can be values which are updated to whatever is passed as a parameter. In my case I have passed in the following parameters:

* gx – the x position of the enemy
* gy – the y position of the enemy
* ghealth – the starting health of the enemy
* gspritesheet – a list containing the spritesheet for the enemy which contains the different animations when the enemy is moving or shooting.
* gstops - the amount of times the enemy stops during the movement animation to shoot.
* gdirection - the direction in which the enemy is moving, 1 represents if enemy moves from left to right, -1 represents movement from right to left. This is randomly chosen during object creation.
* gspeed – the speed of the enemy.
* gcooldown – the amount of time an animation lasts for before it is switched onto the next one.
* gplayer – the player object which contains access to attributes and methods relevant to the user playing.
* gtype – this parameter contains the type of the enemy, for example it is either “tank” or “player”.
* gw – the width of the enemy.
* gh – the height of the enemy.

I also use a conditional statement to change the velocity of the enemy and make it negative if the direction passed in is from left to right.

#### Move Method

This method is used to allow the enemy to move, it contains code to move the enemy’s x-value and to also control how the animation works.

A screenshot of a computer program

Description automatically generated

In this method, I use pygame’s time function to control the enemy’s animation. I start off by making a variable called current\_time which takes in the value of the time elapsed since the function pygame.time.get\_ticks() was called. I then check whether the enemy is moving using the self.moving attribute, which is set to true later on if the enemy is moving. If the current time variable has exceeded the cooldown, the frame increases which allows the animation to increase and the sef.last\_time variable is set equal to the current\_time to allow the conditional statement to be ran again for the next iteration without causing any problems. The x co-ordinate of the player is also changed depending on the speed and the self.rect’s x and y value. I then check to see what direction the sprite is supposed to go, if it is going from left to right (self.direction = 1), then the current animation from the spritesheet is drawn and not flipped. If the player is moving from right to left, the sprite is drawn onto the screen at the current frame and flipped.

#### Shoot method

This method is responsible for allowing the enemy tank or soldier to shoot the player, it takes in the game display as a parameter

A screen shot of a computer code

Description automatically generatedI take a time stamp of the current\_time and then check if enough time has passed since the last shot. If enough time has passed since the last shot I make self.last\_time equal to current\_time, this is so that the conditional statement keeps working, I then draw the 4th sprite from the spritesheet at the current x and y coordinates and decrease the user/player’s health by -0.5(This may change in the future). However, if the time between shots is less than the cooldown, I simply draw the 3rd sprite from the spritesheet at the current x and y coordinate.

#### Stop\_Move Method

This method is used to process when the enemy stops and shoots at the player.

A screen shot of a computer screen

Description automatically generated

The only parameter being passed into this method is the game display.

I first check whether the enemy is stopping once or if it is stopping twice. This is because I have added an element of randomising whether the enemy stops to shoot at the player once or if the enemy stops to shoot at the player twice in one movement from left to right or right to left. I first check whether the enemy will only shoot once, I use the conditional statement “if self.stop2 == None” because if the enemy only stops once, self.stop2 is automatically set to “None” when enemy object is made. I then do another check to see whether the enemy’s x position is on the self.stop1 position which it was assigned to stop at, and if the enemy has not passed this stop yet. If that’s the case, I change the self.moving attribute to False and the self.stop1\_passed attribute to True, these attributes’ change in Boolean value impacts a lot of other processes in this class such as whether the self.move method and the next part of this method.

Nextly, I check whether self.stop1\_passed is True, and if it is I check if self.stop1\_time is equal to None, this is because I subsequently assign the attribute a time stamp of when the enemy stops. I then assign the variable elapsed\_time the time difference between the current time and the time when the enemy started shooting. This is because I then check if this elapsed\_time variable is less than 2500 milliseconds, if it is I run the shoot method, if it is not I change the self.moving attribute to True. Finally at the end of this I subtract the scroll\_speed variable from the enemy, this is because the background map will be moving and if I stop the enemy’s position wherever its at, whether it’s a tank or soldier, it will seem like it is floating.

The next condition checks whether the enemy object is bound to stop 2 times to shoot (“if self.stop2 != None:”), if it is, the contents inside of it are similar to the previous block of code which executed instructions based on enemy stopping once.

#### Spawn\_Back Method

This method is continuously ran throughout the game and it checks whether the enemy has left the screen and has not been killed/destroyed yet.

A computer screen shot of a program code

Description automatically generated

Firstly I check If the enemy has left the screen, the enemy can do this by moving to the right side and leaving or by moving to the left side of the screen and leaving. If that’s the case and the enemy has left, I need some way to respawn the enemy somewhere so that it isn’t lost outside of the “screen” and can walk in front of the player again so that he or she gets another chance to shoot it. I do this by resetting the self.x and self.y to a random integer, for the self.x attribute I had to choose a value which would be appropriate depending on whether the enemy is moving from left to right or vice versa. For example, if the enemy is moving from left to right, I will spawn it at a negative x-position so that its x-position can increase to a positive value when it’s moving. I then randomly choose whether the enemy will stop once or twice by assigning self.stops to random.randint(1,2). I then reset self.stop1\_passed and self.stop2\_passed to False, this is because when this attribute is False I get the co-ordinates for the stops in the update method.

#### Update Method

This method is what is ran in my main.py file, I run all of the other methods in the class in this and it is almost like the brain of the enemy class, it reduces the lines of code in main.py file which makes my code more efficient, neater and easier to work with.

A screenshot of a computer program

Description automatically generated

I take in the game display as a parameter, within the function I declare the 4 variables: tanks\_left, tanks\_shot, soldiers\_killed and soldiers\_left as global. I then use a conditional statement to check whether the enemy’s health is less than or equal to 0 and if it is I then check to see what type of enemy it is (soldier or tank). Depending on whatever type it is, I increase the count for the appropriate enemy shot variable and decrease from the appropriate enemies left variable. I then use the self.kill() function which removes the sprite object from the sprite group which is initially declared in the main.py file. After this, I check to see how many times the enemy is scheduled to stop, depending on the value of stops, I randomly assign the stop attribute to a random integer, this integer corresponds to the x value that the enemy will stop at, I also reset self.stops to 0, this is so that this condition is not met again. After this I basically run the methods I previously explained. If the enemy is moving, I run the self.move method and pass in the game display as a parameter. Then I run both the self.stop\_move() method (I pass in the game display again into this) and I run the self.spawn\_back() method.

#### Change\_Health Method

I use this method to change the health of the enemy if it is hit by the player.



The method takes in the amount of damage that is being done to the enemy and then subtracts it from its self.health attribute.

As a result, I have modified my weapon class and added in the check\_collision method:  
A screen shot of a computer program

Description automatically generated

This method takes in the sprite group as a parameter, it then iterates through all of the sprite objects inside of that group and checks if there’s a collision between the sprite and bullet/grenade, the enemy’s health decreases. Because grenades are much more powerful than a single uzi bullet, I have increased the amount of damage it does but also when I check for the collision using the rectangle collision function, I check the collision between the explosion animation and enemies instead of the collision between the cursor and enemy. This results in a much larger area being checked for collision and as a result killing more enemies within the radius.

#### Enemy Counter

I have created an enemy counter which tracks the amount of tanks and soldiers left and then draws this onto the screen. This helps the player keep track of how many enemies he needs to eliminate in order to finish the game.

A screen shot of a computer program

Description automatically generated

I have two functions declared in my functions.py files, these are ran in my main.py file’s game loop. The show\_soldiers() function takes in as a parameter the soldier\_count, a photo of the soldier to be shown onto the screen, the x and y position of this photo. I draw the soldier onto the screen using the blit function, then I use a function which I made called display\_text() to display the soldier\_count variable. I do the same for the show\_tanks() function but with the relevant parameters. The finished product looks like this:  
A cartoon of a person and a tank

Description automatically generated

As the player shoots the enemies, the appropriate variables decrease and are updated to the screen.

#### Determining a Loss

To determine whether the player lost I check its health.

A black screen with white text

Description automatically generated

This section of code was added to my main loop, I use the python built-in function getattr() to get the value of an attribute, it takes in the object and attribute name as parameters. In this case I use it to check whether the player’s health is less than or equal to 0 and if it is I change the game state to “loss”, I am still to implement the logic for what happens when the player loses so this is only temporary, it was necessary to add it at this stage because the player’s health could reach 0 and nothing would happen.

#### Determining a Win

To determine whether the player has won I check if the amount of soldiers and tanks left is 0.

A computer screen shot of a black screen

Description automatically generated

I give first\_soldier the value of the first object in the soldiers group (if there is one), if there are no soldier objects then the variable no\_soldiers is set to true.

The next I statement checks if both soldiers and number of tanks left is 0, if it is, I change the game state to “win”. This will then lead onto a win screen which I will implement at a later stage.

#### Stage 5.4 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | How will the test be conducted | Outcome |
| 5.4.1 | Check that the enemy is visible and is moving randomly in a non-pattern way. |  | Visual test | The enemy was visible and moving, it was stopping randomly at various points to shoot at the player. |
| 5.4.2 | Check that the enemy is creating a visual effect when it shoots. |  | Visual test. | The enemy was creating a visual effect when it shot as the sprite flashed white whenever a bullet was fired. |
| 5.4.3 | Check that the enemy is doing damage to the player. |  | Print the player’s health on the console and visually check health bar. | The player’s health was printed onto the console and it was accurate as it only decreased when enemy was flashing. |
| 5.4.4 | Check HUD to check number of enemies left. |  | Visual test, check if the HUD for the amount of enemies changes when you shoot at the enemy. | The HUD did indeed change after I shot at the enemy, it decreased by 1, and the tanks also decreased by 1 when I destroyed them. |
| 5.4.5 | Check that enemy dies if you shoot at it. |  | Create a temporary test in the main loop to print “dead” in the console log if the enemy is killed. | The word “dead” was printed every time the enemy was killed/destroyed. |

#### Stage 5.4 Problems and Solutions

This stage has been the most difficult so far, it took the longest and it also had the most bugs out of any other stage.

##### Variable Scope Issue

I had some problems with the following variables: soldiers\_left, soldiers\_killed, tanks\_left and tanks\_shot variables. I wanted to be able to declare them in the settings.py file and to access/modify them inside a method of my enemy class in the enemy.py file. The first time I only declared the variables global in the settings file, however, this did not seem to do anything as the values in the variables did not change after I printed them in the main.py file. I then tried to explicitly declare the variables as global again in the method, this also did not do anything, however, it helped me in picking my next step as I then created 4 methods inside the enemy class which return values related to the enemy count.

A screen shot of a computer program

Description automatically generated

These methods can be called from main.py and will return their respective values depending on the function being called. This is very useful for when I display the HUD containing the amount of enemies left.

##### Problem with the soldiers count and end of game

One of the problems that occurred whilst trying to implement logic to check whether the player had won was that when the game stopped there were still soldiers on the screen despite the “enemies\_left” variable being equal to 0, to overcome this issue I had to change my max\_soldiers variable to match the number of enemies left when the player successfully eliminated 30 soldiers. Before I changed this, the soldiers sprite group kept getting filled with soldiers, there would always be 5 or 10 soldiers present. This meant that when all 40 enemies were killed, the soldiers from the sprite group were still drawn onto the screen, again, this was due to the 2 not being synced together.

A computer screen shot of a number

Description automatically generated

As you can see from above, I came up with a simple but effective solution after trying to debug this problem for hours. If the player has killed more than 10 but less than 29 soldiers, the maximum amount of soldiers that can be on the screen at any one time is 10, however when the player has killed more than 30 soldiers, the number of soldiers present on the screen is matched/synced with the number of soldiers left.

I also had another bug which rose from fixing the previous bug. When I tested the game it wouldn’t always stop when the number of enemies left reached 0, It would sometimes stop when the number of enemies left was “-1”. To overcome this issue I printed out some key variables in the console log and screen recorded some gameplay to see whether there was any pattern. I noticed that whenever I killed two or more soldiers at the same time, the enemies\_left variable would decrease normally, the sprites from the sprite group would also be removed normally, but, one new soldier object would be created, I realised that this happened simply due to the order in which my code executed. I was changing the value of max\_soldiers after I created new objects of the soldiers class, and because I only created new objects when the max\_soldier was greater than the current amount of soldiers in the sprite group, it meant that I was updating the values too late, so I moved the if statement from the picture above to be before the part where I added new objects.

### Stage 5.5 Creating the Power Ups Class

Power ups are objects in a game that players can “pick” up and they are designed to give a competitive advantage. In my game I decided to have 3 power ups, and they are as follows:

1. Health boost, if the player picks up this power up, his health increases.
2. Ammo, if the player picks up this power up, his ammunition count for his uzi gun increases.
3. Damage, when the player picks up this power up, the next x bullets in his uzi gun will do more damage than usual. E.g. if a tank takes 5 shots to get destroyed, it may only take 2 in this case.

I have come up with temporary values for how much the ammunition count, health and damage should increase by because when I start creating an algorithm to calculate the score of the player, I will need to make sure that the difficulty is not too hard but also not too easy.

#### Constructor Method

A screen shot of a computer program

Description automatically generated

The constructor method for the power ups class takes in 5 parameters which are:

* gplayer - the player object from the main.py file which contains information about the user playing the game.
* gweapon – weapon object specifically for the uzi gun.
* gx – the x coordinate of the crate containing the power ups. (A random integer between 800 and 1100 is passed in at the start).
* gy – the y coordinate of the crate containing the power ups. (-100 is passed in at the start so that when the crate spawns in outside of the screen).
* gopacity – the opacity of the visual effect which is seen after a power up is accepted of type “health” or type “ammo”.

There are also other attributes which are declared at the start of the class, and they are as follows:

* self.first, self.second, self.third – these attributes take in a random integer within a given limit, this integer corresponds to when the crate should be spawned with respect to the enemies left. E.g. if self.first is 34, the first crate will be spawned in when the enemies left are equal to 34.
* self.powerups\_list – this is a list which contains all of the attributes we have just explained in the bullet point above.
* self.crate – this attribute contains the image of the crate which is spawned in, it is initially declared in the settings.
* self.moving – this attribute is a flag which takes the boolean values true or false depending on whether the crate which contains the power up is moving or not.
* self.shot – this attribute takes in the boolean values true or false depending on whether the crate was shot by the player.
* self.message\_x, self.message\_y – these two attributes take in the x and y values of the message/visuals which appear after the player shoots the power up crate.
* self.message\_photo – this attribute takes in the image of the visual effect that appears after shooting the power up crate.
* self.show\_message – this attribute takes in the boolean value of true or false, this depends on whether the player has shot the crate and the visuals can be show or not.
* self.type – this attribute takes in either of 3 string values: “health”, “ammo” or “damage”. This depends on which power up was “picked up”. Initially this attribute is set to None.
* self.size – this attribute takes in the width and height of the visual effects image which come onto the screen after shooting the crate. It takes it in this form (width, height).

#### Update Method

The update method is the main “brain” of this class, it co-ordinates calls to all other methods and makes important decisions using if statements.

A screenshot of a computer program

Description automatically generated

The method takes in the following parameters:

* soldiers – this contains the number of soldiers left.
* gd – this variable contains the game display.
* x – this contains the x co-ordinate of the mouse cursor respective to the display.
* y – this contains the y co-ordinate of the mouse cursor respective to the display.
* left\_click – this contains a value of true or false depending on whether the player is left-clicking the mouse.

Firstly, I loop through all of the contents of self.powerups\_list, this will contain 3 random integers which represent when a power up should be “dropped”, the integer needs to match the number of enemies left in order for the power up to drop. I then check whether the number of soldiers left matches any of the random integers, I also check if there are any crates on the screen which were not shot using the self.moving attribute, if this is False, it means that there are no other crates on the screen. If the previous conditions are met, a random power up is chosen from the 3 options and is assigned to the self.type attribute, I also remove the random integer from the self.powerups\_list for consistency because I don’t want multiple powerups to spawn by accident. Finally in this part, self.moving is set to True, this is a simple flag which allows the next condition to be met.

If self.moving is True, self.shot is reset to false, this is because if it is not reset, then only the first crate can be spawned and any other crates are not spawned. After this, the following 3 methods are called: self.draw(), self.move() and self.check\_collision(). I will be explaining what each of these methods does after.

If self.show\_message is equal to True, this means that the player has shot the crate containing the power up. The contents of this if statement allow a visual effect/message to be shown to the user so that he knows what power up he has gotten. I firstly call the self.visual\_effect() method which I will explain in a later sub section.I then decrease the y value of the visual effect by 0.5 each cycle, this is so that the message goes up, I then decrease the opacity of the visual effect by 1.5, this is done so that it feels like the visual effect is fading away. I then have another if statement to check whether the opacity is less than or equal to 5, this is done because when the opacity is around that amount, the visual effect will almost be invisible, as a result, I change the self.show\_message attribute to False so that this condition is not met anymore as the message has disappeared.

#### Draw Method

The method only takes in the game display as a parameter. It copies the parameter gd onto the self.gd attribute. It checks if a power up actually exists and if the player has not yet shot the crate. If that is true, the crate image is drawn onto the x and y co ordinates denoted by self.x and self.y.A screen shot of a computer screen

Description automatically generated

#### Move Method

A screenshot of a computer program

Description automatically generatedThe move method does not have any parameters. It is mainly responsible for synchronizing the crate’s movement.

I firstly check if the y co ordinate of the crate is less than or equal to 350, if it is, the y co ordinate increases by 5 each cycle. This is done so that the crate can “drop” down onto the floor. I then check whether the y co ordinate of the crate is greater than or equal to 350, if it is, the x co ordinate of the crate moves with the same speed as the background scrolls, this is to add the illusion that the crate stays “still”. The final if statement checks if the x co ordinate is less than or equal to 100, this is done because the player may not have shot the crate and this possibility needs to be considered and processed. As a result, I reset the x co ordinate of the crate to be a random integer between 800 and 1100, I also reset the y co ordinate to be equal to -100. I finally reset the self.moving attribute to equal false, to allow new power ups to be chosen and spawned without restrictions.

#### Check\_Collision method

The check collision method checks whether the player has clicked on the crate. It has the following parameters:

* mouse\_x – the x co ordinate of the mouse cursor relative to the screen.
* mouse\_y - the y co ordinate of the mouse cursor relative to the screen.
* mouse\_left – this variable contains either the value true or false depending on whether the mouse’s left side is being clicked.

A computer screen with text

Description automatically generated

I firstly check whether the mouse’s left side is being clicked, if it is and if the crate has not been shot, I will check whether the mouse is over the crate in terms of its cursor position using two more if statements which compare the x and y position of the crates and the width and height of it. If the cursor is on “top” of the crate then the use\_power\_up method is run. I then reset some attributes such as self.shot, self.x, self.y and self.moving because this allows more power ups to be spawned in without restrictions.

#### Use\_power\_up Method

This method makes sure that the power up systemically works and gives the player more health, more ammo or more damage.

A screen shot of a computer program

Description automatically generated

##### Health

I check whether the power up type if equal to “health”, if it is then I use the change\_health() method which I wrote in the player class, it simply changes the health of the player, in this case it increases it by 50. (However this is just a guideline and so it may change later). I then play the slurp sound which is a sound designed to mimic a special potion being drank to increase health. After this, I give self.message\_x and self.message\_y their appropriate values (these two variables take in the x and y co ordinates of where the visual effect/message will appear after the crate is shot). I also change self.message\_photo to the appropriate visual effect to be shown. Self.show\_message is set to true as it is a flag which (like previously mentioned) allows the message/visual effect to appear and then fade away slowly. I set the size of the image using the self.size attribute and finally I set the opacity to be at 300.

##### Ammo

If the type of power up received is equal to “ammo” I call the change\_bullets() method from the weapon class. Here is what it looks like as it is something I implemented at this stage.



It simply changes the number of bullets of the weapon to be added by the amount parameter, in our case this is 20 bullets added on.

I then play the chaching sound which is supposed to be the sound from a cash register opening, in this case, the “money” is the amount of bullets.

The rest of the attributes declared in this subsection of “ammo” is the same as those declared in “health”.

##### Damage



If the type of power up is “damage”, then the more\_damage method is called from the weapon class.

The method works as follows:

A black background with white text

Description automatically generated

I take in the font colour of the number of bullets HUD and the new sound of the uzi gun. I firstly, change the self.more\_damage\_active attribute to true, this is a flag which meets another condition when updating the weapon objects. I change the self.font\_colour to equal font\_colour, this is to change the colour of the HUD which shows the number of bullets. This tells the player that the more damage power up is active. I also change self.sound to equal sound so that the gun sounds different whilst this power up is active. The way I can tell whether the power up is active or not is as follows:  
A screen shot of a computer program

Description automatically generated

If the self.more\_damage\_active is true, then the self.current\_bullets takes a copy of the bullets at the current moment for the uzi just before the power up is in place. The number of bullets increases by 15 and the damage for the uzi increases to 250, I also change self.more\_damage\_active to equal False as the condition has been met successfully. After this, I check whether the number of uzi bullets in the present moment is less than or equal to the copy of the current bullets we took earlier. If it is (as in the extra, more damage bullets the player got have run out), then the uzi’s damage is reset back to 100, font colour for the HUd returns to black and the sound of the gun becomes the same as before.

#### Visual\_effect method

This method displays the visual effect and makes sure that it fades away appropriately.

A screen shot of a computer program

Description automatically generated

Firstly, the variable temp contains a surface which is created of size equal to self.size, this attribute contains the width and height of the image message. The .convert() at the end is used to improve performance, this is done by converting the surface to the same pixel format as the display. The next step may be a little hard to visualise, basically, I create a copy of the game display contents which take up the area where I want to draw the visual effect. This is done by passing in the game display as a parameter that needs to be “cut”, the game display is drawn at -self.message\_x and -self.message\_y of the surface, this is a little trick which allows the correct area from the game display to be “cut” and displayed. After this I draw the visual effect/message image onto the surface temp at the (0,0) coordinates. I then change the opacity of the surface to self.opacity. Finally, I draw the temp surface using the blit function at the self.message\_x and self.message\_y coordinates.

#### Stage 5.5 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No | Test Description | Success  Criteria  Reference | Data and Type | Outcome |
| 5.5.1 | Check that the powerups are being displayed properly on the map and “flowing” smoothly. |  | Visual test. | I implemented a crate to store the power ups, this crate is being displayed properly as it falls down from the sky fairly smoothly and then once it reaches the floor, it starts moving at the same speed as the background, this is to make it feel like it is “still” on the floor, hence increasing smoothness. |
| 5.5.2 | Check that the power up can be shot at and is removed after. |  | Whilst in the level, wait for the power up to spawn and then shoot at it. | I can shoot the power up anytime, whether it’s in the sky or not. Once I shoot it, it disappears. |
| 5.5.3 | Check that the power up is having an effect systemically. |  | Create a temporary procedure in the main loop to print out the player’s health, ammo and damage. Do a visual test to see difference in value on HUD. | The power up is actually increasing health, ammo and damage. This is because, I manually selected all of the power ups and checked whether the ammo, health and damage were all increasing like they were supposed to and they were. I also looked at the HUD to check whether the health bar and amount of ammo are updating properly and they are. Finally, I also checked to see if the damage actually increased and it did because after picking up the more damage power up, it takes less bullets to shoot down a tank. |

#### Stage 5.5 Problems and Solutions

There weren’t many problems whilst I was adding the power ups, however these two problems slowed down progress as I spent some time debugging the root issue and having to think intensively about a feature.

##### Unable to see visual effects after shooting the 2nd or 3rd crate

The problem was as follows, imagine you are the player and you have just shot the first crate containing a power up, the visual effect comes up with information saying what you have just “picked up”. However, when you see the crate again and shoot it, you don’t see the visual effect, but you do still gain the benefits of the power up (e.g. your health increases on the HUD). To find the root cause of this, I manually followed through the program and noted down the values of the key variables being used. I realised the issue was very easy to fix.



The fix was to change the value of the opacity back to 300 just after the player shot the crate. This meant that although the message/visual effect was on the screen, the opacity was just too small for this to be visible.

##### Implementing a Method to spawn in and fade away the message after shooting a power up crate

I wanted to implement a small feature for the game where a message or visual effect spawns in after the player shoots the crate and picks up the power up. I then wanted this message to fade away. The hardest part for this was actually thinking of a way in which I could do this, it was conceptually hard and there was not much information on this online. I came across the .set\_alpha() pygame function which is used to change the opacity of a surface. However, I was not sure how I would use this in my game because I had a png image as my visual effect, this meant that any pixels from the surface which were not part of the png image would be automatically black at the start so the actual surface would not really be a png image, it would be a transparent image on top of a black rectangle which was not aesthetically pleasing. After some thought and research, I came up with one line of code which solved this problem.



Part of the game display was copied onto the surface at the same co ordinates as where the visual effect needed to be shown. Therefore, I drew the visual effect on top of this game display copy which was on top of a surface and faded it as required. As a result, it solved the problem.

## Prototype 3

### Analysis of Feedback from Stakeholders

I have asked my stakeholders to provide feedback to me for this prototype. Below I have summarised this feedback into positives and negatives:

Positives:

* Feels responsive
* Suitable difficulty
* The visual effects when the user picks up a power up are nice

Negatives:

* When the soldiers are shooting at you and you shoot back at them, they do not die.
* Tanks sometimes face the wrong way when they shoot at you when passing the screen.

### How did I fix this negative feedback?

The two negative pieces of feedback that I received both involved logical errors rather than syntax errors.

#### Unable to kill soldiers when the shoot you

The first issue I fixed was to do with the player being unable to kill the enemy when it was shooting at them. This was specifically only for soldiers not tanks.

The fix for this issue was actually relatively easy but took some time to trace what was happening. When the enemy stopped to shoot at the player, the self.rect.x and self.rect.y attributes were not being updated, this is what allowed the collision detection to work. Therefore I moved these from the move() method into the update() method of the enemy class.

#### Tanks face the wrong way when shooting

The second issue described by stakeholders was to do with the tanks facing the wrong way when they shot at me. (e.g. if they were moving from left to right, their orientation would change to right to left when shooting and return to normal when moving back again).

The issue causing this was that the orientation was changed inside the constructor method by using an if statement to check for direction. However, when more tank objects were created from the main loop, this did not happen. Therefore, I changed this by having if statements in the shoot method to check for direction and showing the relevant shooting animation depending on this. (This is show below)

A screen shot of a computer program

Description automatically generated

When fixing this issue, I accidentally came across another bug and that was that, if you did not shoot at the tank or the soldiers, they would eventually stop shooting at you and just drive/walk across the screen. The issue causing this was fairly simple, in my spawn\_back() method (which was used to spawn the enemies back if you did not kill them), I was not resetting the following variables:  
A screen shot of a computer code

Description automatically generated

Therefore, I updated that method to reset the above attributes to those values. If I didn’t, the if statements in the stop\_move wouldn’t have been triggered to allow the enemy to shoot the player.

## Stage 6 – Creating The Pause Menu

If the player wishes to pause the game for whatever reason, they can press space on they keyboard to pause the game, they can then resume by pressing enter and their progress will remain saved and the same as before. Here is what the screen looks like when the player pauses the game.

A screenshot of a video game

Description automatically generated

This Stage was split into two, I firstly had to calculate the score of the player, then, I needed to implement logic to display the pause menu.

### Calculating the player’s score

I created a new method in the player class to calculate the player’s score. This formula was adapted from what I originally had in the pseudocode algorithms previously. I give the player an initial score of 500, I then add 200 for each soldier killed and 500 for each tank shot down. I then add up the player’s health lost and uzi gun bullets used and remove them from this score. I subsequently remove a portion of the time which has elapsed since the game started. Doing this ensures that to get the highest score, the player has to use the least amount of bullets, maintain as much health as possible and finally kill all enemies in the smallest time.

### Pausable Timer Class

One of the features I needed to implement was a timer to check how long the player has played the game. However, I had to pause the timer if the player paused the game. Therefore, due to the complexity of this, I created a class to handle the game timer.

A screen shot of a computer program

Description automatically generated

The key attributes in this class are:

* self.start\_time: this attribute contains the time from when the game started.
* self.paused\_time: this attribute copies the value of self.start\_time when the game pauses to process how long the game has been running for up until the player paused.
* self.is\_paused: this attribute contains the value True or False depending on whether the game is paused or not.
* self.started: this attribute contains the value True or False depending on whether the game has started.

#### Start Method

This method is called from the main loop when the game starts, it triggers the self.start\_time attribute to start “counting”. And the self.started attribute is changed to True this is so that the method is not triggered again from the main loop.

#### Pause Method

When the player presses the spacebar, this method is called, it checks whether the self.is\_paused attribute is False, if it is, the self.paused\_time attribute takes a “snapshot” of the time the game has been running for by subtracting the self.start\_time from the current time by using pygame.time.get\_ticks(), it then changes self.is\_paused to True.

#### Resume Method

This method is called from the main loop when the player is currently in the pause menu and wants to resume gameplay. It firstly checks whether the attribute self.is\_paused is equal to True, if it is then self.start\_time is equal to the current time – self.start\_time, this is so that the attribute contains the time spent on the pause menu. Nextly, self.is\_paused is set to False

#### Get\_elapsed\_time method

This method returns the time spent playing the game. If the player is on the pause menu then it will return self.paused\_time, if the player is still playing, then it will return the difference between the current time since the game started and the times spent on the pause menu.

### Display\_pause\_menu function

I created a new function which is in my functions file to display the pause menu itself.

A screen shot of a computer code

Description automatically generated

It takes the current score of the player and high score as parameters. I firstly draw on the pause\_menu image which is a static photo which contains a navy blue background and writing. I then display the current score of the player alongside the high\_score.

### Stage 6 – Development Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test  No | Test Description | Success Criteria Reference | How will the test be conducted | Outcome |
| 6.1 | Check that the player can pause the level. |  | Press spacebar on the keyboard. | The pause menu is displayed when the player presses the spacebar. |
| 6.2 | Check that the game is not continuing in the background and that the time taken to complete the level is paused. |  | Create a procedure to print out all major values such as the time taken to complete level and health. This procedure should print out these values to console. | I printed out values from the following attributes: soldiers left, tanks left, player health, timer, number of bullets and grenades. All of these remain constant when the game is paused. |
| 6.3 | Check that the keyboard inputs work. |  | When on the pause menu, press enter to resume. Go to the pause menu again and press ESC. | After pressing enter from the pause menu, the game resumes as normal and all processes are done in the same way as before. When I ESC after pausing twice, I can successfully exit the game without any problems. |

### Stage 6 – Problems and Solutions

During this stage I did not encounter any major problems.

## Stage 7 – Creating the loss display

# D. Evaluation

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

# Project Appendixes

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

* Complete Code Listing (ESSENTIAL)
* Interview Transcripts
* Meeting notes
* Observation notes or questionnaires