Table of Contents

Analysis

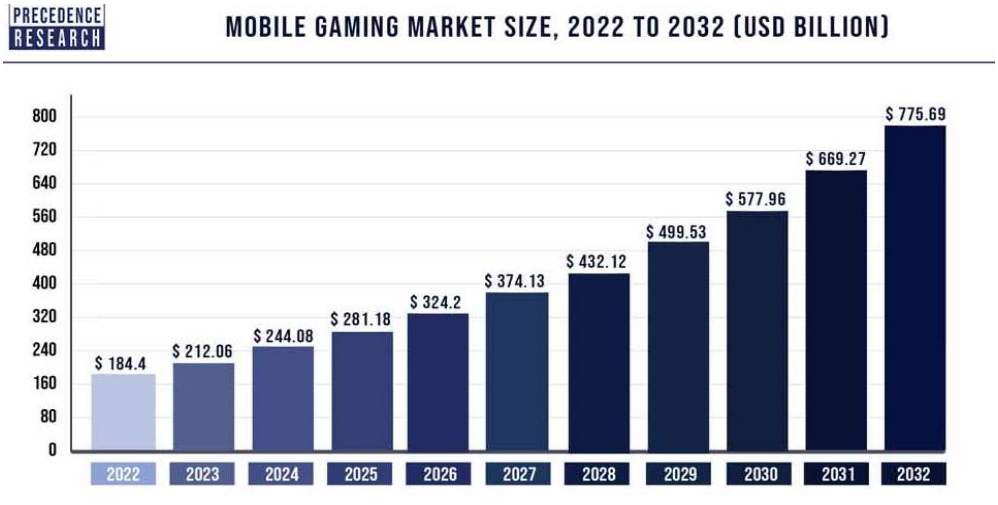
User End

My user end will consist of students who have a great passion for video games but also cannot afford them due to the unaffordable prices or have accessibility requirements which a lot of games do not provide support with. Many of these video games are aimed at people aged at a range of 6-18 all coming from diverse backgrounds. A lot of the times students cannot afford these games for a numerous of reasons for example, they might be too young to make an income of their own so they rely on their parents to buy it for them, and a lot of the times they might not be allowed because a wide range of reasons. They need a solution for this as students are the primary consumers of the gaming industry, and their feedback and preferences can significantly influence the development and success of video games.

Current Solutions

Console games are very popular as these games are very high quality, and most provide a levelling up system which encourages players to keep playing and grind out the game but many games at a lower price point provide a similar system. Right now, students may be paying prices exceeding $70 just for a game that has a one-year cycle, for example the EA FC and the Call of Duty Franchise. They also require expensive hardware to run. Consoles and PC's which are suited to running top end games can all cost a minimum of over $200, and modern-day displays can also cost an upwards of $100 even just for entry level quality.

A solution to this can be playing mobile games instead. The mobile gaming market is vast, offering a wide range of games across different genres and compatible on majority of phones. This ensures that all students might be able to find something they enjoy. It is also very portable meaning that if the game is offline, it can be played from practically anywhere. Many mobile games are free to download and play (to an extent), making them accessible to students who may not have the means to purchase expensive video games or hardware. All these reasons help make it a very fast-growing industry and it is projected to reach $775.69B by the end of 2032 shown in the graph below.



Source [www.precedenceresearch.com]

Although this might seem like the ideal alternative for solving my problem as they are mostly free to play and can be accessed on most phones. However, these games can be very frustrating to play for several reasons. First, although they might be free to play, these games derive all their revenue from microtransactions and Ads. There can be an absurd number of Ads in some games which can frustrate someone who may be trying to relax and enjoy themselves after a long day of work. It will also ruin the whole gaming experience which should be the main goal of every single game.

The In-App Purchases can range from granting you in game cosmetics such as different characters all the way to unlocking new levels in the game. This often leaves behind the players who have not paid stuck on the same few levels making the game repetitive and it leaves no reason to play it again especially if the game is not against real people. Many online mobile games use a pay-to-win model, where players who spend money have a significant advantage over those who do not. This can make it an unfair gaming experience for players who cannot afford to make in-app purchases.

Finally, the performance and quality of these games can sometimes be extremely poor and a lot of the times they drain the devices battery as when you are playing, the hardware of your phone is under a heavy load, and this results in the power consumption and temperature to increase.

Most computers can run browser games such as the ones on websites like FRIV.com. These games are all free to play and very simple to play. However, these games are very low quality, have a lack of depth, no progression saves, a big dependency on internet connection and have a lot of privacy and security risks as visiting untrusted sites to play these games can expose users to malware, phishing attempts and more.

Most browser game and mobile games lack accessibility features which can limit what students who cannot afford the more expensive games with accessibility features can play.

To counteract these problems, I will develop a game which will be optimized to run on most modern computers, we can reduce the need for high-end gaming consoles or PCs, making it accessible to a broader audience. It will also be free to play eliminating the upfront cost for players. This will make it more accessible to individuals who cannot afford expensive games or hardware. Also, the game will be stored on a downloadable file, so that it does not require an internet connection to play, and it will include many accessibilities features that will allow people with special requirements be able to comfortably play the game.

Computational Methods

A computational solution is superior in this context opposed to non-computer-based methods such as going outside and playing sports instead of gaming. First of all, it can be accessed from anywhere as long there is a computer whereas a physical game such as a sport, would require large open space which can be very hard to find in some areas and in many cases, children may not be allowed out on their own because maybe they're too young or the area they live in is too dangerous.

This brings me on to my second point which is that something virtual is significantly safer than an activity that is played outside in the real world. Many people live in areas with high crime rates and young children can be made very vulnerable as they are easy targets to kidnappings. A computer program can completely get rid of this risk if the developers put in the correct safety measures. Also, physical activities may not be accessible to everyone due to physical disabilities or restrictions, while a computer program can offer a high degree of adaptability, with the potential to be customized or modified to accommodate a wide range of individual needs and preferences.

The overall cost of creating a program compared to something physical such as a toy is significantly cheaper. This is largely because once the initial development of the software is completed, it can be distributed and replicated at virtually no cost. Also, updates and improvements can be made over time without the need for manufacturing new units or materials. This not only makes it a cost-effective solution for the developers and the user end, but also a more sustainable choice as the environmental impact associated with the production and disposal of physical goods will be significantly reduced as it will all be mostly digital.

In the context of my problem computational methods such as abstraction can be used for things like asset management. For my game, I can implement a centralised system that organises all the assets in a convenient way. For example, create folders for things such as sprites, backgrounds, sound effects within my project directory structure. This can be particularly useful when updating the game and say I want to replace a certain asset, I can quickly find it due to the good consistency and organisation.

Abstraction can also be used to provide the users a simple UI which only has essential information such as a health bar or a kill count allowing the underlying complexities of game mechanics and technical processes to be hidden. This helps the players to enjoy the game and not be overwhelmed by technical details. It will also attract more casual gamers and a younger audience as a complex UI can be hard to understand and might put players of from playing the game as it gives off an impression that the game might be too hard for them or just make it seem boring due to the overload of unnecessary information.

Decomposition can be used to help make the whole development process more manageable. You can break down the game design process into smaller components such as gameplay mechanics, level design, character design, and user interface design. This can help ensure each component gets each component’s requirements are met and can be ticked off a checklist allowing each aspect of the game receiving a fair amount of attention each. It also reduces the complexity of the problem allowing to prevent being overwhelmed and having a clear set end goal for each section of the game design. Time efficiency will also benefit from this as you can allocate a set time for each part of the project.

Investigation

My end user will primarily consist of students. I plan to communicate with students from my sixth form to gather user requirements and feedback throughout the development process. To find out what features would be admired by this group, I will carry out several surveys on my target audience, primarily from schools/colleges, so I can have an overview of the requirements on my project. I will be using google forms to carry out this survey as it:

- Provides instant access to answers making it more time efficient.
- Easy to use.
- Free to use.
- Mobile friendly meaning majority of students can answer remotely.
- Variety of question types ranging from multiple choice to worded problems.

The main things I intend to research is what would make the ideal game for most of the students. I say most as students, like any other demographic group, have a diverse preference in what would make the ideal game for them and there will be a lack of consensus if I try make it suited for everyone. This is why from my surveys I will always go with the option with the most votes as games that appeal to a broader audience have a higher chance of reaching a wider player base and more success.

I will ask a wide range of questions so I can find out about things such as student preferences, help refine a strong game concept, get ideas for my game and get feedback throughout the whole development process so I can make well-informed decisions so that I can create a high-quality game that meets the expectations and interests of my target audience.

First Survey

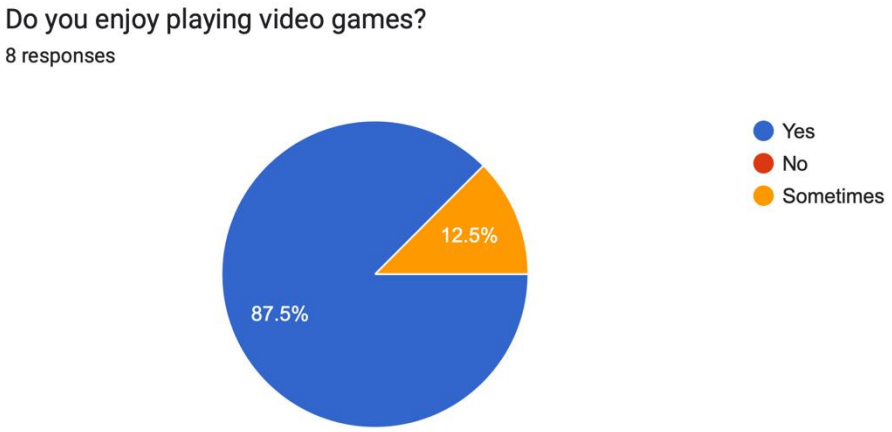
The goal of this survey is to gather the base information about the student's gaming habits, preferences and insights. This data will help serve as a basis of what type of a game I will create and help with further exploration and refinement.

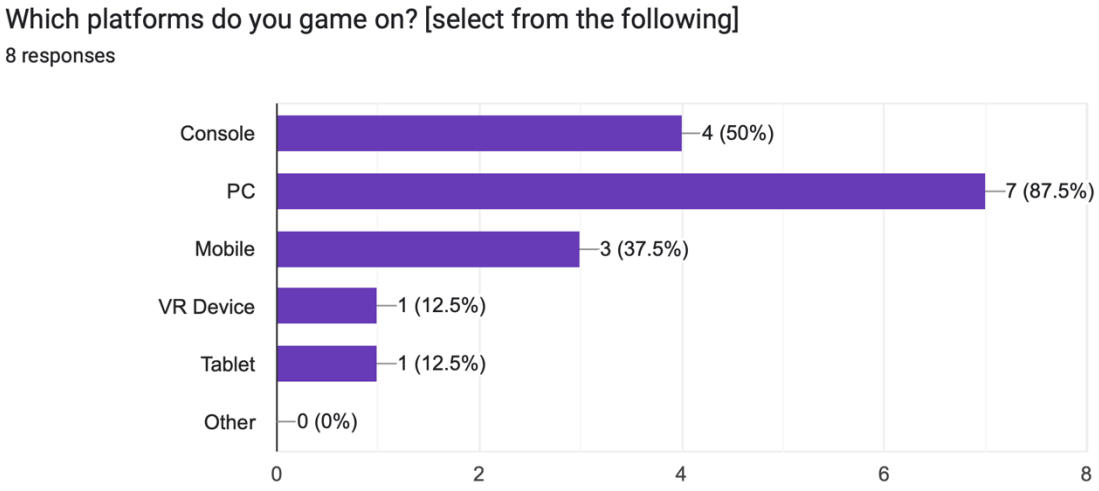
The questions I will ask in my first survey are:

- Do you enjoy playing video games?
  - Yes
  - No
  - Sometimes
- Which platforms do you game on? [select from the following]
  - Console
  - PC
  - Mobile
  - VR Device
  - Tablet
  - Other
- What is your favourite gaming genre?
  - Adventure
  - Battle Royale
  - Shooter
  - Role Playing
  - Sports
- Do you prefer single player or multiplayer games?
  - Single player
  - Multiplayer
  - Both
- What is your favourite thing about the game you enjoy the most?
- What is your LEAST favourite thing about the game you enjoy the most?
- How much do you spend on gaming on average per year?
  - £0
  - £1-£10
  - £10-£30
  - £30-£60
  - £60-£100
  - £100+
- Do you think video games are too expensive?
  - Yes
  - No

First Survey Results

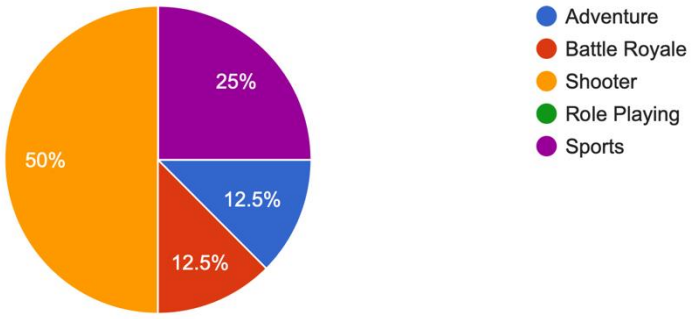
8 students from my sixth form have answered this survey and I plan to treat these students as my user group and plan to communicate with them throughout the development process with surveys and ask them for feedback. Their names are Aaron Traore, Ismail Siddique, Aaron Chifamba, Moses Spurrell, Oluwatosin Somide, Razvan Gheorghe, Kian Garrett and Charlie Searle.





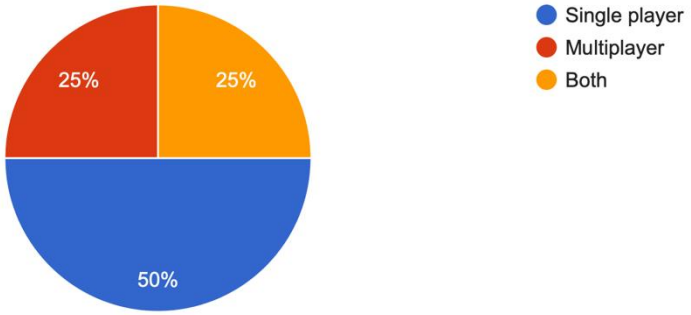
What is your favourite gaming genre?

8 responses



Do you prefer single player or multiplayer games?

8 responses



What is your favourite thing about the game you enjoy the most?

8 responses

I enjoy the art style of the game as it is very appealing

I enjoy the rewards gain in the games as they help give me a feeling of accomplishment.

I enjoy the enaging gameplay with constant action and no boring cutscenes

It is very intense at times so it keeps me engaged

I enjoy how realistic the graphics are

i like the games story

I like how i can personalise my character

it is very competitive

What is your LEAST favourite thing about the game you enjoy the most?

8 responses

I find that it gets boring and repetitive after a while of playing

I dont like the fact that the game series releases a new version of the game every year and leaves behind the older games making all the players go to the newer one leaving behind the old one with barely any players. those games leaving them behind

Once i have completed the games story, i have no desire to play it again as it is the same thing over and over again

It is too expensive for what it is

It is a pay to win game so even if i am more skilled than other players, they can still beat me

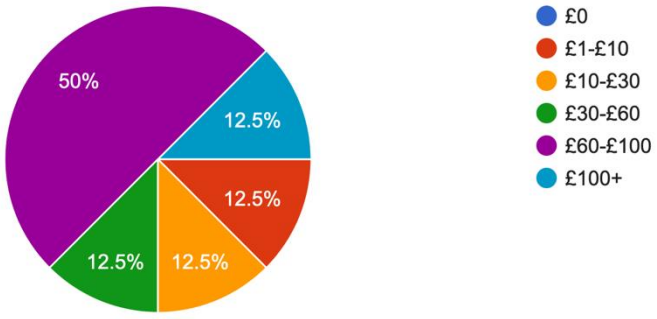
the controls are not customisable so when i switch between other games its hard to adapt to the controls of the other game

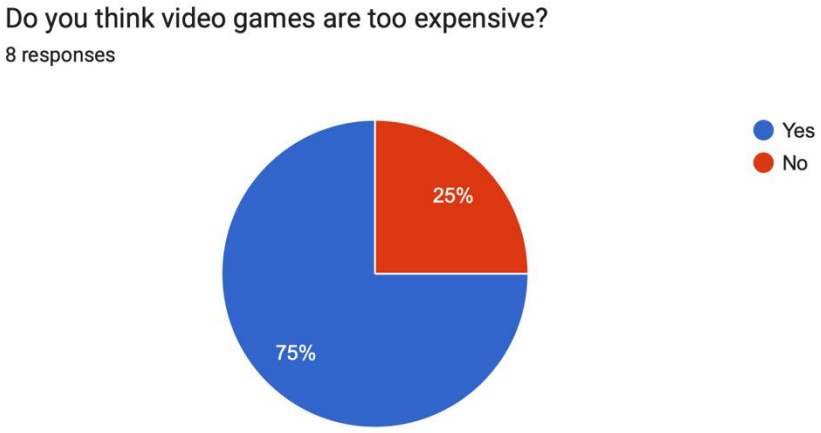
the UI is not very appealing

The servers lag very often

How much do you spend on gaming on average per year?

8 responses





First Survey Results Analysis

From the results I have gathered that most of the students enjoy gaming and 87.5% of the students that have participated in the survey game on a PC with only 50% gaming on a console. This is ideal for me as developing a game that can be played on PC will be easier for me rather than developing one for console. 50% of the students voted the shooter game genre as their favourite with 12.5% selecting battle royale genre which can fall into as a subgenre of shooter games. From this I have decided that my game will be a shooter game as it is favoured by over half of my user end, and I believe with the correct features it can also be enjoyed by those who picked another genre. Around 75% enjoy single player games so the game will not be online. This is beneficial as this means the game will not require an internet connection to play if downloaded. Features adored most by the students in other games are things such as: the art style, the graphics, personalisation, rewards, competitiveness, and engagement. A feature that will be great to implement to add more competitiveness will be a global leader board so players can compete and climb to the top. The only limitation here is graphics as implementing modern 3D graphics can be very difficult and it will require a lot of processing power meaning that students whose hardware does not have high specs will not be able to run the game. However, I believe an appealing game can still be made with 2D graphics if you use an appealing good art style. On the other hand, features disliked by the students are things like repetitiveness, overpriced, poor UI, pay to win and lack of customisation. I already had plans to make this game free to play and have no microtransactions in the game, these answers help reaffirm my decision. The final questions were about the cost of gaming and the answers support what I said in my problem definition.

Second Survey

The aim of this second survey is to dive deeper into specific aspects such as game design, mechanics, and themes. This will give me detailed insights on what to include in my game.

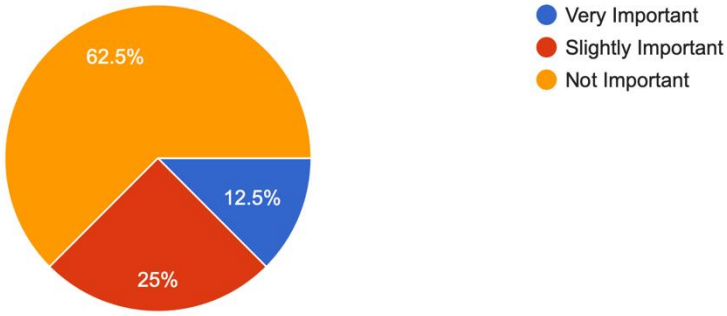
The questions I will ask in this survey are:

- How important is the story in a video game to you?
    - Very Important
    - Slightly Important
    - Not Important
- Which setting would you like to see in my game?
    - Futuristic
    - Ancient
    - Modern
    - Space
    - Retro
- Which one of these 2D art styles do you like the most?
    - Pixel Art
    - Cartoon
    - Vector Art
    - Hand Drawn
    - Watercolour
- Which shooter mechanics do you prefer in a shooter game?
    - Fast paced, run and gun action
    - Tactical/Stealth shooting
- What POV do you prefer?
    - First-person
    - Third person
- Would you like different levels in the game or make it endless where the difficulty increases over time?
    - Level based progression
    - Endless
- Are there any specific features you would like to see? [optional]

Second Survey Results

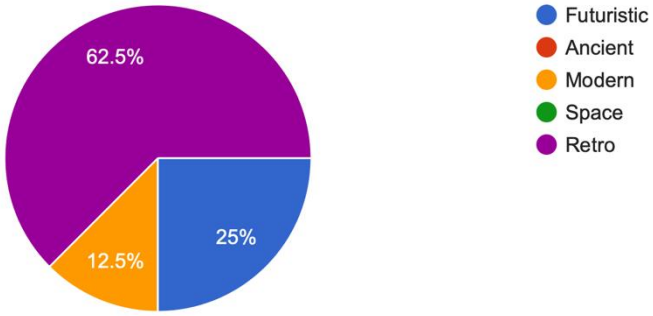
How important is the story in a video game to you?

8 responses



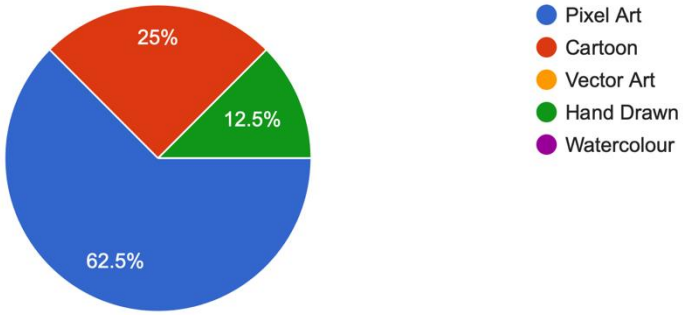
Which setting would you like to see in my game?

8 responses



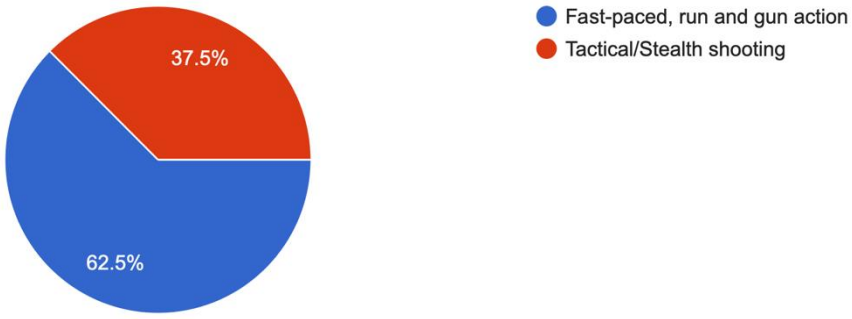
Which one of these 2D art styles do you like the most?

8 responses



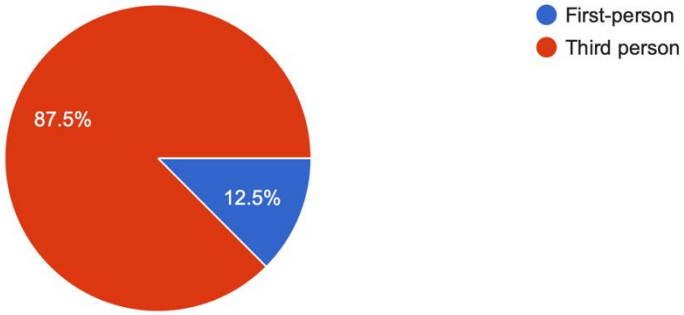
Which shooter mechanics do you prefer in a shooter game?

8 responses



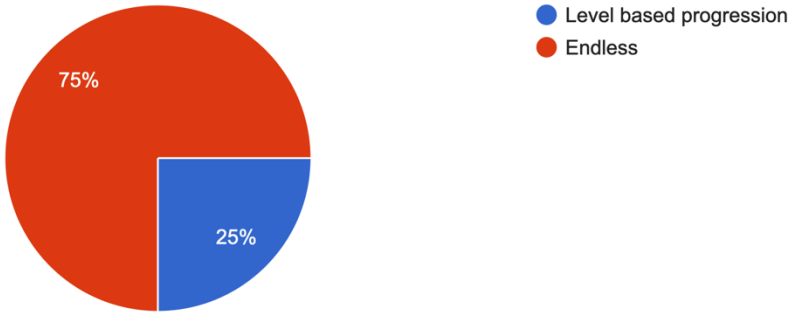
What POV do you prefer?

8 responses



Would you like different levels in the game or make it endless where the difficulty increases over time?

8 responses



Are there any specific features you would like to see?

5 responses

Guns with special abilities such as explosive ammo or something

ability to regenerate health

enemy health bars

Player stats

enemy variety

Second Survey Results Analysis

From these results I can infer that my user end is more focused on the gameplay rather than the story as only 12.5% find it very important. 62.5% of students votes for a retro setting while 25% voted for a futuristic theme. To satisfy both parties, I plan to use a retro futurism theme where I can maybe have a retro themed map with futuristic weapons. I believe this mix will help make my game more unique and entertaining while also getting the best of both worlds for my user end. The game will have a pixel art style as it had 87.5% of the votes. A fast-paced gameplay will be implemented rather than a more stealth approach and it will be from the third person perspective as most of the students prefer it. Reasons why most students voted for a fast-paced gameplay can be things such as more of an adrenaline rush, more engagement and easier accessibility for casual players. I believe this will benefit the game very well. It will also be endless, and the difficulty will increase the longer you are alive as 75% of the students voted for it. Specific features asked for are things such as player stats where a user can view things such as their high score, number of kills and more. A variety in enemies was also suggested and it was requested for them to have a health bar as that can allow players to play more strategically and not just shoot mindlessly at the enemies.

Summary

From all the results I have concluded that to meet the user end requirements I will create a single player third person 2D shooter game that will have a retro futurism theme with pixel art style graphics. The game will have fast paced gameplay with features such as power ups to help avoid repetitive gameplay. It will also include a global leader board and a player stats page so players can compare against their friends as this will help add a competitive aspect. There will be no levels in the game and instead it will be endless with increasing difficulty the longer you are alive.

Solution Research

The programming language I will use to create the game will be python. This is for reasons such as its ease of learning, rich libraries, community support and rapid development which can be useful for game design iterations and testing new ideas quickly. I have decided to use the python library pygame due to its simplicity as it gets rid of many of the complex details involved in game development making it easy to use while also having a large community for support and resources. It is also cross-platform meaning that the game can run on other operating systems with little to no modifications making it more accessible for my user end and other students from around the world. My chosen IDE is PyCharm as it is the one I am most familiar with, and it has features such as syntax highlighting which helps code to become more readable and easier to spot errors and typos. Code navigation is also exceptionally good as you can explore the structure of your code making it easier to manage and understand large projects such as gave development.

Hardware Requirements

These are the minimum requirements to run a game on python:

- 2GB of free RAM minimum
- 1.5GB of free disk space minimum
- Intel Pentium 4 or AMD Athlon 64 minimum
- Integrated graphics that support OpenGL 2.0 or higher
- Display with a resolution of at least 1024x768 pixels
- Keyboard and mouse
- Speakers or headphones

Software Requirements

The minimum operating system required for this game will be Windows 7 or later, macOS 10.10 (Yosemite) or later, Linux with a kernel version that supports Python 3.7 or later. Python 3.7 is required as a minimum and pygame 1.96 or later is needed. Most modern-day budget computers meet or exceed these software and hardware requirements which is ideal for my user end as the whole goal of this project is to create an affordable and enjoyable game that can be played comfortably by a wide range of people regardless of their background.

Libraries

For the program the following libraries need to be installed:

- Pygame: Provides many modules for handling game features such as graphics (drawing sprites), sound, inputs and game loops.
- SYS: Exits the game with sys.exit() function.
- Math: Used for calculations such as angles, distances, and trigonometric operations.
- Random: Provides functions to generate random numbers.
- Datetime: Tracks date and time.
- Time: Time-related functions such as delays.
- Hashlib: Creates hash values using algorithms like SHA-256.
- Json: Handles reading and writing JSON data.
- OS: Allows file manipulation

Success Criteria

To successfully solve the problem, I need to create a game that includes everything listed below:

- High Performance: The game must be compatible with PCs ranging from the low to mid-range, with a minimum of 2GB of RAM, 1.5GB of disk space, and integrated graphics that support OpenGL 2.0 or higher. It should run smoothly without noticeable lag or crashes on these devices.

- Engaging and Enjoyable Gameplay: The game should be fast-paced and engaging so that it is more enjoyable. The mechanics must be simple but still require skill, ensuring that there is a room for players to improve. The game should also gradually get harder the longer you play. A global leader board should track high scores, allowing players to measure their progress against others.

- Offline Playability: The game must be downloadable and playable offline, allowing for players to enjoy the game wherever they are without the need for constant connectivity.

- Appealing Art Style and Theme: The game must have a 2D pixel art graphic style with futuristic assets as it will create a retro-futurism theme that appeals to a broad audience. The visuals should be clear and visually appealing, even on low-spec devices, without affecting performance.

- Player Customization and Power-ups: The game should provide player customization options, such as different character skins and have power-ups to keep the gameplay dynamic and prevent repetition.

- Working Enemy AI: The game should have an enemy AI which gradually gets harder during gameplay. Their stats such as speed, strength and health would be increased the longer the player is alive. Also, the enemy should have 2 states where in one state it is not attacking anybody and is idle and in the other where it is following the player and shooting them.

- Progression System: The game should encourage players to play again so they can improve their player stats. This will encourage players to play the game over and over again.

- Intuitive User Interface: The game must have a simple and intuitive UI that is easy for players to navigate through. The buttons should be easy to see, and their purpose should be self-explanatory.

- Accessibility: The game must allow players to fully customise every control through key bindings in a settings menu. This will ensure that individuals with physical disabilities or different keyboard layouts can adjust controls to suit their preferences, making the game more inclusive. Also, a dedicated colourblind mode should be included to help players with visual impairments see the game more easily.

Design

Design Overview

To start off, I am first going to break down my game into sections to get a clearer overview of what needs to be done.

Section A - **Gameplay Mechanics**

- **Controls**: Add character controls for movement, interacting with items and shooting.
- **Shooting mechanics**: Implement a shooting system that allows the player to shoot.
- **Enemy spawning**: Add code to spawn enemies.
- **Enemy AI**: Create an AI which controls the enemy and carries out tasks such as movement towards the player, attacking them, have an idle mode and take away health from them.
- **Power-ups:** Have a variety of power ups that spawn throughout the game boosting different player stats or regenerating health.

Section B - **UI Components**

- **Main menu**: Create buttons for starting the game, viewing the leaderboard, and changing settings.
- **Settings menu**: Add options for customizing controls, adjusting audio and colourblind modes.
- **Game over screen**: Display players stats and inform they have died.
- **Stats menu**: Player can view their game stats such as all-time kills, high score and a global leaderboard.
- **Pause screen:** Player can pause the game at anytime and it will freeze everything while displaying them the option to mute, restart or quit the game.

Section C - **Leaderboard**

- **Scoring system**: Track score based on points earned.
- **Global leaderboard**: Use a file to track scores and display on leaderboard.

Section D – **Log in Page**

- **Log In Menu:** A screen where players can either sign up or log in.
- **Secured Data Base:** The password they used will be hashed before it is stored.

Section E - **Art Style and Theme**

- Use pixel art for all assets (player, enemies, background).
- Use a retro-futurism theme by combining old-school elements with futuristic details in the art and sound.

Section A

The default controls for movement will be the ‘W, A, S, D’ keys, the ‘R’ key for reload. This will be familiar for most gamers as the majority of keyboard and mouse games use this layout. For aiming I will be using the pointer arrow of the mouse, with the left click button being shoot. I decided to use the mouse pointer for aiming as then for the game I can use a top down 2D angle rather than a classic side on view which restricts the movement of the character to only forwards and backwards and is very generic as it is used in a lot of 2D games already. The top-down view creates a 3D illusion and makes the game require more skill as the players direction can move a full 360 degrees meaning that aiming the weapon will require more skill and enemies can come from all directions. I will add these controls using Pygame’s pygame.key and pygame.mouse modules for keyboard inputs and mouse events.

Enemies will spawn randomly with the frequency of them spawning in, their speed, health and attack damage increasing the longer the player is alive. They will all have a health bar and be controlled by AI which attack the player when in range. The AI is crucial to provide a challenge for the player. For the AI system I will use the pygame.sprite class to manage enemies. The AI enemies will calculate the distance between the player and enemies and move towards the player using vector math with the math.atan2 function. As the time goes on, the AI will increase its stats to provide more challenge for the player. All enemies will be stored in a sprite group so that making updates to them is easier.

The power ups in the game will also spawn randomly and there will be 3 main power ups. One for gaining back a portion of your health, one to boost speed temporarily and one to boost strength. These power ups will help the player a lot especially in the crucial stages where the enemies start to become very difficult. The power ups spawning in random locations on the map will encourage the player to move around and not just stay in the same place the whole time. The unique ability boosts will also help the player change their playstyle meaning less repetitive gameplay. To implement these into the game I will use pygames collide.rect function which will grant the player the powers if the player rect collides with the power up rect.

The pause screen will be accessed using the ‘P’ key as the game uses the mouse to aim so it would ruin the players aim if they had to click a physical button to pause the game rather than just pressing a key and carrying on from where they left off. The pause screen will have 3 options: Mute, Restart and Quit. To un-pause the game the player must press the ‘P’ key. The pause button will work by using flags and if the pause flag is set to true, the game loop will lop through the pause function and if the pause is set to false, the game loop will loop through the game logic.

Below is an example of what the movement code may look like.

IF left movement key is pressed:

Decrease horizontal speed by set speed

IF right movement key is pressed:

Increase horizontal speed by set speed

IF up movement key is pressed:

Decrease vertical speed by set speed

IF down movement key is pressed:

Increase vertical speed by set speed

An example of an enemy spawning function:

FUNCTION spawn_enemy():

WAIT random_interval

SELECT enemy_type

SPAWN enemy at random_location

INCREASE enemy_stats_over_time()

There will be different types of enemies with different stats to make fighting each enemy require a different tactic. For example, one enemy may shoot very slow but deal a lot of damage so you best approach is to dodge the bullets then kill the enemy. All enemies will inherit from a parent class and below is an example on how I will implement this.

Class Enemy:

Self.speed = 5

Self.strength = 10

Self.firerate = 1000

Def attack_player():

#CODE TO ATTACK PLAYER

Class Enemy_2(Enemy):

Self.strength = 20

Self.firerate = 2000

 # INHERITS ALL THE OTHER CHARACTERISTICS

An example of collecting power ups:

Health_boost = pygame.rect(Image)

Def apply_powerup():

If player.collide.rect(Health_boost.rect)

Player.health += 20

Section B

The main menu will have 3 buttons; Start Game, Stats, and Settings. It will have a simple layout which will be intuitive for players and be easy to access, enhancing user experience by providing clear paths to important game functions. It will have retro-futurism style graphics made with pixel art as this was requested from the students in the survey. I will create all these graphics using Aseprite or Canva and will use them as Pygame surface objects when I import them in. Button clicks will be detected using mouse events.

The settings menu will allow players to customize key bindings, turn on a colourblind mode, toggle music and sound effects. I will include three colourblind modes in my game: Deuteranopia, Protanopia and Tritanopia. These options will help the game become more inclusive as they provide accessibility for players with different preferences and needs.

After the player loses, a game over screen will be displayed. This display will show survival time and the player's score. Showing the stats straight after the player dies is very important as it give the player a sense of accomplishment and encourages them to play again so that they can beat their previous score. This will be displayed the same way as the other UI components, but it will use pygame.font to render the text. When this screen is displayed, all enemies in the game will be killed to enhance performance for users.

The stats menu will consist off the leaderboard, players high score and their all-time kills. This will update after every game and when a player logs in, their individual stats will be shown.

All menu's will be set as functions and to change between menus, a game state will be used to identify what state the game is in and what state the game needs to change to. Below is an example of what the code could look like.

MAIN_MENU = "main_menu"

SIGN_IN = "sign_in"

GAME = "game"

SETTINGS = "settings"

STATS = "stats"

KEYBINDING = "keybinding"

current_state = SIGN_IN

Game loop()

if current_state is SIGN_IN:

Call SigninPage function

if current_state is MAIN_MENU:

Call MainMenu function

if current_state is STATS:

Call StatsMenu function

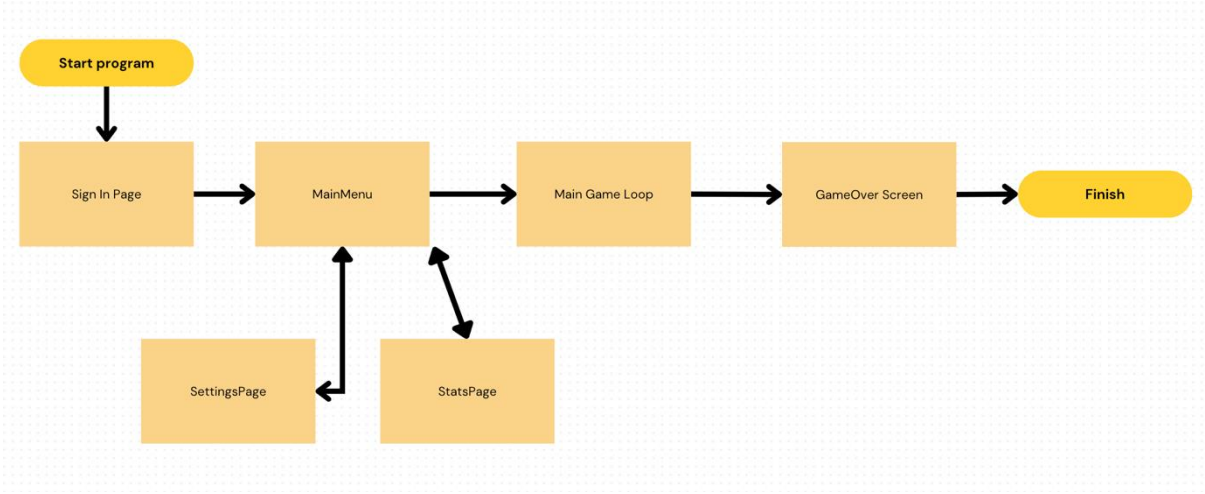
if current_state is SETTINGS:

Call SettingsMenu function

if current_state is KEYBINDING:

Call KeybindsMenu function

The flowchart below shows how each menu will be linked with each other.



The pause screen will be accessed using the ‘P’ key as the game uses the mouse to aim so it would ruin the players aim if they had to click a physical button to pause the game rather than just pressing a key and carrying on from where they left off. The pause screen will have 3 options: Mute, Restart and Quit. To un-pause the game the player must press the ‘P’ key. The pause button will work by using flags and if

the pause flag is set to true, the game loop will lop through the pause function and if the pause is set to false, the game loop will loop through the game logic. For example:

Game_loop():

If pause is True:

pause_function()

Elif pause not true:

 # Runs game logic

Section C

The scoring system will calculate score based on the number of enemies defeated and the total time survived. For each enemy killed there will be a set number of points granted and for each minute survived points will be awarded. This encourages players to perform better, compete with friends, and improve their skills. As the game is an endless survival game, this will help give a reason and motivate players to stay alive for as long as possible with the most kills so that they can climb up the global leaderboard. Pygame's pygame.time module will track the survival time.

I will use a JSON file to store each players score after they die and place their best score on the leaderboard. I am using a JSON file as it has simple syntax and is easy to implement into pygame. It is also very lightweight with minimal overhead allowing for more efficiency which will help with the game's performance. Player scores will be read from the JSON file and displayed in-game using pygame.font. This will be stored locally within the computer therefore internet access will not be required to view the leader board.

Below is an example of player score and kills increasing.

If enemy is dead:

Increase player score by 10

Increase player kills by 1

And an example for enemies score being saved into a file.

OPEN file "scores.json" in write mode

WRITE player score into the file

CLOSE the file

Section D

When the game starts, the user will be prompted to sign in or log in. If they enter their details incorrectly an error message will show. If they create an account, their details will be stored into a JSON file and the password will be stored in its hashed for to prevent anyone from accessing it. This will be done by using the Hashlib module in python and I will use SHA256 to hash the passwords.

Below is an example of the hashing function.

FUNCTION hash_password(password):

ENCODE password to bytes

HASH the password using SHA-256

RETURN the hashed password as a hex string

When the user enters their password during account creation this function will be applied to it before it is stored into a file. Here is an example of this happening.

USER_PASWORD = Enter password

hashed_password = hash_password(USER password)

STORE hashed_password

To verify password when logging in, the hash function can be aplied to the entered password and the hashed value can then be compared. For example:

USER enters password

hashed_input_password = hash_password(USER password)

RETRIEVE stored_hashed_password

IF hashed_input_password is equal to stored_hashed_password:

AUTHENTICATE the user

DISPLAY success message

ELSE:

DISPLAY error message

Section E

All game assets will be made from pixel art with a retro-futuristic aesthetic. The art will feature bright neon colours with dark backgrounds to emphasize a futuristic tone. Pixel art is consistent with the retro theme, creating a nostalgic feel for older games while adding it with a futuristic aesthetic to attract to modern players. The choice of neon colours will help convey the sci-fi setting and make the game look more eye catching. I will get all my assets from itch.io as they have a wide range of free to use assets. These assets will be stored in a separate file from the main game logic for organisation purposes. They will be loaded by

pygame.image.load(File path of image)

I will use 8-bit music and sound effects as it matches the retro-futurism theme, with sounds for shooting, enemy attacks, and player actions. 8-bit music will keep the fast-paced gameplay engaging and all the music and effects can be muted in the settings menu if they are not wanted by the user. I will find copyright free 8-bit tracks and sound libraries online on platforms such as YouTube or GitHub and use pygame.mixer to handle sound playback. Below is an example of how the sounds can be played in the game.

```
#ASSETS FILE

GUN_SOUND = pygame.mixer(Audio file path)

#MAIN GAME FILE

IF SHOOT GUN IS TRUE:

        GUN_SOUND.play()
```

The player design I have decided to use for this game is this

The character has a futuristic design, but the pixel art helps make it look retro and have an arcade game theme.

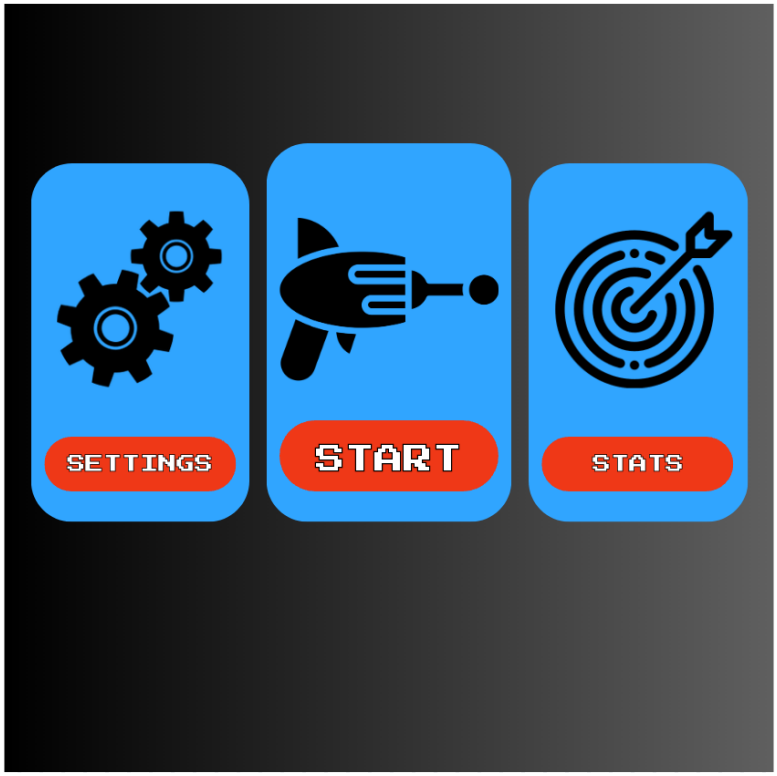
For the enemies I have chosen these:

And for the power ups I have chosen:

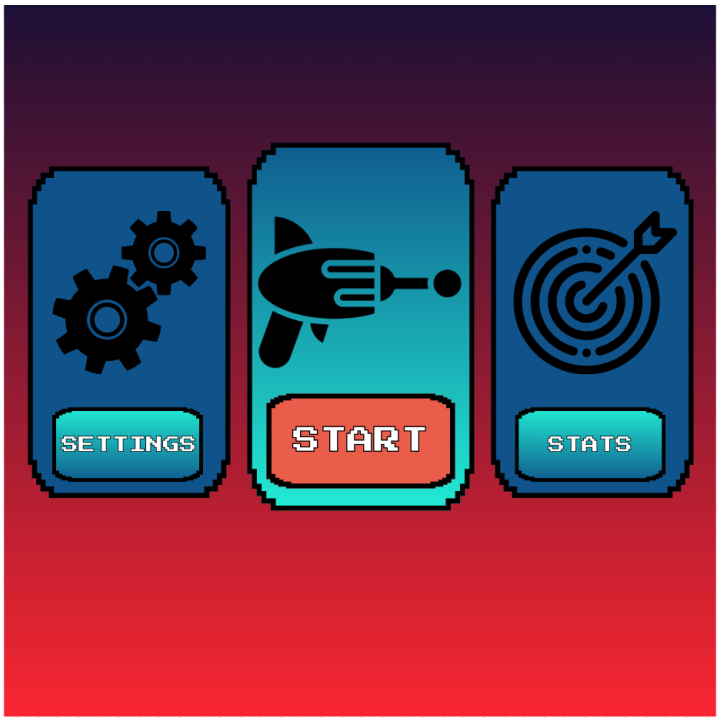
All assets above were free to download on itch.io.

UI Mock Ups

Below there will be an image of a main menu screen mock up that I have designed on Canva.



This menu design shows a neat 3 button menu layout with large icons that are easy to see and correspond to the function of the button, however it lacks the pixel art design, and the black gradient background is extremely dull. The whole colour scheme looks unappealing right now so in my next mock up I will aim to fix this issue.



This new design uses more pixelated shapes and has a new colour theme which I believe looks more appealing than the one before. The gradients help add more personality to the colours and having the start button be big and red helps it stand out from the other buttons making it more tempting to press. However, this design still has a few problems. First, the rest of the screen ignoring the buttons looks empty as it is just a plain gradient. Secondly, the button icons do not match the pixilated theme, so they look very off. Finally, I have realised the game doesn't have a title yet so when I get one, I can put it at the top of the screen to make it look more complete. I will ask the students that I had surveyed for their ideas on what the title can be but for the time being I will call it 'Shooter Game' so that I can complete the UI design. In my next mock up I will aim to add more detail in the background and change the icons on top of each button so that they match the theme.

This is my final mock-up of what the home screen will look like. This design includes details in the background such as a city being attacked by asteroids with a futuristic portal opening in the sky. The whole UI now is made of pixel art and follows the main retro futurism theme. All icons and backgrounds details that I added for this mock-up are premade pixel graphics that are available to use for free on Canva. The settings and stats menu will look very similar to this with the same background and just different buttons. I will send off this design to the students, asking them for their feedback and for ideas for the actual name of the game. This questionnaire will be made on Google Forms. (Form)

Feedback

Below I will attach screenshots of the questions I have sent to the students for their feedback for the main menu design.

Main Menu Design

On a scale of 1 to 10 rate this design *

	1	2	3	4	5	6	7	8	9	10	
Bad	○	○	○	○	○	○	○	○	○	○	Great

What do you like about this design?

Your answer

What do you not like about this design?

Your answer

Do you have any ideas on what to name this game?

Your answer

Submit Clear form

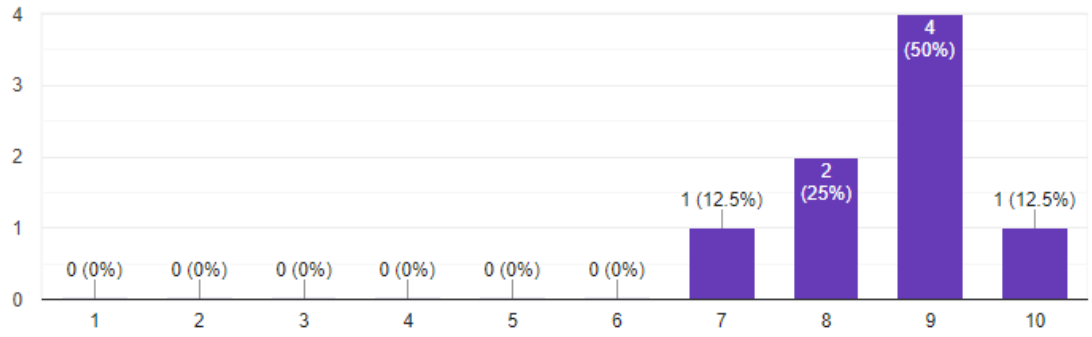
Below this I will attach the answers that they have submitted.

https://forms.gle/CCYmXNDsLRBG8RF37

On a scale of 1 to 10 rate this design

Copy

8 responses



What do you like about this design?

8 responses

I really like the background design and the whole overall colour scheme

The pixelated UI buttons

The colours used go well together

the city skyline below helps the background not look plain

The start button is bigger than the rest of the buttons and a different colour which helps it stand out from the other buttons

it has a lot going on making it look interesting and not boring and plain

The pixel font and theme is very nice

the colour gradients help add more personality to the whole design

What do you not like about this design?

8 responses

the icon used for start game is a controller when the game is meant to be keyboard and mouse. It should be changed
The red planet top left looks out of place
The controller icon looks out of place
there is no problem with it
The controller icon is not very fitting for a keyboard and mouse game
the controller in the middle should be changed for something else
N/A
The icon in the middle should be swapped for something that matches the theme of the game.

Do you have any ideas on what to name this game?

8 responses

Atomic Assault
Velocity Strike
Cosmic Carnage
blaster strike
Photon Blast
ChronoStrike: 2089
Radium Raiders
Bullet Storm

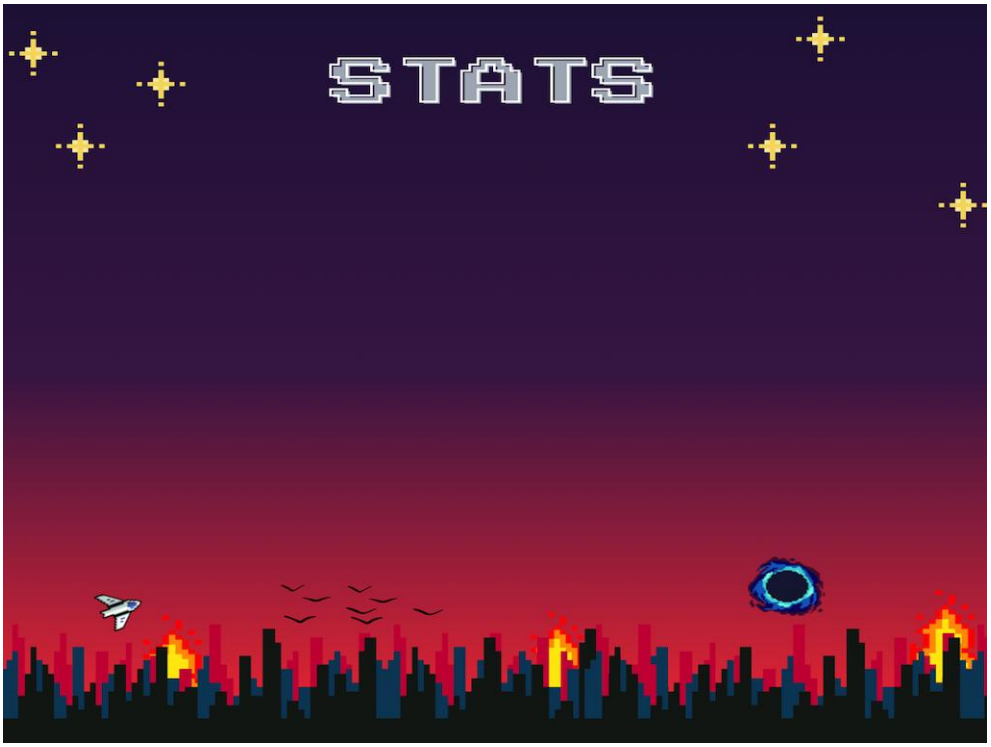
<u>Feedback Analysis</u>

From the feedback that I have received from the students I can tell that everyone likes the overall design and has no problem with it. Quite a few people have praised the colour scheme as well saying things like ‘The colours used go well together’. However there has been a lot of critics on the controller icon placed

above the start button with people saying it looks ‘out of place’ and it needs to be replaced with something else more fitting. Also, the red planet top left will be removed in my next design as I have had some negative feedback about that as well. The name that appealed to me the most was ‘Bullet Storm’ so that will be the final title for the game. Apart from that there will not be anything else changing. This main menu will provide all 3 essential buttons needed for menu.

For the settings and stats page I will use the same image but with different title text and draw on the buttons myself using pygames window.blit.


SETTINGS


STATS

Data Structures

I will now create a table to give a brief overview of what data strctures i will be using for each component in the game. This will help provide guidance while programming the game on what data structures will be most appropriate to use for all components.

Component	Data Structure	Function	Justification
Player and Enemy Objects	pygame.sprite.Sprite	Represents player and enemy objects in game and stores their attributes	Makes it easier to manage player or the enemies, improving game scalability and readability.
Bullet Management	pygame.sprite.Group	Stores all bullets fired by the player and enemies.	Manages and updates all projectiles in one loop.
Menu Options	List	Stores the menu buttons "Start Game," "Settings," and "Leaderboard."	Lists are simple to use for iterating over menu items and detecting user input.
Player Scores	Dictionary	Stores player scores with the players name as the key and their score as the value.	Fast lookups and updates making it be effective for managing scores during the game.
Game Assets	Pygame.image	Handles and renders sprites for players, enemies, and backgrounds.	Allows smooth rendering of sprites which will help with the performance
Global Leaderboard	JSON file	Stores player scores in JSON file	Allows leaderboard to be viewed offline
User Information	JSON file	Stores user information in JSON file	

Beta Test Plan

Below I will attach the questions I have created for me and the students and will be answered by us at the end of the program. The questions will be short, concise and go over usability, difficulty, gameplay mechanics, and our overall experience with the game.

1. How easy was it to understand the controls?

- A. Very easy
- B. Somewhat easy
- C. Difficult
- D. Very difficult

This will help identify whether additional tutorials or adjustments are needed.

2. How responsive did the controls feel during gameplay?

- A. Very responsive
- B. Somewhat responsive
- C. Unresponsive at times
- D. Very unresponsive

This checks if the controls responsive, which is critical for a shooter game and the whole overall experience.

3. How would you rate the game's difficulty?

- A. Too easy
- B. Easy
- C. Balanced
- D. A bit hard
- E. Too difficult

This will check if the game's difficulty is well-balanced for the target audience. If players find it too easy or too difficult, adjustments may be required.

4. Did you encounter any bugs or technical issues while playing?

- A. No issues at all
- B. Minor bugs, but nothing major

- C. Several bugs
- D. The game crashed or had major issues

This will help identify any bugs or glitches in the games.

5. How engaging was the gameplay?

- A. Very engaging
- B. Somewhat engaging
- C. Average
- D. Boring

This will give an idea if the game is enjoyable and worth playing.

6. Did the game run smoothly on your device?

- A. Yes, very smoothly
- B. Mostly smooth with occasional lag
- C. It was a bit laggy
- D. It lagged a lot or crashed

One of the main goals of this game is to ensure that it can be accessed by all students regardless of what computer they have so if it is not running smoothly in some then i can make adjustments to improve the performance

7. How would you rate the game's visual design?

- A. Excellent
- B. Good
- C. Average
- D. Poor

This will gather feedback on the game's visual appeal, ensuring the design is appealing for the players.

8. Did the game progressively get harder the longer you were alive for?

- A. Yes
- B. Somewhat
- C. No
- D. Not sure

This will inform if the enemy AI is getting noticeably harder the longer you play.

9. What needs to improve?

This will give a good idea on what the games weaknesses are and what to improve on.

10. Name the bugs you encountered while playing, if found any.

This will let me know what bugs are in the game so that I can easily find and fix them.

Test Table

The test table below will be used for iterative development for each algorithm.

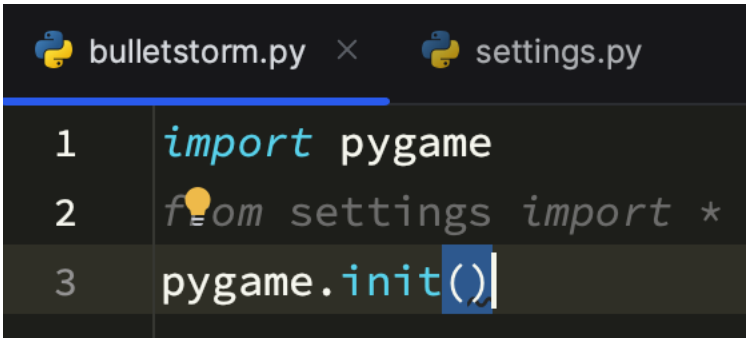
Algorithm	Intended Test	Test Data	Expected Results	Justification
Player Movement	Test if the player moves when keys are pressed	W, A, S, D inputs for movement in their respective directions	Player moves forwards, backwards, left or right depending on input	To make sure the player can navigate correctly
Shooting	Test if bullet is fired in the direction of the mouse pointer when mouse is clicked	Left mouse button clicked with pointer in different directions	Bullet is fired towards the mouse pointer	To confirm that player can aim properly, which is an essential game mechanic
Enemy Attack	Test if enemies move towards player when spawned.	Spawn in enemies near player	Enemies move towards player if player is close or else they will be in idle mode.	Ensure that enemies are moving towards player which creates more challenge
Bullet Collision Detection	Test if bullets hit enemies and deduct health from them	Shoot at an enemy	Enemy dies after being hit by a certain amount of bullets	This makes sure that enemies are damaged by the players bullets
Enemy Idle Behaviour	Test if enemy will move in random directions patrolling the	Stand far away from enemy	Enemy moves in random directions without	Prevents enemies from staying in one place after they have spawned

	map when the player is not close		acknowledging player	
Game Over	Test if game ends when player health zero	Let player get hit by enemies until health zero	Game over screen displayed when health zero	Ensures that the game ends and displays screen to inform player has died.
Calculating Score	Test if score increases every kill or minute survived	Kill 3 enemies in 60 seconds	Score reaches expected amount	Ensures score is calculated correctly so that the players performance can be represented by it
Health Bar	Test if health bar decrease's when damage is taken and regenerates when health is gained	Take damage and regenerate health	Health bar decreases and increases by correct amount	Confirms that health bar displays correct information and if player takes damage or not
Enemy Spawns	Test if enemies spawn in at random intervals with the frequency increasing as the game progresses	Observe the frequency and spawn patterns for the enemies	Enemies spawn at random intervals with frequency increasing as time goes on	Ensures enemies spawn at a rate which makes them unpredictable and provides a challenge
Power Ups	Test each power up one by one ensuring they update corresponding stats	Collect each power up	Each power up will either temporarily upgrade a stat or bring back some health	To make sure each power up is upgrading its unique abilities for the correct amount of time
Leaderboard Submission	Test if player's final score is submitted to	Submit players score to JSON file	The score will display on the global leader board	Confirms leader board works correctly allowing players

	the leader board			to compare scores
Game Pause	Test if game pauses when pause button pressed	Press pause button during a game	Game stops until its resumed again and starts from exactly where it was stopped	Ensures game can be paused and continued smoothly
Restart Game	Ensure game is set back to its original state after restart button pressed	Press restart button mid-game	Game timer will be set to 0, all enemies and power ups will be killed, and players position will move back to spawn with original start game stats.	Ensures restart button properly resets the game to exactly how it was when it initially started
Changing Key binds	Test if key bindings can be changed	Change W, A, S, D for arrow keys	Player moves correctly with arrow keys	Ensures players can customise their controls so that it fits their needs
Log In authentication	Test if player can only log in with correct password	Enter a correct password and an incorrect password	If correct password entered, log in is successful or else an error message is displayed	Allows only the person that knows the password to log in.
Colourblind mode	Test if each colour-blind mode outputs correct colour blind filter	Enable each colour-blind mode one by one.	Each colourblind mode will output its corresponding filter in game.	Ensures users with accessibility needs get correct settings for their needs

Implementation

Programming



I have created 2 files, one for the main game loop and one to store all the game constants which I have named settings. I also imported the pygame library and the settings file onto the main file.

```
1   import pygame
2   from settings import *
3   pygame.init()
4
5   #Window
6
7   window = pygame.display.set_mode((WIDTH, HEIGHT))
8   pygame.display.set_caption("Bullet Storm")
9
10  #Frame Rate
11
12  GameClock = pygame.time.Clock()
13
14  #Main Loop
15
16  while True:
17      UserInputs = pygame.key.get_pressed()
18      for event in pygame.event.get():
19          if event.type == pygame.QUIT:
20              pygame.quit()
21              exit()
```

I have now created a window for the game and named the window Bullet Storm. The width and height variables are stored in the settings file. The game clock will be used to keep the game at 60 FPS constantly. The main loop is used to cycle through the user inputs and events so when the user closes the window the game quits.

```python
import pygame
pygame.init()

PLAYER_1_IMAGE = pygame.image.load('enemy1idle1.png').convert_alpha()
#PLAYER_2_IMAGE = pygame.image.load('enemy2idle1.png').convert_alpha()
#PLAYER_3_IMAGE = pygame.image.load('enemy3idle1.png').convert_alpha()
```

created a file called assets where I will store PNG images of all my assets as variables which I will then import over to the main file. The player assets were found online from aztrakatze.itch.io and the file was named enemy even though it is being used for the player.

```python
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = PLAYER_1_IMAGE
        self.pos = pygame.math.Vector2(START_X, START_Y)

player = Player()
```

created a class for the player and added its image, gave it a position on the screen then initialised it.

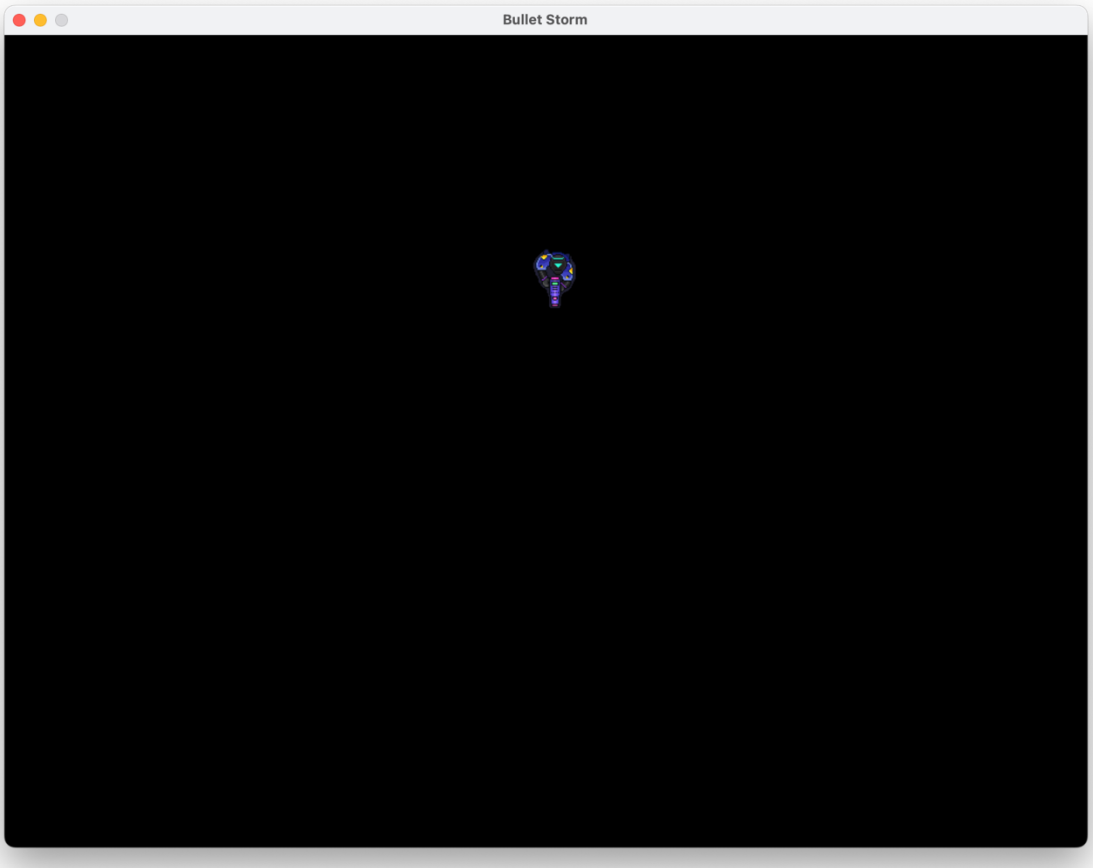
```python
while True:
    UserInputs = pygame.key.get_pressed()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()

    window.blit(player.image, player.pos)

    pygame.display.update()
    GameClock.tick(FPS)
```

In the main looped I blitzed the player onto the screen and added the game clock so that the window updates 60 times every second.

```
Traceback (most recent call last):
  File "/Users/waeztariq/PycharmProjects/PythonGrind/implementationn/bulletstorm.py", line 3, in <module>
    from assets import *
  File "/Users/waeztariq/PycharmProjects/PythonGrind/implementationn/assets.py", line 4, in <module>
    PLAYER_1_IMAGE = pygame.image.load('enemy1idle1.png').convert_alpha()
                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
pygame.error: No video mode has been set

Process finished with exit code 1
```

ran the code and received this error due to a mistake I made of converting the player image on the assets file and not the main file. I removed convert.alpha() from the variable PLAYER_1_IMAGE and added it to self.image.

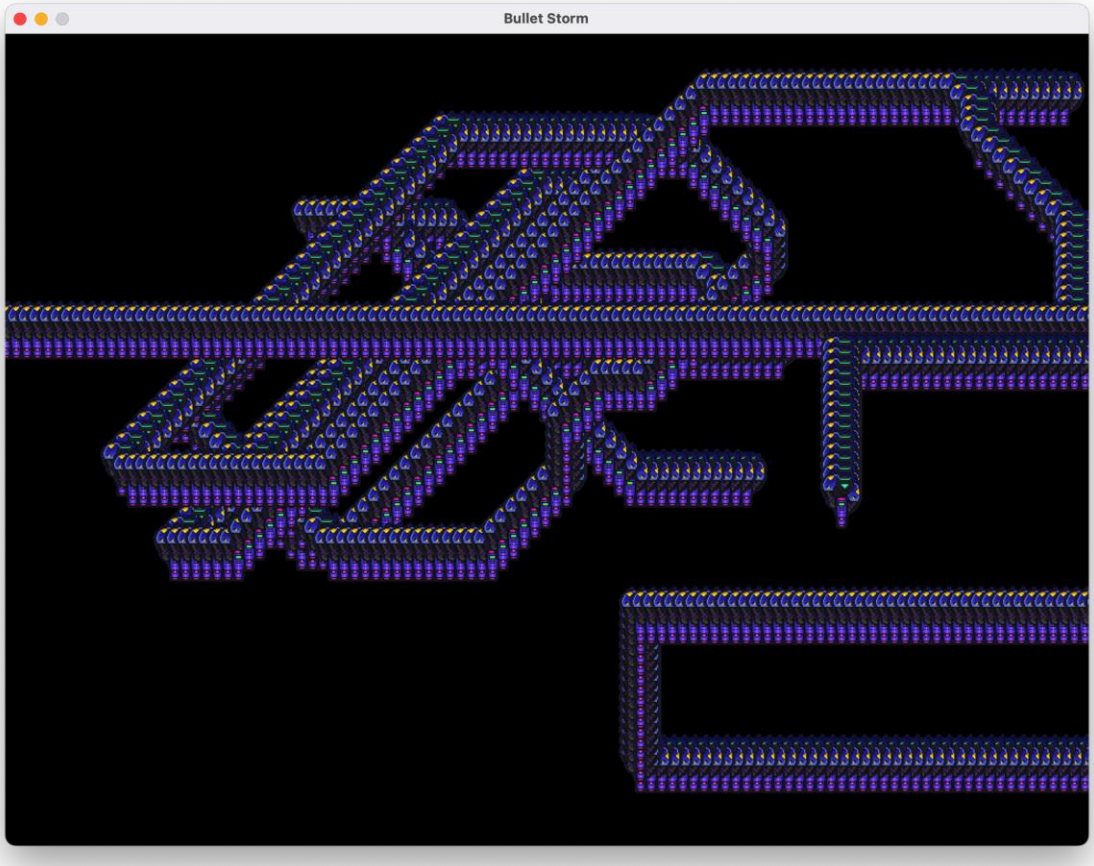
```
self.image = PLAYER_1_IMAGE.convert_alpha()
```



The code is now running as expected and this is the output. The character is displayed correctly on the screen.

```
21      def Movement(self):
22          self.speed_x = 0
23          self.speed_y = 0
24
25          keys_pressed = pygame.key.get_pressed()
26
27          if keys_pressed[pygame.K_a]:  # LEFT
28              self.speed_x -= self.speed
29          if keys_pressed[pygame.K_d]:  # RIGHT
30              self.speed_x += self.speed
31          if keys_pressed[pygame.K_w]:  # UP
32              self.speed_y -= self.speed
33          if keys_pressed[pygame.K_s]:  # DOWN
34              self.speed_y += self.speed
35
36      def Move(self):
37          self.pos += pygame.math.Vector2(self.speed_x, self.speed_y)
38
39
40      def update(self):
41          self.Movement()
42          self.Move()
43
```

have created 3 new methods inside of the player class so that the player can move. The keys W, A, S, D will make the player move in their respective directions. The Move method adds on the velocity of the player to its coordinates on the map and the update method will allow functions to be called onto the main loop.



The player is now moving correctly however when the player image is drawn onto the screen, it is not deleted after every loop therefore I will clear the screen before anything new is drawn onto the screen.

```
window.fill((0, 0, 0))
window.blit(player.image, player.pos)
player.update()
```

The screen is now working correctly however when moving diagonally, the players speed doubles due to 2 speeds being added to the players position at one time. I searched online for a solution to this and from a YouTube video from 'JCode' found out that if the diagonal keys are being pressed at the same time, I need to divide the x and y velocity by the square root of 2.

```
if self.speed_x != 0 and self.speed_y != 0:
    self.speed_x /= math.sqrt(2)
    self.speed_y /= math.sqrt(2)
```

To get the square root function I had to import the maths module. The player is now moving correctly.

```
import pygame
pygame.init()

PLAYER_1_FRAME1 = pygame.image.load('2dassets/idle/enemy1idle1.png')
PLAYER_1_FRAME2 = pygame.image.load('2dassets/idle/enemy1idle2.png')
PLAYER_1_FRAME3 = pygame.image.load('2dassets/idle/enemy1idle3.png')
PLAYER_1_FRAME4 = pygame.image.load('2dassets/idle/enemy1idle4.png')
```

In the assets file I have added 4 idle frames for the player which will cycle through when player is stationary.

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.idle_frames = [PLAYER_1_FRAME1.convert_alpha(),
                            PLAYER_1_FRAME2.convert_alpha(),
                            PLAYER_1_FRAME3.convert_alpha(),
                            PLAYER_1_FRAME4.convert_alpha()

        ]
        self.current_frame = 0
        self.image = self.idle_frames[self.current_frame]
        self.pos = pygame.math.Vector2(START_X, START_Y)
        self.speed = PLAYER_SPEED
        self.frame_count = 0
        self.frame_speed= 0.15

```

In the player class I have created a list for all the idle frames and replaced self.image with the list.

```
    def animate(self):
        if self.speed_x == 0 and self.speed_y == 0:
            self.frame_count += self.frame_speed
            if self.frame_count >= len(self.idle_frames):
                self.frame_count = 0

            self.current_frame = int(self.frame_count)
            self.image = self.idle_frames[self.current_frame]

```

I then created a method within the class where if the players speed is 0, it will iterate through all the frames and go back to the first frame when finished. Then I updated the current frame based on the frame count on self.image. When I ran the code, the player was still stuck on just one frame as I didn't pass through the animate method through the update function. Now the player is animated correctly when idle.

```
def animate(self):
    if self.speed_x == 0 and self.speed_y == 0:
        self.frame_count += self.frame_speed
        if self.frame_count >= len(self.idle_frames):
            self.frame_count = 0

        self.current_frame = int(self.frame_count)
        self.image = self.idle_frames[self.current_frame]

    elif self.speed_x != 0 and self.speed_y != 0:
        self.frame_count += self.frame_speed
        if self.frame_count >= len(self.idle_frames):
            self.frame_count = 0

        self.current_frame = int(self.frame_count)
        self.image = self.walk_frames[self.current_frame]
```

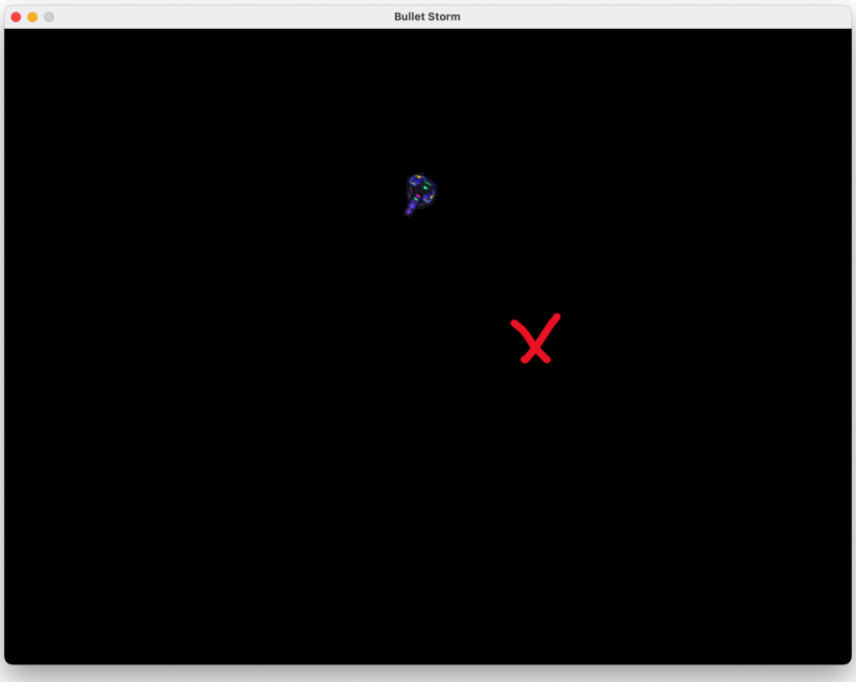
I have now repeated this but added frames for the player when they are moving. I ran the code and for some reason the player's walking animation wasn't working. This is because I set the condition of the speed of x AND y both being greater than zero instead of just x OR y. I also put the len(self.idle_frames) instead of len(self.walk_frames). The animation is now working correctly.

```
self.hitbox_rect = self.base_player_image.get_rect(center=self.pos)
self.rect = self.hitbox_rect.copy()
```

I now created 2 rectangles for the player where one will act as a hitbox and be used for collision detection and the other rectangle which will be used to draw the player onto the screen.

```
def Aiming(self):
    self.mouse_coords = pygame.mouse.get_pos()
    self.x_change_mouse_player = (self.mouse_coords[0] - self.hitbox_rect.centerx)
    self.y_change_mouse_player = (self.mouse_coords[1] - self.hitbox_rect.centery)
    self.angle = math.degrees(math.atan2(self.y_change_mouse_player, self.x_change_mouse_player))
    self.image = pygame.transform.rotate(self.base_player_image, -self.angle)
    self.rect = self.image.get_rect(center=self.hitbox_rect.center)
```

then created a new method for aiming where the angle between the mouse pointer and the player will be used to make the player rotate towards the mouse pointer. This is done by taking the distance between the mouse pointer and the centre of the players hitbox rectangle and then using the tan function to work out the angle in between. However, when I ran the code, I came across 2 issues.



Firstly, the player was rotating correctly but wasn't facing towards the mouse pointer where the red cross is. To fix this problem I added 90 degrees to the players rotation and now it is now aligned.

```
self.image = pygame.transform.rotate(self.base_player_image, -self.angle + 90)
```

The second issue is that the player animations are no longer working. This is because my aiming method overwrites the image with the rotation method, effectively removing the animation's influence. To fix this I rotated the current frame rather than the base frame.

```
self.base_player_image = self.idle_frames[self.current_frame]
```

```
if pygame.mouse.get_pressed() == (1, 0, 0):
    self.shoot = True
    self.Shooting()
else:
    self.shoot = False
```

I now created a new if statement in the movement method which will detect if the player is clicking the left mouse button. If they are then Shooting = True.

```
class Bullet(pygame.sprite.Sprite):
    def __init__(self, x, y, angle):
        super().__init__()
        self.image = BULLET_1.convert_alpha()
        self.image = pygame.transform.rotozoom(self.image, 0, BULLET_SCALE)
        self.rect = self.image.get_rect()
        self.rect.center =(x, y)
        self.x = x
        self.y = y
        self.speed = BULLET_SPEED
        self.angle = angle
        self.x_speed = math.cos(self.angle * (2*math.pi/360)) * self.speed
        self.y_speed = math.sin(self.angle * (2*math.pi/360)) * self.speed
```

I next created a new class for the bullets. This class defines the behaviour of the bullet such as where it will spawn, how fast it will move and how it will change its direction.

```
def bullet_movement(self):
    self.x += self.x_speed
    self.y += self.y_speed


    self.rect.x = int(self.x)
    self.rect.y = int(self.y)
```

This method then updates the bullet and its rectangles position on the map causing it to move.

```
def update(self):
    self.bullet_movement()
```

Then I created an update method so that it updates every loop.

```
SpritesGroup = pygame.sprite.Group()
BulletGroup = pygame.sprite.Group()
SpritesGroup.add(player)
```

I then created 2 sprite groups, one for characters such as the characters/objects and one for the bullets. Then I added the player class to the first sprite group. This will allow me to update all the sprite all at once which will make it easier after when i add the enemy sprites.

```
def Shooting(self):
    if self.shoot_cooldown == 0:
        self.shoot_cooldown = SHOOT_COOLDOWN
        spawn_bullet_pos = self.pos
        self.bullet = Bullet(spawn_bullet_pos[0], spawn_bullet_pos[1], self.angle)
        BulletGroup.add(self.bullet)
        CharacterGroup.add(self.bullet)
```

Within the player class I created a shooting method where if the player is shooting, the bullets will spawn in their designated position. There is also a cooldown function which acts as a timer where the player must wait to press shoot again. SHOOT_COOLDOWN is equal to the integer 30. The player can only shoot when it is equal to zero and every time they shoot it goes back to 30.

```
def update(self):
    self.Movement()
    self.Move()
    self.Animate()
    self.Aiming()
```

```
    if self.shoot_cooldown > 0:
        self.shoot_cooldown -= 1
```

In the update method the cooldown is subtracted by one each loop until it reaches 0 allowing the player to shoot.

```
while True:
    UserInputs = pygame.key.get_pressed()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()


    window.fill((0, 0, 0))


    CharacterGroup.draw(window)
    CharacterGroup.update()


    pygame.display.update()
    GameClock.tick(FPS)
```

I now changed the main loop to where the character groups will all be drawn and updated at the same time.

The bullets are now firing correctly however there are 2 issues. The bullets are not directly coming from the barrel of the weapon, and they do not despawn after a certain amount of time which can cause performance issues if the game is played long enough.

```
self.bullet_life = BULLET_LIFE
self.spawn_time = pygame.timeget_ticks()
```

In the bullet class I created 2 instances. One for how long until the bullet is deleted and one which counts how long the bullet has been spawned in for.

```
if pygame.time.get_ticks() - self.spawn_time > self.bullet_life:
    self.kill()
```

In the bullet movement method i added a if statement where if the bullet has spawned in longer than its BULLET_LIFE it will despawn. To test if it is now working correctly I set the bullet speed very slow and saw it despawn so it is working.

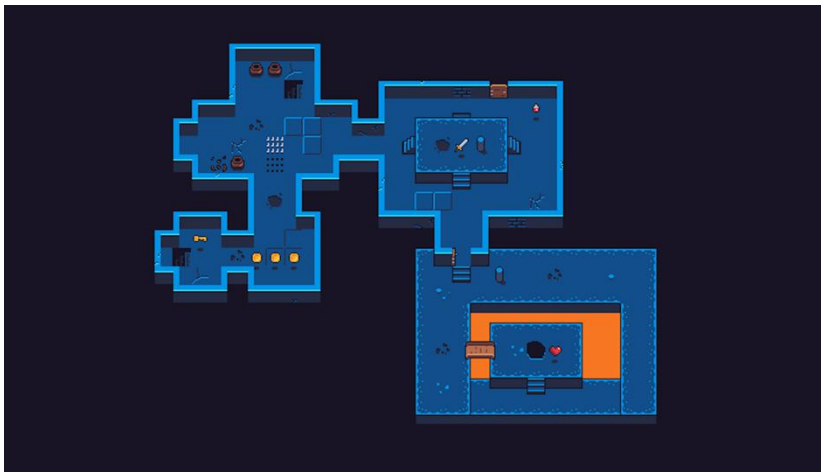
```
spawn_bullet_pos = self.pos + self.gun_pos.rotate(self.angle)
```

To make the bullets come out the barrel I added the guns offset position to the bullet spawn point. The rotate function makes sure it works at all angles.

```
class Camera(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.offset = pygame.math.Vector2()
```

```
        self.floor_rect = MainMap.get_rect(topleft=(0, 0))

    def custom_draw(self):
        self.offset.x = player.rect.centerx - WIDTH // 2
        self.offset.y = player.rect.centery - HEIGHT // 2

        floor_offset_pos = self.floor_rect.topleft - self.offset
        window.blit(MainMap, floor_offset_pos)



        for sprite in CharacterGroup:
            offset_pos = sprite.rect.topleft - self.offset
            window.blit(sprite.image, offset_pos)
```

To create the camera, I borrowed code from Shaw Code on YouTube and adjusted the code so that it fits into my program. To make the camera work you have make the background move in the opposite direction to the player to create an illusion that a camera is following the player.



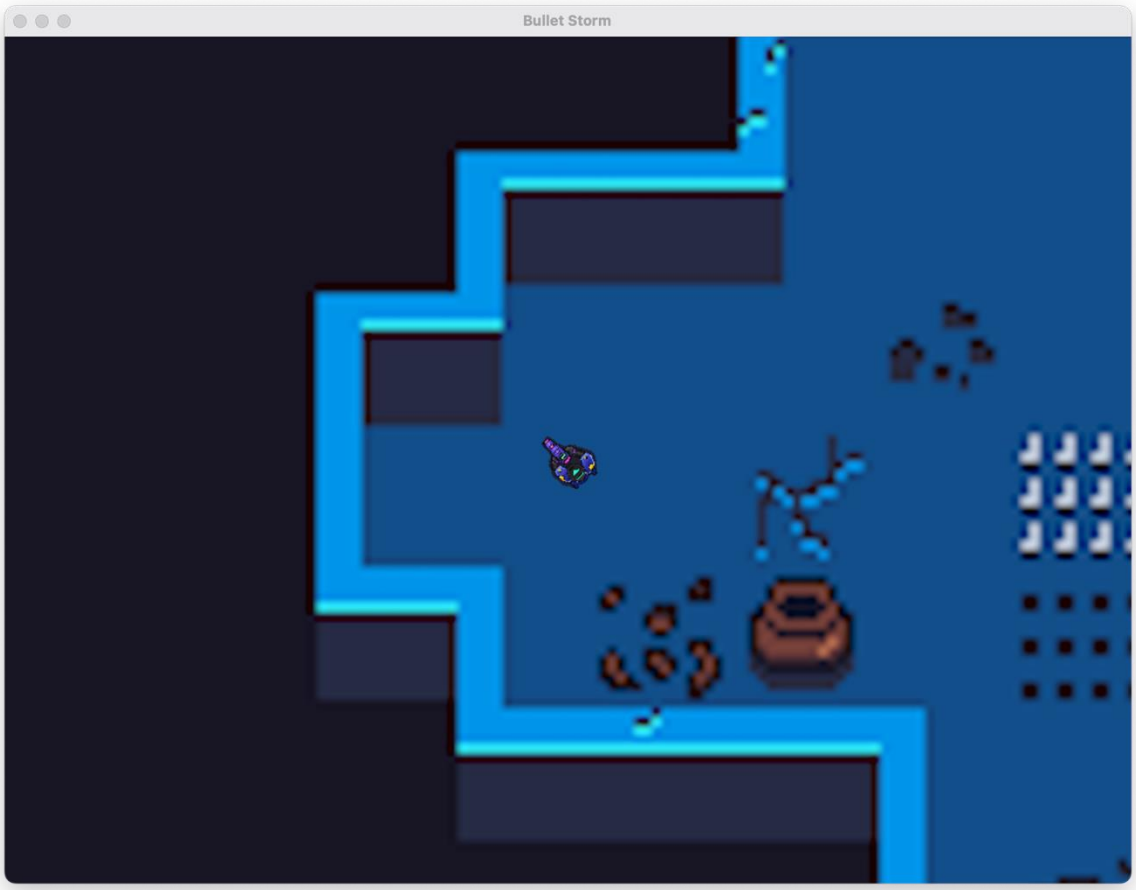
For the time being I am setting the background image to this so that i can test if the map is working or not.

```
MainMap = pygame.transform.rotozoom(BACKGROUND_1.convert(), 0, .7)
```

This is the code for the map. I have scaled it so that it is zoomed in.

```
while True:
    UserInputs = pygame.key.get_pressed()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()


    window.fill((0, 0, 0))


    camera.custom_draw()


    #CharacterGroup.draw(window)
    CharacterGroup.update()
```

```
    pygame.display.update()
    GameClock.tick(FPS)
```
In the main loop I got rid of CharacterGroup.draw(window) to avoid drawing the sprites on twice as they are all now being drawn in the custom_draw method.



The camera is now working correctly however the player is no longer facing towards the mouse when I try to aim.

```
self.x_change_mouse_player = (self.mouse_coords[0] - WIDTH//2)
self.y_change_mouse_player = (self.mouse_coords[1] - HEIGHT//2)
```
To fix the code I had to subtract the mouse coordinates by the centre of the screen instead of the position of the players hitbox as the rectangle has no offset applied. The camera is now working.

```
EnemyGroup = pygame.sprite.Group()
```
I next created a sprite group for my enemies.

```
class Enemy(pygame.sprite.Sprite):
    def __init__(self, position):
        super().__init__(EnemyGroup, CharacterGroup)
        self.frames = [ENEMY_1_FRAME_1.convert_alpha(),
                       ENEMY_1_FRAME_2.convert_alpha(),
                       ENEMY_1_FRAME_3.convert_alpha(),
                       ENEMY_1_FRAME_4.convert_alpha(),
```

```
                        ENEMY_1_FRAME_5.convert_alpha(),
                        ENEMY_1_FRAME_6.convert_alpha(),
                        ENEMY_1_FRAME_7.convert_alpha(),
                        ENEMY_1_FRAME_8.convert_alpha()]
        self.current_frame = 0
        self.image = self.frames[self.current_frame]
        self.rect = self.image.get_rect()
        self.rect.center = position
```

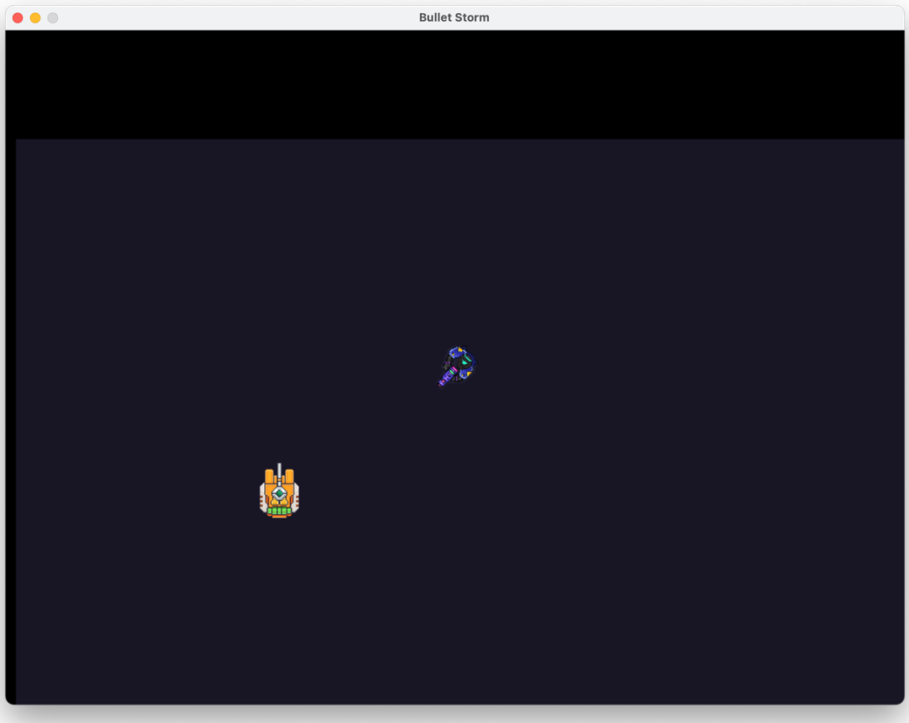
Then I created a class for the enemy with all its frames stored in a list and then created a rectangle for it.

```
enemy1 = Enemy((300, 400))
```

I created an instance for it then passed in the coordinates for its centre then ran the code.

```
Traceback (most recent call last):
  File "/Users/waeztariq/PycharmProjects/PythonGrind/implementation/bulletstorm.py", line 195, in <module>
    enemy1 = Enemy((400, 400))
             ^^^^^^^^^^^^^^^^
  File "/Users/waeztariq/PycharmProjects/PythonGrind/implementation/bulletstorm.py", line 157, in __init__
    super().__init__(EnemyGroup, CharacterGroup)
                     ^^^^^^^^^^
NameError: name 'EnemyGroup' is not defined
```

received this error. This is because I did not define all the sprite groups before creating the sprites. I moved all the sprite group definition code to the top of the program before creating the sprites and it is now working.



The enemy sprite is now visible on the map.

```
def Animate(self):
    self.frame_count += self.frame_speed
    if self.frame_count >= len(self.frames):
        self.frame_count = 0

    self.current_frame = int(self.frame_count)
    self.image = self.frames[self.current_frame]



def update(self):
    self.Animate()
```

I then copied and pasted the animate function from the player class and adjusted it for the enemy and added it to enemy class.

```
def FollowPlayer(self):
    player_vector = pygame.math.Vector2(player.hitbox_rect.center)
    enemy_vector = pygame.math.Vector2(self.rect.center)
```

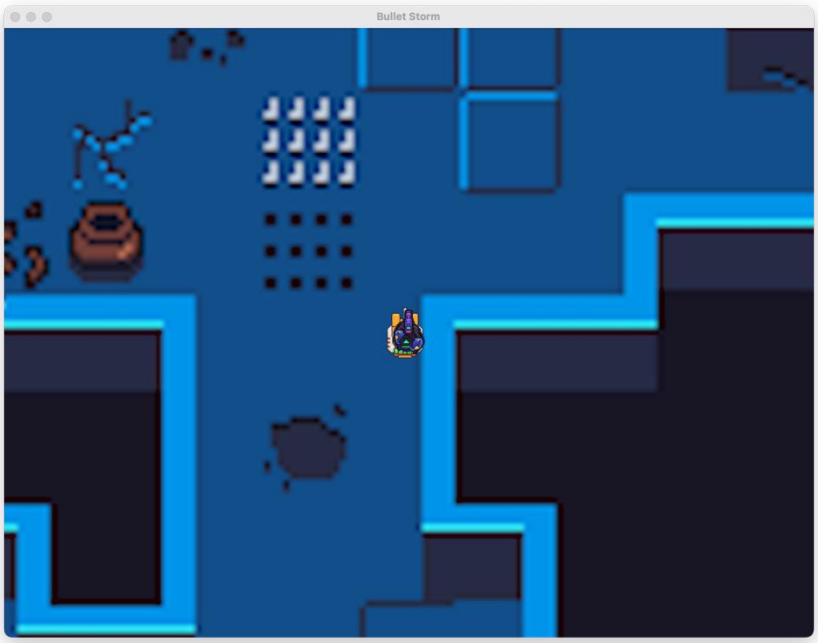
This new method is for the enemy to follow the player. First it will get the player and enemies vector position.

```
def enemy_distance(self, player_vector_distance, enemy_vector_distance):
    return (player_vector_distance - enemy_vector_distance).magnitude()
```

This method finds out the magnitude of how far away the enemy is from the player.

```
def follow_player(self):
    player_vector = pygame.math.Vector2(player.hitbox_rect.center)
    enemy_vector = pygame.math.Vector2(self.rect.center)
    distance = self.enemy_distance(player_vector, enemy_vector)

    if distance > 0:
        self.direction = (player_vector - enemy_vector).normalize()
        self.velocity = self.direction * self.speed
        self.rect.center += self.velocity
    else:
        self.direction = pygame.math.Vector2()
```

In the follow player function I used the logic that if the distance between the player and enemy is greater than 0 the enemy will chase the player else it will stay still.



The enemy now follows the player however it gets too close and doesn't rotate towards the player as it moves.

```
def rotate_towards_player(self, player):
    dx = player.rect.centerx - self.rect.centerx
    dy = player.rect.centery - self.rect.centery
    angle = math.degrees(math.atan2(dy, dx))

    self.image = pygame.transform.rotate(self.base_enemy_image, -angle - 90)
    self.rect = self.image.get_rect(center=self.rect.center)
```

I created a new method which allows the enemy to rotate towards the player by calculating the angle between the enemy and players current position. Then using the angle, the image is rotated, and the rectangle is centred to the enemy's centre. I ran the code, and the rotation was working however the animation stopped working. This is because I am not using the base enemy image instance in the animation method.

```
self.base_enemy_image = self.frames[self.current_frame]
```

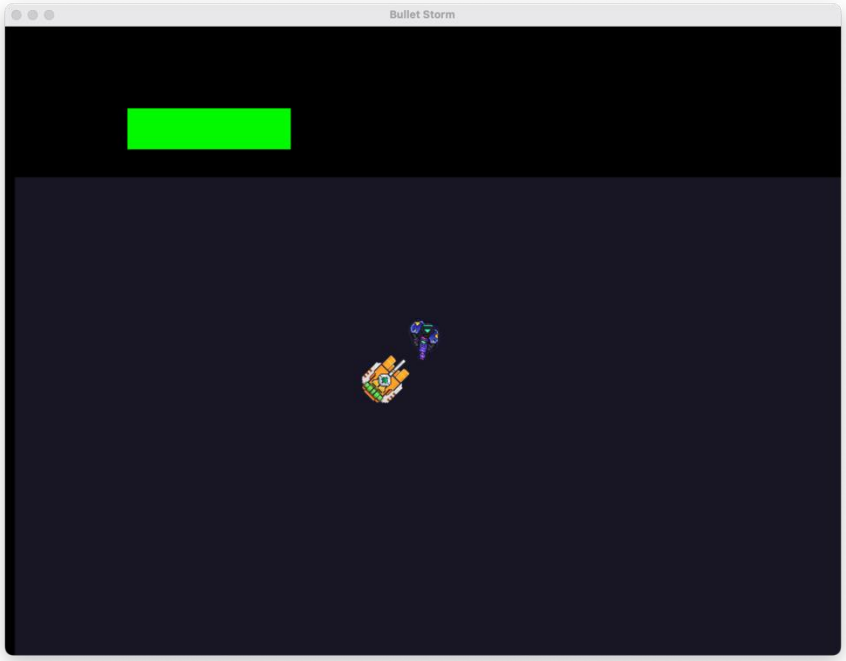
I then made the distance between the player and enemy have to be greater than 70 for the enemy to follow.

```
if distance > 70:
```

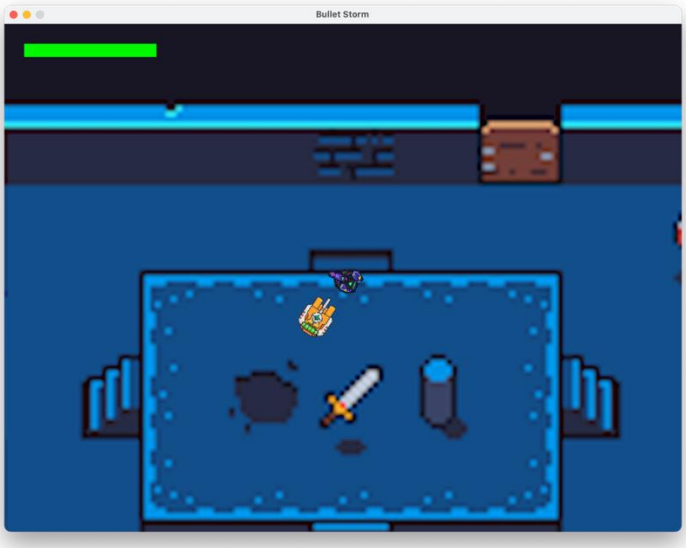
The enemy is now facing the player and is not too close.

```
class HealthBar():
    def __init__(self, x, y, w, h, max_hp):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.hp = max_hp
        self.max_hp = max_hp

    def draw(self, surface):
        ratio = self.hp / self.max_hp
        pygame.draw.rect(surface, "red", (self.x, self.y, self.w, self.h))
        pygame.draw.rect(surface, "green", (self.x, self.y, self.w * ratio, self.h))



health_bar = HealthBar(150, 100, 200, 50, 100)
```

I next added a health bar class which uses rectangles to draw the health bar onto the screen. The red bar is the maximum health and drawn over it is a green bar for remaining health. When the health decreases the width of the green bar will decrease making the red bar be displayed. This code was from a tutorial by Coding with Russ on YouTube.



The health bar is now on the screen however it is not positioned correctly on the screen and is too big.



It is now displayed at the correct position on the screen however it does not match the games pixel art theme.

```
class HealthBar():
    def __init__(self, x, y, w, h, max_hp):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.hp = max_hp
        self.max_hp = max_hp
        self.background_image = pygame.transform.scale(HEALTH_BAR_1, (self.w, self.h))  # Empty health
bar
        self.foreground_image = pygame.transform.scale(HEALTH_BAR_2, (self.w, self.h))  # Filled health
bar


    def draw(self, surface):
        ratio = self.hp / self.max_hp


        surface.blit(self.background_image, (self.x, self.y))
        filled_width = int(self.w * ratio)


        if filled_width > 0:
            filled_bar = self.foreground_image.subsurface((0, 0, filled_width, self.h))
            surface.blit(filled_bar, (self.x, self.y + 31))
```

In my new health bar class, instead of using rectangles, i am using 2 images of a health bar, one that is full and one that is emptly. I am drawing the full bar over the empty bar and then multiplying it by the ratio variable which will decrease the width of the full bar as the players health decreases. This will then reveal the empty bar below and create the effect of the health decreasing. If the ratio is 0, this means the health is 0 and then the full bar will not be drawn at all.

```
def collision_detect(self):
    for enemy in EnemyGroup:
        if self.rect.colliderect(enemy.rect):
            ENEMY_HEALTH = ENEMY_HEALTH - 10
            self.kill()
```

I now created a collision detection method in the bullet class where if the bullets rectangle collides with enemy the bullet disappears, and the enemy's health decreases by 10.

```
  File "/Users/waeztariq/PycharmProjects/PythonGrind/implementation/bulletstorm.py", line 161, in update
    self.collision_detect()
  File "/Users/waeztariq/PycharmProjects/PythonGrind/implementation/bulletstorm.py", line 156, in collision_detect
    ENEMY_HEALTH -= 10
    ^^^^^^^^^^^^
UnboundLocalError: cannot access local variable 'ENEMY_HEALTH' where it is not associated with a value
```

ran the code and received this error.

```
def collision_detect(self):
    for enemy in EnemyGroup:
        if self.rect.colliderect(enemy.rect):
            self.kill()
```

I removed the line where the enemy's health goes down and now the bullet is disappearing upon collision.

```
self.health = ENEMY_HEALTH
self.damage = BULLET_DAMAGE
```
In the enemy class a created an instance for enemy health and in the bullet class I created one for bullet damage.

```
def damage_taken(self, damage):
    self.health -= damage
    if self.health <= 0:
        self.kill()
```
I then created a method for the enemy to take damage and if health reaches zero they die.

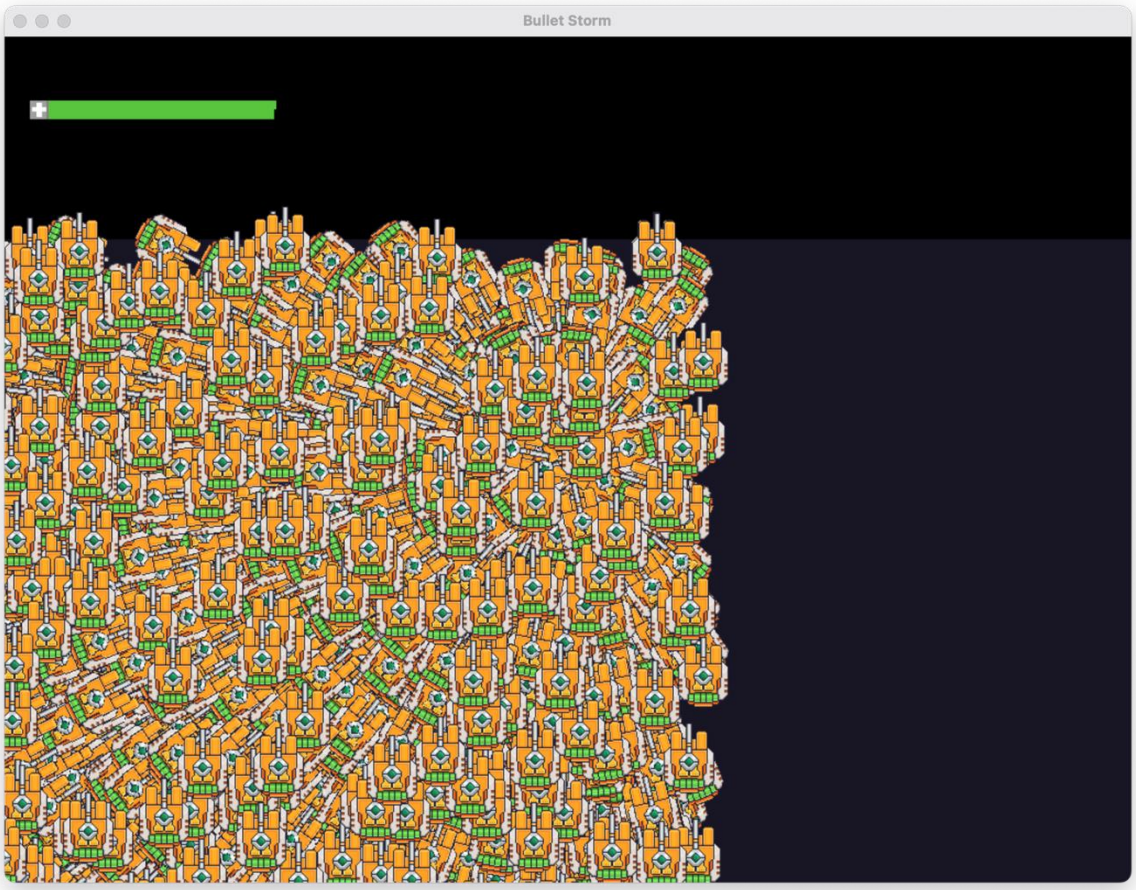
```
def collision_detect(self):
    for enemy in EnemyGroup:
        if self.rect.colliderect(enemy.rect):
            enemy.take_damage(self.damage)
            self.kill()
```
Finally, I put the enemy damage function into the collision detection method.

```
AttributeError: 'Enemy' object has no attribute 'take_damage'
```

I received this error because I put the wrong name for the enemy to take damage function.

```
enemy.damage_taken(self.damage)
```
I changed the code, and it is now working correctly. The enemy disappears off the screen when its health reaches 0.

```
def enemy_spawn(self):
    current_time = pygame.time.get_ticks()

    if current_time - self.last_spawn_time > self.enemy_spawn_time:
        self.last_spawn_time = current_time

       spawn_x = random.randint(0, WIDTH)
       spawn_y = random.randint(0, HEIGHT)

       new_enemy = Enemy((spawn_x, spawn_y))
        EnemyGroup.add(new_enemy)
       CharacterGroup.add(new_enemy)
```
In the enemy class I created a new function which spawns in enemies after a certain amount of time in a random location on the map. For this I had to import the random module so that I can generate random spawn locations.



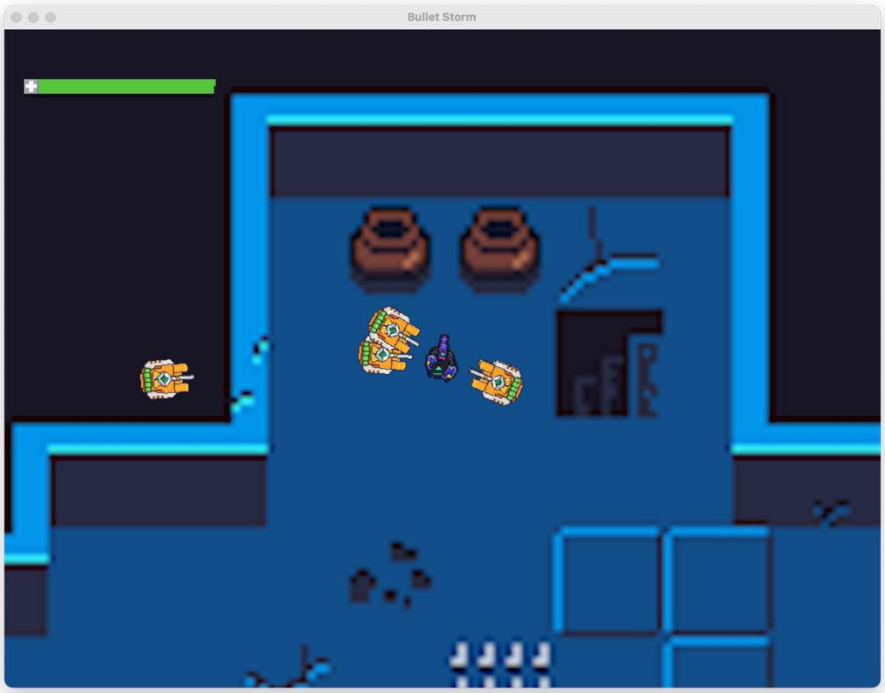
I ran the code and came across a huge error. The enemies kept spawning in at a very fast rate and not periodically after a certain amount of time. This happened because I implemented the enemy spawning logic inside the update method of the enemy class meaning that every frame new enemy will spawn.

```
class EnemySpawner:
    def __init__(self, spawn_time, enemy_group):
        self.spawn_time = spawn_time
        self.last_spawn_time = pygame.time.get_ticks()
        self.enemy_group = enemy_group



    def enemy_spawn(self):
        current_time = pygame.time.get_ticks()
        if current_time - self.last_spawn_time > self.spawn_time:
            self.last_spawn_time = current_time

            spawn_x = random.randint(0, WIDTH)
            spawn_y = random.randint(0, HEIGHT)

            new_enemy = Enemy((spawn_x, spawn_y))
            EnemyGroup.add(new_enemy)
```

I have now instead made a separate class for enemy spawning with its own method that will be called in the main loop.

```
enemy_spawner = EnemySpawner(spawn_time= ENEMY_SPAWN_TIME, enemy_group= EnemyGroup)
```

I then created an instance for it and ran the code.



The enemies are now spawning in correctly at regular intervals.

```
class EnemyBullet(pygame.sprite.Sprite):
    def __init__(self, x, y, angle):
        super().__init__()
        self.image = BULLET_1.convert_alpha()  # Use an image for enemy bullets
        self.image = pygame.transform.rotozoom(self.image, 0, BULLET_SCALE)
        self.rect = self.image.get_rect(center=(x, y))
        self.speed = BULLET_SPEED
        self.angle = angle
        self.bullet_life = BULLET_LIFE  # Define bullet life span
        self.spawn_time = pygame.time.get_ticks()

        # Calculate bullet velocity based on the angle
        self.x_speed = math.cos(math.radians(self.angle)) * self.speed
        self.y_speed = math.sin(math.radians(self.angle)) * self.speed

    def update(self):
        self.rect.x += self.x_speed
        self.rect.y += self.y_speed

        if pygame.time.get_ticks() - self.spawn_time > self.bullet_life:
            self.kill()
```

I now created a new class for enemy bullets which has the same logic as the player bullets but now shoot from the enemies.

```
self.shoot_cooldown = 1000
self.last_shot_time = pygame.time.get_ticks()
```

In the enemy class I defined the shot cooldown for the enemy and the time since the last shot.

```
def shoot_player(self, player):
    current_time = pygame.time.get_ticks()

    if current_time - self.last_shot_time >= self.shoot_cooldown:
        self.last_shot_time = current_time

        dx = player.rect.centerx - self.rect.centerx
        dy = player.rect.centery - self.rect.centery
        angle = math.degrees(math.atan2(dy, dx))

        new_bullet = EnemyBullet(self.rect.centerx, self.rect.centery, angle)
        BulletGroup.add(new_bullet)
```

Then I created a new method in which the enemy will shoot a bullet towards the players direction after the shoot cooldown has finished. I ran the code, but the enemy bullets were not displaying on the screen as I didn't add the bullet group to the draw method on the camera class.

```
def custom_draw(self):
    self.offset.x = player.rect.centerx - WIDTH // 2
    self.offset.y = player.rect.centery - HEIGHT // 2

    floor_offset_pos = self.floor_rect.topleft - self.offset
    window.blit(MainMap, floor_offset_pos)



    for sprite in CharacterGroup:
        offset_pos = sprite.rect.topleft - self.offset
        window.blit(sprite.image, offset_pos)

    for bullet in BulletGroup:
        offset_bullet_pos = bullet.rect.topleft - self.offset
        window.blit(bullet.image, offset_bullet_pos)
```

I updated the method and now the enemy bullets are shown on the screen.

However, the bullets currently do no damage.

```
def collision_detect(self):
    if self.rect.colliderect(player.rect):
        player.damage_taken(self.damage)
        self.kill()
```

First, I added a collision detection method within the enemy bullet class to damage player if bullet collides.

```
def damage_taken(self, damage):
    self.health -= damage
    if self.health <= 0:
        self.kill()
```

Next i added a damage taken method for the player which c=subtracts the players health by the enemy bullets damage and kills the player when health reaches zero.

The player now dies when health reaches zero however the healthbar is not working correctly as it is staying the same.

```
def draw(self, surface):
    ratio = self.player.health / PLAYER_HEALTH
```

I changed the logic for the ratio to use the players actual health and then ran the code and received this error.

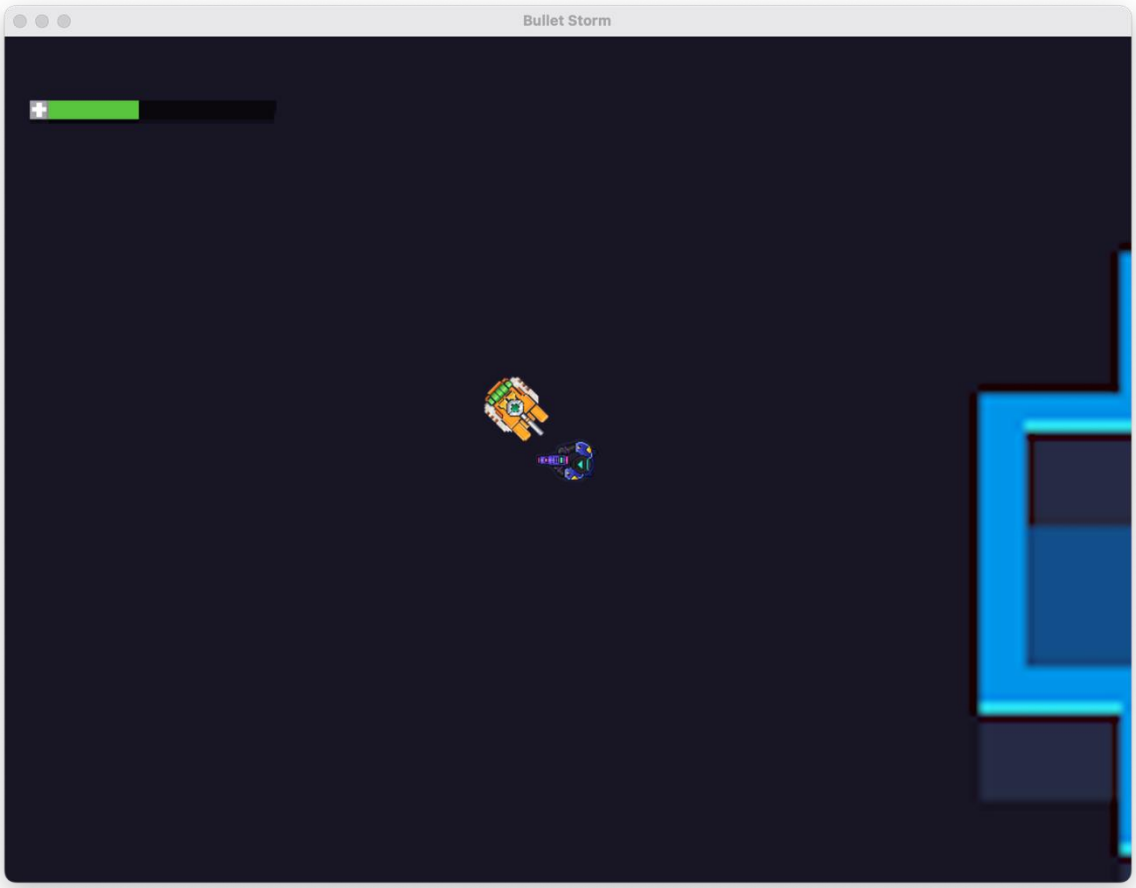
```
AttributeError: 'HealthBar' object has no attribute 'player'
```

```
class HealthBar():
    def __init__(self, player, x, y, w, h):
        self.player = player
```

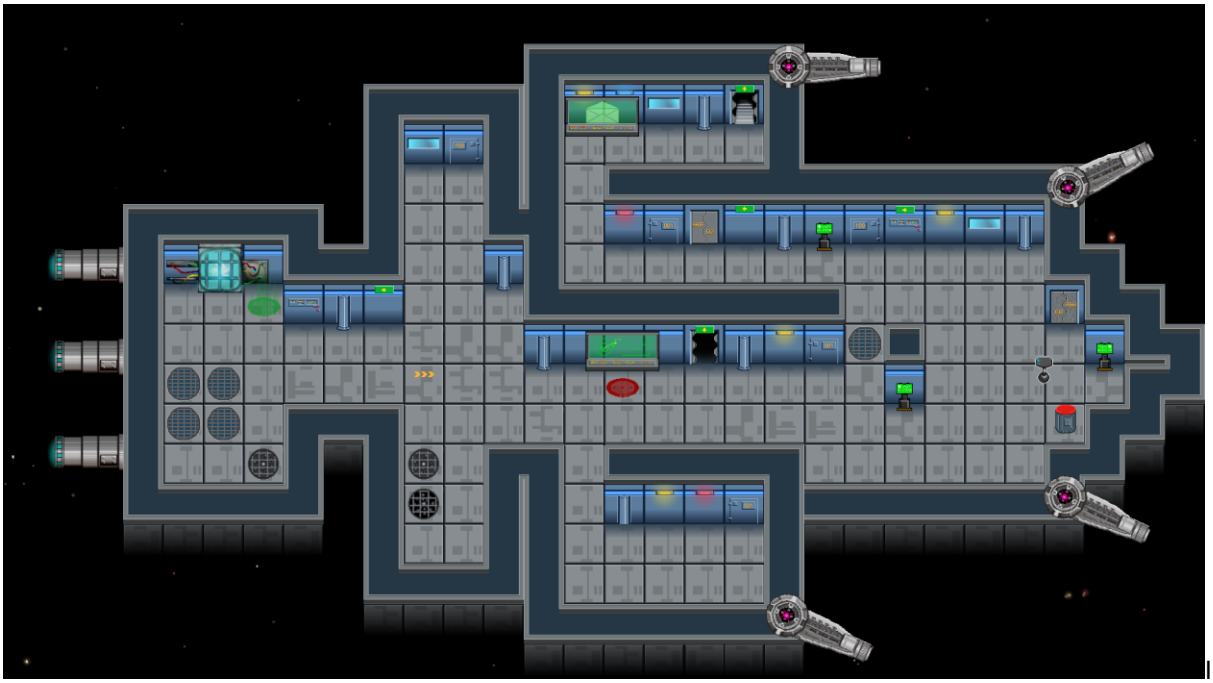
I defined player in the class.

```
health_bar = HealthBar(player, 2, -50, 300, 300)
```

Then updated the health bar instance and ran the code.



The health bar is now working correctly, decreasing each time the player gets hit.



next found this free space themed map on itch.io and replaced the previous temporary map. I will now work on creating boundaries in the map.

```
def boundary_detection(self):
    if self.pos.y < 0:
        self.speed = 0
    else:
        pass
```

First, I will create a border around the main map's rectangle. To do this I created a method within the player class which checks if the players position in the map is out of bounds or not. I ran the code to see if it is working and found that the player gets stuck as soon as they reach the border.

```
def boundary_detection(self):
    if self.pos.y < 100:
        self.pos.y = 100
    else:
        pass
```

To fix the issue, instead of setting their speed to zero, i set the players position equal to the boundaries position so that they can still move.

```
def boundary_detection(self):
    if self.pos.y < 100:
        self.pos.y = 100
    elif self.pos.y > 1500:
        self.pos.y = 1500


    if self.pos.x < 200:
        self.pos.x = 200
    elif self.pos.x > 3000:
        self.pos.x = 3000
```

Now there is a base perimeter around the map which the player cannot exit.

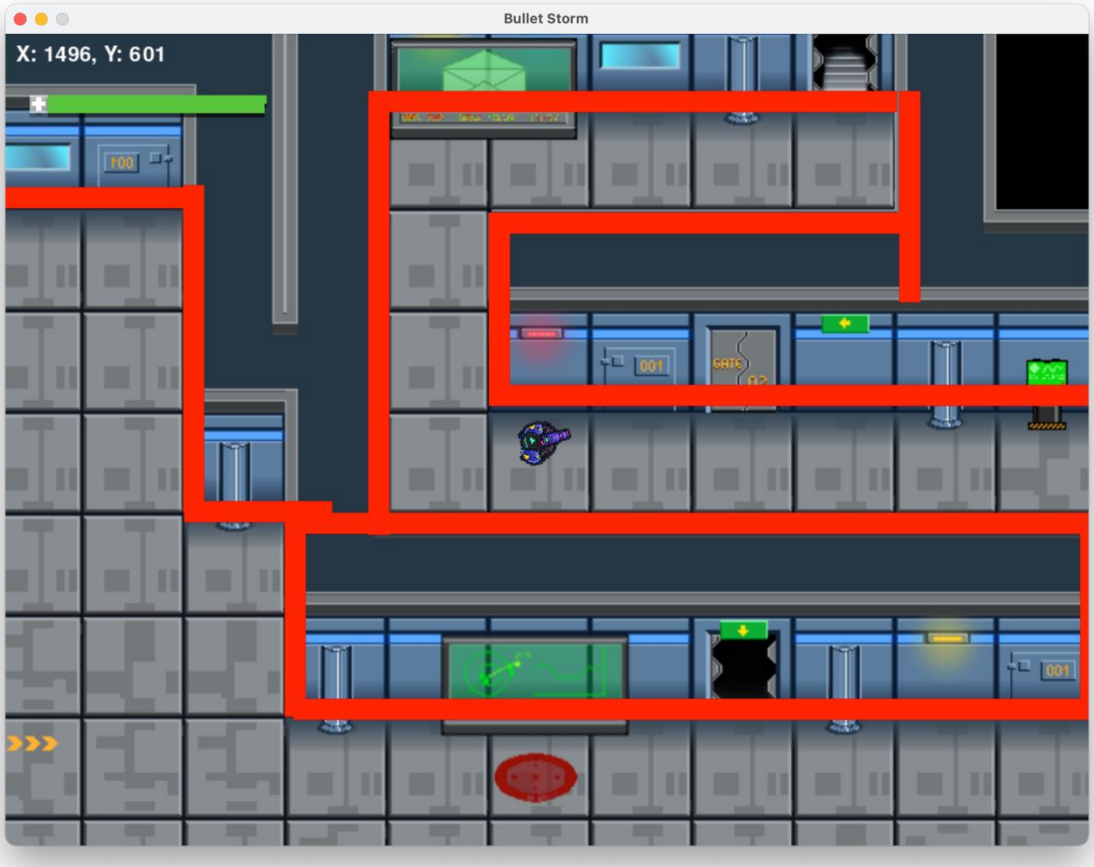
```
def display_coordinates(self, surface):
    coordinates_text = self.font.render(f'X: {int(self.pos.x)}, Y: {int(self.pos.y)}', True, (255, 255,
255))
    surface.blit(coordinates_text, (10, 10))
```

Next I created a method that will show me the players exact coordinates on the map so that I can use it to calculate what sized rectangles i will need to create boundaries in the walls off the actual map.

```
class MapBoundaries:
    def __init__(self):
        self.boundaries = [
            pygame.Rect(100, 100, 200, 20),
           pygame.Rect(300, 200, 20, 100),
           pygame.Rect(500, 400, 50, 50),
       ]

    def draw(self, surface):
        for boundary in self.boundaries:
            pygame.draw.rect(surface, (255, 0, 0), boundary)
```

I created a class for the rectangles so that i can see them on the screen making sure that they will cover the maps boundaries correctly.

The red rectangle is my first boundary and I will do this all over the map using he coordinates displayed top left to get the positions.



I have now completed this across the whole map and will now update the boundary method in the player class.

```
def boundary_detection(self):
    for boundary in map_boundaries.boundaries:
        if self.hitbox_rect.colliderect(boundary):
            if self.hitbox_rect.bottom > boundary.top and self.hitbox_rect.top < boundary.top:
                self.pos.y = boundary.top - 30


            elif self.hitbox_rect.top < boundary.bottom and self.hitbox_rect.bottom > boundary.bottom:
                self.pos.y = boundary.bottom + 30


            elif self.hitbox_rect.right > boundary.left and self.hitbox_rect.left < boundary.left:
                self.pos.x = boundary.left - 30


            elif self.hitbox_rect.left < boundary.right and self.hitbox_rect.right > boundary.right:
                self.pos.x = boundary.right + 30
```

In the updated boundary detection method, i removed the old perimeter boundary as it is not needed and changed the code so that if the players hit box collides with the boundary rectangles their position will be set the the outside of the boundary rectangles, causing them to stop.

```
def collision_detect(self):
    for enemy in EnemyGroup:
        if self.rect.colliderect(enemy.rect):
            enemy.damage_taken(self.damage)
            self.kill()
    for boundary in map_boundaries.boundaries:
```

```
        if self.rect.colliderect(boundary):
            self.kill()
```
Next, I updated the collision detection method in the bullet class so that bullets cannot travel through walls. Next, I need to add this same logic in the enemy class but first need to update their spawn method so that they can only spawn inside of the map and not on the boundaries of the map.

```
ENEMY_SPAWN_LOCATIONS = [(1773, 337), (1771, 1293), (620, 1094), (2438, 629), (2643, 915)]
```
First I created a list of possible enemy spawn points. I will add more locations to this list later.

```
def enemy_spawn(self):
    current_time = pygame.time.get_ticks()
    if current_time - self.last_spawn_time > self.spawn_time:
        self.last_spawn_time = current_time


        spawn_point = random.randint(ENEMY_SPAWN_LOCATIONS)


        new_enemy = Enemy((spawn_point))
        EnemyGroup.add(new_enemy)
```
Then i changed the enemy spawn method so that it chooses a random spawn point from the list.

```
TypeError: Random.randint() missing 1 required positional argument: 'b'
```
ran the code and received this error because i used the wrong function from the random module to select and item from the list.

```
spawn_point = random.choice(ENEMY_SPAWN_LOCATIONS)
```
I changed the code to this and now the enemies are spawning at the correct locations.

```
enemy = Enemy(random.choice(ENEMY_SPAWN_LOCATIONS))
```
I updated the enemy instance to so that the first enemy also spawns in a random location. Now i need to add the boundary detection method into the enemy class.

```
def boundary_detection(self):
    for boundary in map_boundaries.boundaries:
        if self.rect.colliderect(boundary):
            if self.rect.bottom > boundary.top and self.rect.top < boundary.top:
                self.rect.bottom = boundary.top
            elif self.rect.top < boundary.bottom and self.rect.bottom > boundary.bottom:
                self.rect.top = boundary.bottom
            elif self.rect.right > boundary.left and self.rect.left < boundary.left:
                self.rect.right = boundary.left
            elif self.rect.left < boundary.right and self.rect.right > boundary.right:
                self.rect.left = boundary.right
```
I added the boundary detection method to the enemy class using the same logic for the player class however, the enemies can still shoot through walls.

```
def collision_detect(self):
    if self.rect.colliderect(player.rect):
        player.damage_taken(self.damage)
        self.kill()
    for boundary in map_boundaries.boundaries:
        if self.rect.colliderect(boundary):
            self.kill()
```

The enemies can no longer shoot through the walls, however they can still see the players through the walls. To fix this i will create an idle state where the enemy will only react to the player if they come close enough.

```
def check_idle(self, player):
    player_vector = pygame.math.Vector2(player.hitbox_rect.center)
    enemy_vector = pygame.math.Vector2(self.rect.center)
    distance = self.enemy_distance(player_vector, enemy_vector)

    if distance < 200:
        self.idle = False
    else:
        self.idle = True
```

The enemy will be idle until the player is close enough to them.

```
def idle_behavior(self):
    current_time = pygame.time.get_ticks()

    if current_time - self.patrol_timer > self.patrol_cooldown:
        self.patrol_timer = current_time
        random_angle = random.uniform(0, 2 * math.pi)
        self.patrol_direction = pygame.math.Vector2(math.cos(random_angle),
math.sin(random_angle)).normalize()
```

While in idle, the enemy will patrol the map in random directions and keep doing this until it sees the player.

```
idle_speed = self.speed / 4
self.velocity = self.patrol_direction * idle_speed
self.rect.center += self.velocity
```

It will move at reduced speed when at idle, so the player is able to distinguish between enemy states.

```
angle = math.degrees(math.atan2(self.patrol_direction.y, self.patrol_direction.x))
self.image = pygame.transform.rotate(self.base_enemy_image, -angle - 90)
self.rect = self.image.get_rect(center=self.rect.center)
```

Finally, the enemy will rotate towards the direction it is moving so it looks more natural.

```
def follow_player(self):
    player_vector = pygame.math.Vector2(player.hitbox_rect.center)
    enemy_vector = pygame.math.Vector2(self.rect.center)
    distance = self.enemy_distance(player_vector, enemy_vector)
    self.check_idle(player)

    if 70 < distance and not self.idle:
        self.direction = (player_vector - enemy_vector).normalize()
        self.velocity = self.direction * self.speed
        self.rect.center += self.velocity
    elif self.idle:
        self.idle_behavior()
```

I updated the follow player function so that the enemy only follows the player when not in idle state and if the enemy is not following the player, it goes to idle behaviour.

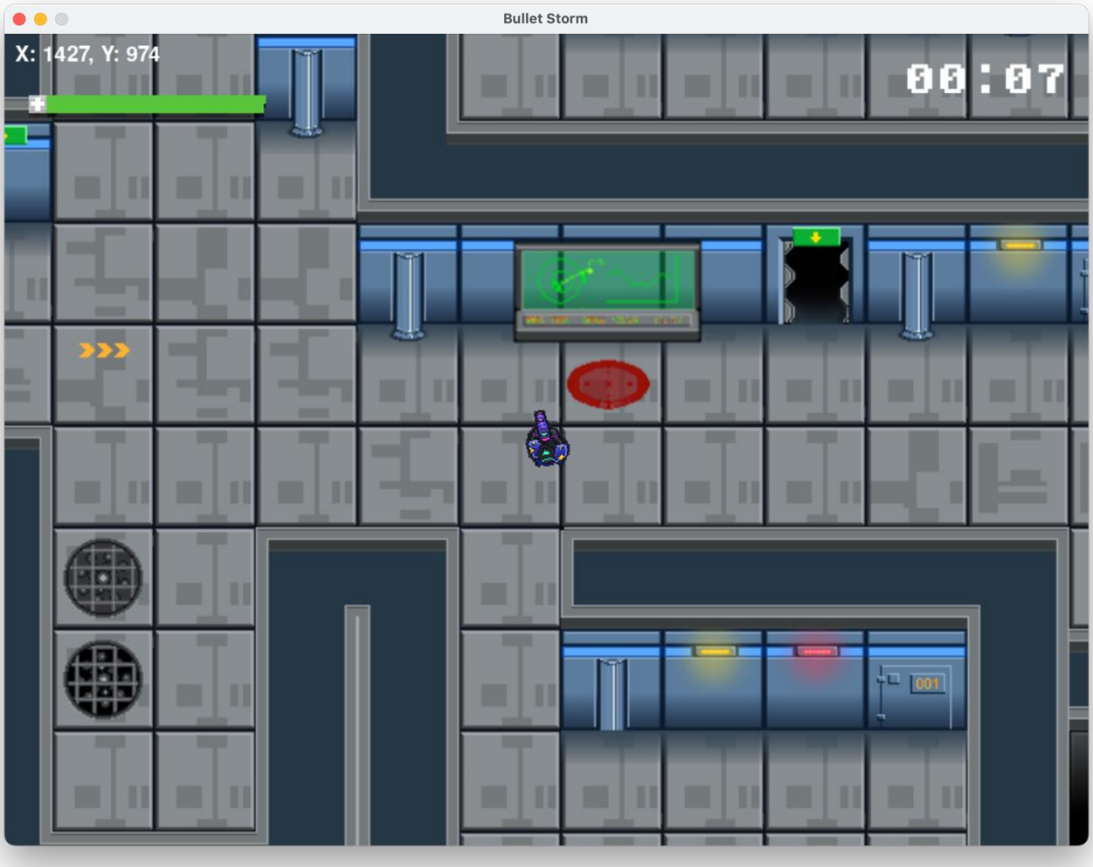
```
if not self.idle:
```

I added this if statement to the rotate towards player and shoot at player methods, so the enemy only does this when it's not in idle.

In screenshot above enemies are in idle and not attacking player.

```
#TIMER
current_time = pygame.time.get_ticks()
elapsed_time = (current_time - start_time) // 1000
minutes = elapsed_time // 60
seconds = elapsed_time % 60
timer = f"{minutes:02}:{seconds:02}"
timer_surface = FONT.render(timer, True, WHITE)

window.fill((0, 0, 0))
camera.custom_draw()
#map_boundaries.draw(MainMap)
window.blit(timer_surface, (window.get_width() - timer_surface.get_width() - 20, 20))
health_bar.draw(window)
enemy_spawner.enemy_spawn()
```

Next, I added a timer which starts as soon as the game starts so the player can see how long they are alive for.



Next i initialised pygame.mixer and added gun sounds for the player.

```
GUN_SOUND = pygame.mixer.Sound('2dassets/Sound Effects/gun_shot.wav')
def Shooting(self):
    if self.shoot_cooldown == 0:
        self.shoot_cooldown = SHOOT_COOLDOWN
        spawn_bullet_pos = self.pos + self.gun_pos.rotate(self.angle)
        self.bullet = Bullet(spawn_bullet_pos[0], spawn_bullet_pos[1], self.angle)
        BulletGroup.add(self.bullet)
        CharacterGroup.add(self.bullet)
        GUN_SOUND.play()
```

Every time the player shoots the sound plays.

```
GUN_SOUND = pygame.mixer.Sound('2dassets/Sound Effects/gun_shot.wav')
GUN_SOUND.set_volume(0.5)
GUN_SOUND2 = pygame.mixer.Sound('2dassets/Sound Effects/gun_shot2.wav')
GUN_SOUND2.set_volume(0.15)
```

I added a sound effect for the enemy shooting and adjusted the sounds for both of them so that the player one is louder.

```
def Movement(self):
    self.speed_x = 0
    self.speed_y = 0

    keys_pressed = pygame.key.get_pressed()
    moved = False

    if keys_pressed[pygame.K_a]:  # LEFT
        self.speed_x -= self.speed
        moved = True
    if keys_pressed[pygame.K_d]:  # RIGHT
        self.speed_x += self.speed
        moved = True
    if keys_pressed[pygame.K_w]:  # UP
        self.speed_y -= self.speed
        moved = True
    if keys_pressed[pygame.K_s]:  # DOWN
        self.speed_y += self.speed
        moved = True

    current_time = pygame.time.get_ticks()
```

For footsteps i updated the movement function to return true every time player is moving, and footstep sounds will play continuously with 500ms gaps while player is moving.

```
def damage_taken(self, damage):
    self.health -= damage
    if self.health <= 0:
        self.kill()
        ENEMY_DEATH_SOUND.play()
```

Next I added an 8 bit explosion sound for when the enemies die.

```
def damage_taken(self, damage):
    self.health -= damage
    if self.health <= 0:
        self.kill()
        GAME_OVER.play()
```

I then added a retro game over sound for when the player dies but there was a problem as the sound kept repeating even after the player died.

```
def damage_taken(self, damage):
    self.health -= damage
    if self.health <= 0 and not self.game_over_sound:
        self.game_over_sound = True
        self.kill()
        GAME_OVER.play()
```

To fix this i created a game over sound instance which is set from false to true when the player dies allowing for the sound to only play once.

```
def damage_taken(self, damage):
    self.health -= damage
    HITMARKER.play()
    if self.health <= 0:
        self.kill()
        ENEMY_DEATH_SOUND.play()
```

Then I added a hit marker sound for everytime the bullet hits the enemy.

```
def damage_taken(self, damage):
    self.health -= damage
    HURT.play()
```

Then i added one for the player getting hurt.

Next I will add a reload function to the game.

```
def Reload(self):
    keys_pressed = pygame.key.get_pressed()

    if keys_pressed[pygame.K_r]:
        self.ammo = 30
```

This function puts the players ammo back to 30 after 'R' is pressed.

```
def Shooting(self):
    if self.shoot_cooldown == 0 and self.ammo > 0:
        self.shoot_cooldown = SHOOT_COOLDOWN
        spawn_bullet_pos = self.pos + self.gun_pos.rotate(self.angle)
        self.bullet = Bullet(spawn_bullet_pos[0], spawn_bullet_pos[1], self.angle)
        self.ammo -= 1
        BulletGroup.add(self.bullet)
        CharacterGroup.add(self.bullet)
        GUN_SOUND.play()
```

In the shoot function, everytime the player shoots, 1 bullet is taken away from self.ammo and if self.ammo is zero then the player cannot shoot.

The reload method is now working however, i want to add a reload speed so that the gun does not reload instantly.

```
def draw_ammo(self, surface):
    ammo_text = FONT.render(f'{self.ammo}/{self.max_ammo}', True, WHITE)
    surface.blit(ammo_text, (830, 580))
```

First i drew the ammo and max ammo onto the bottom right of the screen so that i can see if it is reloading correctly or not.


```
def Reload(self):
    keys_pressed = pygame.key.get_pressed()

    if keys_pressed[pygame.K_r] and self.ammo < self.max_ammo and not self.reloading:
        self.reloading = True
       self.last_reload_time = pygame.time.get_ticks()

   if self.reloading:
        current_time = pygame.time.get_ticks()
        if current_time - self.last_reload_time >= self.reload_speed:
            self.ammo = self.max_ammo
           self.reloading = False
```

Next i updated the reload method so that when the player presses “R”a timer starts counting until the reload speed is met. When the reload speed is met the guns ammo will go back to max ammo. The self.reload instance is used so that the player cannot shoot while reloading if they reload when ammo is greater than zero.

```
def Shooting(self):
    if self.shoot_cooldown == 0 and self.ammo > 0 and not self.reloading:
```

Then for the shooting method I updated the if statement so that the gun wont shoot during reload or zero ammo. The reload function is now working correctly.

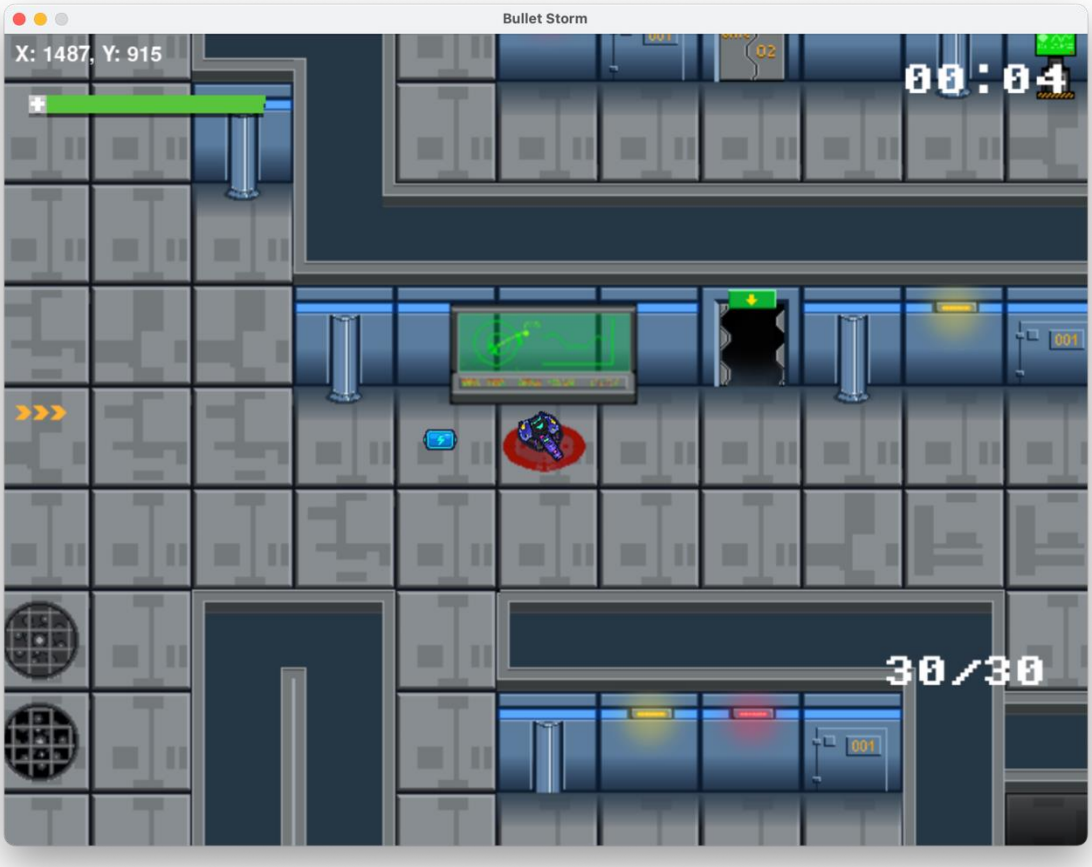
```
class Energy(pygame.sprite.Sprite):
    def __init__(self, pos):
        super().__init__()
        self.frames = [ENERGY_FRAME_1.convert_alpha(),
                       ENERGY_FRAME_2.convert_alpha(),
                       ENERGY_FRAME_3.convert_alpha(),
                       ENERGY_FRAME_4.convert_alpha(),
                       ENERGY_FRAME_5.convert_alpha(),
                       ENERGY_FRAME_6.convert_alpha(),
```

```
                        ENERGY_FRAME_7.convert_alpha(),]

        self.current_frame = 0
        self.image = self.frames[self.current_frame]
        self.rect = self.image.get_rect()
        self.frame_speed = ENERGY_FRAME_SPEED

        self.frame_count = 0
        self.rect.center = pos  # Set the position of the coin
        self.health = 20
```

Next i created my first power up which is called energy. When the player collides with the energy, they gain health.

```
def Animate(self):
    self.frame_count += self.frame_speed
    if self.frame_count >= len(self.frames):
        self.frame_count = 0

    self.current_frame = int(self.frame_count)
    self.image = self.frames[self.current_frame]

def collision_detect(self):
    if self.rect.colliderect(player.rect):
        player.energy_power_up(self.health)
        self.kill()
        ENERGY_SOUND.play()
```

I then animated the energy and created a sound for when you collect it.



Next I added a method for the player to gain health when he collects the energy.

```
def energy_power_up(self, health):
    if self. health <= 80:
        self.health += health
    else:
        self.health = self.max_health
```

The player will gain +20 health if health is 80 and below and go to max health if health over 80 to avoid health going over 100. The player now gains health when touching coin. Now I need to create more spawn locations for the coin.

```
ENERGY_SPAWN_LOCATIONS = [(1579, 1292), (2152, 1005), (1111, 1295), (1012, 426), (1593, 322), (626,
724)]
```

In these locations on the map, the energy will spawn.

```
class PowerupSpawner:
    def __init__(self, spawn_time, powerups_group, spawn_locations):
        self.spawn_time = spawn_time
        self.last_spawn_time = pygame.time.get_ticks()
        self.powerups_group = powerups_group
        self.spawn_locations = spawn_locations
```

I then created a new class which will handle all the power up spawn locations by adding them to the power group, setting their spawn locations and spawn times.

```
def spawn_powerup(self):
    current_time = pygame.time.get_ticks()
    if current_time - self.last_spawn_time > self.spawn_time:
        self.last_spawn_time = current_time
        spawn_point = random.choice(self.spawn_locations)
        new_powerup = self.create_powerup(spawn_point)
        self.powerups_group.add(new_powerup)

def update(self):
    self.spawn_powerup()
```

Then i created a new method which uses the same logic for the enemy spawner but spawns in power ups at given time intervals and adds them to the power up group.

```
class EnergySpawner(PowerupSpawner):
    def create_powerup(self, position):
        return Energy(position)
```

Finally i created a sub class for the energy spawner which inherits from the spawner class and spawns in the energy power up.

```
energy_spawner = EnergySpawner(spawn_time=ENERGY_SPAWN_TIME, powerups_group= powerupsGroup,
spawn_locations=ENERGY_SPAWN_LOCATIONS)
```

I created an instance for the energy spawner.

```
energy_spawner.update()
```

Then updated it in the main loop.

```
NameError: name 'powerupsGroup' is not defined. Did you mean: 'PowerupsGroup'?
```

made a small error where I forgot to write PowerupsGroup correctly and ran the code again. The powerups are now spawning correctly randomly at the given locations.

```
class StrengthBoost(pygame.sprite.Sprite):
    def __init__(self, pos):
        super().__init__()
        self.image = STRENGTH_BOOST.convert_alpha()
        self.rect = self.image.get_rect()
        self.rect.center = pos
        self.duration = STRENGTH_TIME


    def collision_detect(self):
        if self.rect.colliderect(player.rect):
            player.strength_boosted(self.duration)
            self.kill()
            STRENGTH_SOUND.play()


    def update(self):
        self.collision_detect()
```

Next, I created a strength boost powerup where the bullet damage temporarily increases once the powerup is collected.

```
def strength_boosted(self, duration):
    self.is_boosted = True
    self.boost_end_time = pygame.time.get_ticks() + duration
```

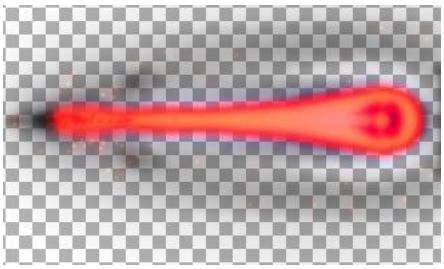
Then i created a timer method in the player class where it starts counting time as soon as the boost is picked up.

```
if self.is_boosted and pygame.time.get_ticks() > self.boost_end_time:
    self.is_boosted = False
```

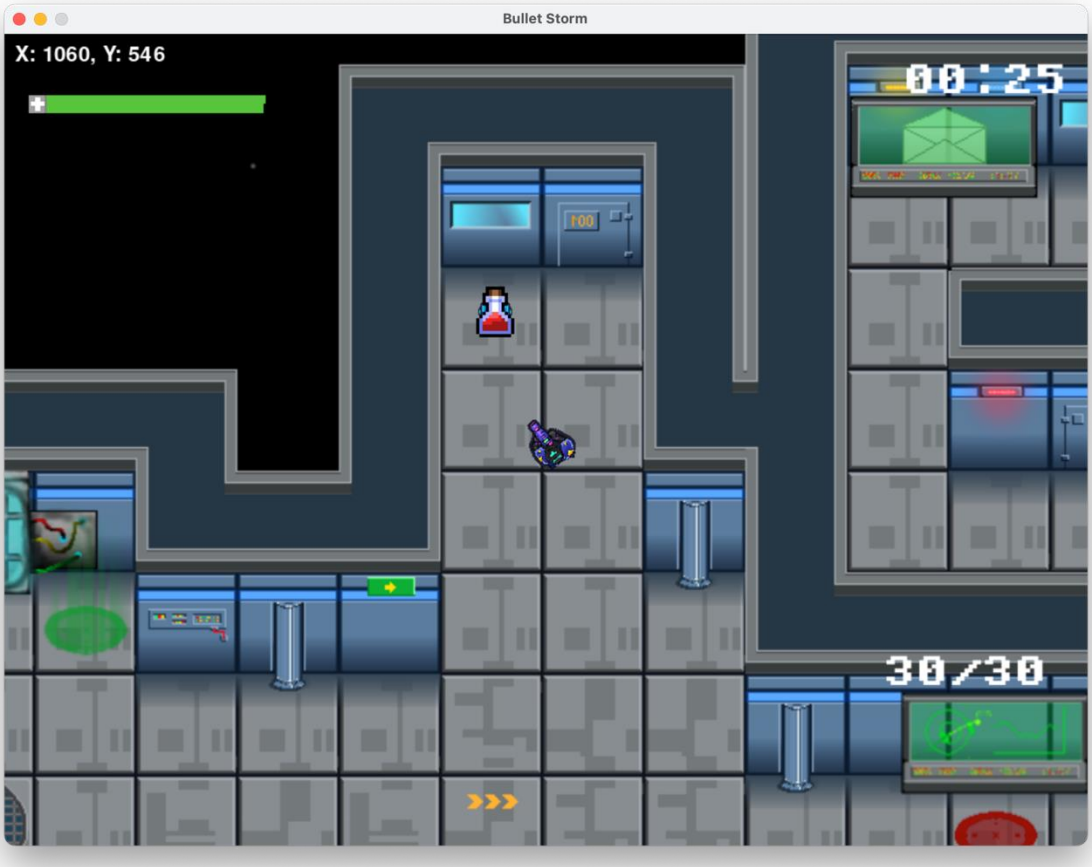
In the update method when the boost is finished, the is boosted instance is set back to false and bullet damage goes back to normal.

```
if player.is_boosted:
    self.damage = STRENGTH_BOOST
    self.image = BULLET_3
    self.image = pygame.transform.rotozoom(self.image, -player.angle , BULLET_SCALE)

else:
    self.damage = BULLET_DAMAGE
```

In the bullet class I made the damage boosted if the strength boost was true otherwise turned it back to default. I also changed the image of the bullet so that the player can tell that the boost is activated.



This new bullet image is larger and has a red colour so that it can easily be distinguished by the regular damage bullets.



I ran the code, and you can now collect the strength boost powerup and the bulets change colour. However, when the bullets hit the enemy, an error is recieved.

```
    self.health -= damage
TypeError: unsupported operand type(s) for -=: 'int' and 'pygame.surface.Surface'
```

This is because i set the bullet damage to STRENGTH_BOOST which is assigned to an image and the new bullet strength at the same time. To fix this problem i renamed the actual strength value to STRENGTH_BOOST_VALUE and now the code is working fine without any problems.

```
if player.is_boosted:
    self.damage = STRENGTH_BOOST_VALUE
```

Next I created its spawner function and initialised it.

```
class StrengthBoostSpawner(PowerupSpawner):
    def create_powerup(self, position):
        return StrengthBoost(position)
strength_boost_spawner = StrengthBoostSpawner(spawn_time=STRENGTH_SPAWN_TIME, powerups_group=
PowerupsGroup, spawn_locations=ENERGY_SPAWN_LOCATIONS)
```

It is now spawning correctly at the random locations. I will rename the variable to POWERUPS_SPAWN_LOCATIONS.

```
class SpeedBoots(pygame.sprite.Sprite):
    def __init__(self, pos):
        super().__init__()
        self.image = SPEED_BOOTS.convert_alpha()
        self.rect = self.image.get_rect()
        self.rect.center = pos
        self.duration = SPEED_BOOTS_TIME


    def collision_detect(self):
        if self.rect.colliderect(player.rect):
            player.speed_boots(self.duration)
            self.kill()
            SPEED_BOOTS_SOUND.play()


    def update(self):
        self.collision_detect()
```

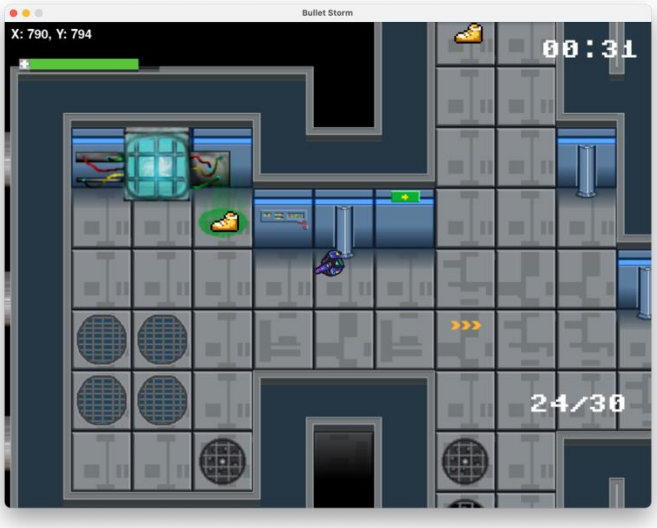
The final powerup was the speed boost power up where the player collects speed boots and gain extra speed temporarily.

```
def speed_boots(self, duration):
    self.speed_boosted = True
    self.speed_boost_end_time = pygame.time.get_ticks() + duration
    self.speed = BOOSTED_SPEED


def new_speed(self):
    if self.speed_boosted:
        if pygame.time.get_ticks() > self.speed_boost_end_time:
            self.speed_boosted = False
            self.speed = PLAYER_SPEED
   else:
        self.speed = self.speed
```

I created 2 methods in the player class. One where the timer for the boost start and one in where the speed boost is applied.

```
self.new_speed()
```

Then I called it in the update method.



The boots are now boosting the players speed when collected.

```
class SpeedBootsSpawner(PowerupSpawner):
    def create_powerup(self, position):
        return SpeedBoots(position)
speed_boots_spawner = SpeedBootsSpawner(spawn_time=SPEED_BOOTS_SPAWN_TIME, powerups_group=
PowerupsGroup, spawn_locations=POWERUPS_SPAWN_LOCATIONS)
```

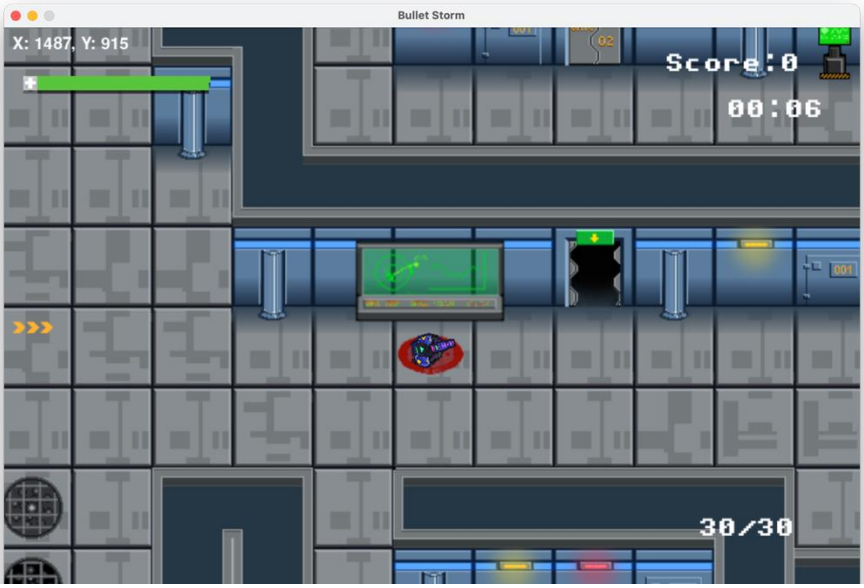
Finally, I created the spawner method and initialised it. I am next going to create the score system.

```
def draw_score(self, surface):
    score_text = FONT.render(f'Score:{self.score}', True, WHITE)
    surface.blit(score_text, (790, 25))
```

In the player class I created a method which draws the score top right of the screen.

```
def increase_score(self, points):
    self.score += points
```

Next, I created another method which will be called to increase the player's score.

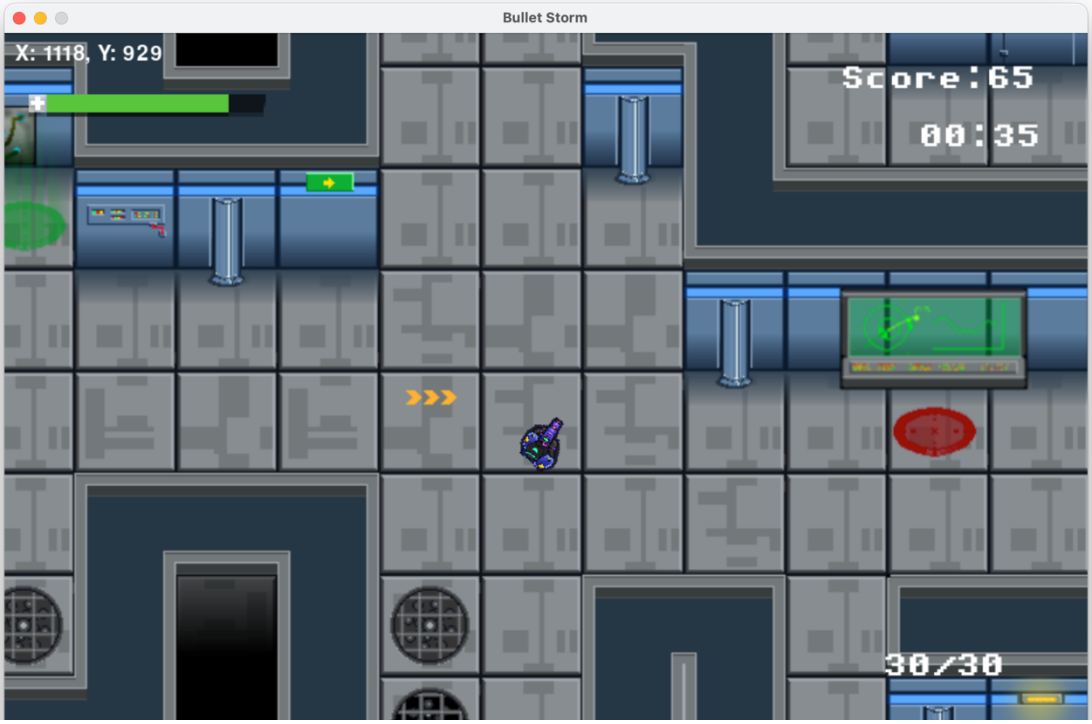



```
def damage_taken(self, damage):
    self.health -= damage
    HITMARKER.play()
    if self.health <= 0:
        self.kill()
        ENEMY_DEATH_SOUND.play()
        player.increase_score(20)
```

When an enemy dies, the score is incremented by 20.

```
def collision_detect(self):
    if self.rect.colliderect(player.rect):
        player.speed_boots(self.duration)
        self.kill()
        SPEED_BOOTS_SOUNDS.play()
        player.increase_score(5)
```

For each power up collected, the score increased by 5.



The score is correctly updating during these events.

```
def time_survived_score(self):
    current_time = pygame.time.get_ticks()
    seconds_passed = (current_time - self.last_score_update) / 1000

    if seconds_passed >= 60:
        self.increase_score(100)
        self.last_score_update = current_time
```

I then added a system where the score increases by 100 for every minute survived.

Now I am going to add a game over screen where the score will be displayed and the time survived.

```
def damage_taken(self, damage):
    self.health -= damage
    HURT.play()
    if self.health <= 0 and not self.dead:
        self.dead = True
        self.kill()
        GAME_OVER.play()
```

First i change the logic of the damage taken method so that it uses a flag to determine whether the player is alive or not.

```
def game_over_screen(self, surface):
    current_time = pygame.time.get_ticks()
    elapsed_time = (current_time - start_time) // 1000
    minutes = elapsed_time // 60
    seconds = elapsed_time % 60
    timer = f"{minutes:02}:{seconds:02}"
    game_over_message = LARGE_FONT.render('GAME OVER', True, (255, 255, 255))
    score_message = FONT.render(f'Your Final Score Was: {self.score}', True, (255, 255, 255))
    time_message = FONT.render(f'You survived for: {timer}', True, (255, 255, 255))
```

Next, I created a game over method and created a variable for a time which counts how long the player was alive for and variables for what text will be displayed.

```
if self.dead:
    window.fill((0, 0, 0))
    surface.blit(game_over_message, (WIDTH/2 -(game_over_message.get_width()/2), HEIGHT/2 +
(game_over_message.get_height()/2)-300))
    surface.blit(score_message, (WIDTH / 2 - (score_message.get_width() / 2), HEIGHT / 2 +
(score_message.get_height() / 2)))
    surface.blit(time_message, (WIDTH / 2 - (time_message.get_width() / 2), HEIGHT / 2 +
(time_message.get_height() / 2)+100))
    pygame.display.update()
```

Then if the player is dead, the text will be displayed on the screen.

```
if player.dead:
    player.game_over_screen(window)
```

Finally, in the main loop I added the method so that the screen can update and display it.

The screen is showing correctly however the timer kept increasing and didn't stop. To fix this problem, i changed the logic of the timer message so that it counts the time of when the player died instead of elapsed time.

```
def damage_taken(self, damage):
    self.health -= damage
    HURT.play()
    if self.health <= 0 and not self.dead:
        self.dead = True
        self.final_time = pygame.time.get_ticks()
        self.kill()
        GAME_OVER.play()
```

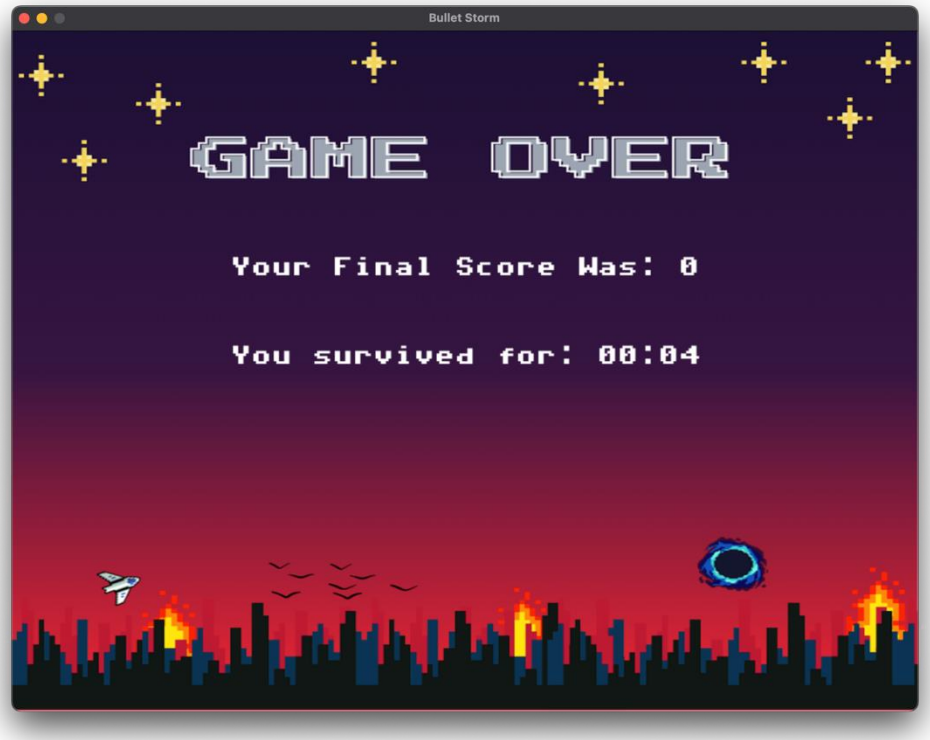
I created a final time instance which is recorded when player dies.

```
elapsed_time = (self.final_time - start_time) // 1000
minutes = elapsed_time // 60
seconds = elapsed_time % 60
timer = f"{minutes:02}:{seconds:02}"
```

Then updated the logic for the timer. It is now displaying correctly.

```
self.game_over_image = GAME_OVER_IMAGE


if self.dead:
    window.fill((0, 0, 0))
    window.blit(self.game_over_image, (0, 0))
    surface.blit(score_message, (WIDTH / 2 - (score_message.get_width() / 2), HEIGHT / 2 +
(score_message.get_height() / 2)-150))
    surface.blit(time_message, (WIDTH / 2 - (time_message.get_width() / 2), HEIGHT / 2 +
(time_message.get_height() / 2)-50))
    pygame.display.update()
```

I next replaced the black background with an image matching the theme of my menu saying, ‘GAME OVER” and removed the font from python.




```
for enemy in EnemyGroup:
    enemy.kill()


pygame.display.update()
pygame.time.wait(30000)
pygame.quit()
sys.exit()
```

To finish off, I killed all enemies as soon as the player dies and then after 30 seconds the game will automatically close.

Next I will create the pause button.

```
def pause_screen(surface):
    pause_text = LARGE_FONT.render('PAUSED', True, WHITE)
    surface.blit(pause_text, (WIDTH / 2 - (pause_text.get_width() / 2), HEIGHT / 2 +
(pause_text.get_height() / 2) - 300))
```

I created a function for the word 'PAUSED' to be displayed at the top of the screen when the game is paused. I then created a variable called paused and set it to false.

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()


        if event.type == pygame.KEYDOWN and event.key == pygame.K_p:
            paused = not paused
```

Then in the main game loop I set paused equal to not paused of the ‘P’ key was pressed so that it can toggle in between the paused state when pressed.

```
if not paused:



    current_time = pygame.time.get_ticks()
    elapsed_time = (current_time - start_time) // 1000
    minutes = elapsed_time // 60
    seconds = elapsed_time % 60
    timer = f"{minutes:02}:{seconds:02}"
    timer_surface = FONT.render(timer, True, WHITE)

    window.fill((0, 0, 0))
    camera.custom_draw()
    # map_boundaries.draw(MainMap)
    window.blit(timer_surface, (window.get_width() - timer_surface.get_width() - 45, 80))
    health_bar.draw(window)
    enemy_spawner.enemy_spawn()
    CharacterGroup.update()
    BulletGroup.update()
    PowerupsGroup.update()
    energy_spawner.update()
    strength_boost_spawner.update()
    speed_boots_spawner.update()

    if player.dead:
        player.game_over_screen(window)

    pygame.display.update()
    GameClock.tick(FPS)

else:
    pause_screen(window)
    pygame.display.update()
```

Finally, I indented all the game logic into the if not paused statement and if it is paused i called the display pause screen function.

```
UnboundLocalError: cannot access local variable 'paused' where it is not associated with a value
```

ran the code and received this error because the paused variable I used is not defined locally. To fix it I defined it is a global variable.

```
global paused
```

I now ran the code and the pause button is working correctly but the in game timer is still increasing while the game is paused.

```
if event.type == pygame.KEYDOWN and event.key == pygame.K_p:
    if not paused:
        paused = True
        paused_time = pygame.time.get_ticks()
   else:
        paused = False
        start_time += pygame.time.get_ticks() - paused_time
```

To fix this problem I modified the pause event statement so that as soon as the pasue button is pressed, a variable will track how long the game is paused for and subtract that value from the game's timer.

```
button_pressed = False
mute_text_message = 'MUTE'
mute_audio = False


def pause_screen(surface):
    global button_pressed, mute_text_message, mute_audio
    pause_text = LARGE_FONT.render('PAUSED', True, WHITE)
    mute_text = FONT.render(mute_text_message, True, WHITE)
    mute_button = pygame.Rect(300, 230, 420, 100)
    mouse_pos = pygame.mouse.get_pos()
    if button_pressed:
        mute_button_image = BUTTON_PRESSED
    else:
        mute_button_image = BUTTON_IDLE
    surface.blit(pause_text, (WIDTH / 2 - (pause_text.get_width() / 2), HEIGHT / 2 +
(pause_text.get_height() / 2) - 300))
    surface.blit(mute_button_image, (mute_button.x, mute_button.y))
    surface.blit(mute_text, (WIDTH / 2 - (50), 250))
```

I then created a mute button. The purpose of this button is to mute all game sounds, and this button will be displayed when the game is paused.


```
if mute_button.collidepoint(mouse_pos):
    if pygame.mouse.get_pressed() == (1, 0, 0):
        BUTTON_CLICK_SOUND.play()
        if not button_pressed:
            button_pressed = True
            mute_audio = not mute_audio
            mute_text_message = 'UNMUTE' if mute_text_message == 'MUTE' else 'MUTE'
            mute_sounds()
    elif pygame.mouse.get_pressed() == (0, 0, 0):
        button_pressed = False
```

This code adds function to the button. When the button is clicked the image will change to show that it has been pressed and the text on the button will switch to ‘UNMUTE’. Also when the button is pressed the function ‘mute_sounds()’ is called which mutes all games sounds and unmutes them to their originally set volumes when the mute button is pressed again.

```
sound_group = [GUN_SOUND, GUN_SOUND2, FOOTSTEPS, ENEMY_DEATH_SOUND, GAME_OVER, HITMARKER, HURT,
RELOAD_SOUND,
               ENERGY_SOUND, STRENGTH_SOUND, SPEED_BOOTS_SOUNDS, BUTTON_CLICK_SOUND]


def mute_sounds():
    if mute_audio:
        for sound in sound_group:
            sound.set_volume(0)
    elif not mute_audio:
        GUN_SOUND.set_volume(0.3)
        GUN_SOUND2.set_volume(0.15)
        FOOTSTEPS.set_volume(0.15)
        ENEMY_DEATH_SOUND.set_volume(0.4)
        HITMARKER.set_volume(0.3)
        HURT.set_volume(0.2)
        GAME_OVER.set_volume(1)
        RELOAD_SOUND.set_volume(1)
```

```
        ENERGY_SOUND.set_volume(1)
        STRENGTH_SOUND.set_volume(1)
        SPEED_BOOTS_SOUNDS.set_volume(1)
        BUTTON_CLICK_SOUND.set_volume(0.5)
```

To make this function, i created a list of all sounds so they can be muted at once and once the mute button is toggled again, they are set back to their original values which were initially set in the assets file.

The mute button now works correctly and displays the correct text when clicked.

```
quit_button = pygame.Rect(300, 530, 420, 100)
quit_text = FONT.render('QUIT', True, WHITE)
if quit_button_pressed:
    quit_button_image = BUTTON_PRESSED
else:
    quit_button_image = BUTTON_IDLE
surface.blit(quit_button_image, (quit_button.x, quit_button.y))
surface.blit(quit_text, (WIDTH / 2 - (50), 550))

if quit_button.collidepoint(mouse_pos):
    if pygame.mouse.get_pressed() == (1, 0, 0):
        BUTTON_CLICK_SOUND.play()
        if not quit_button_pressed:
            quit_button_pressed = True
            pygame.quit()
            sys.exit()
```

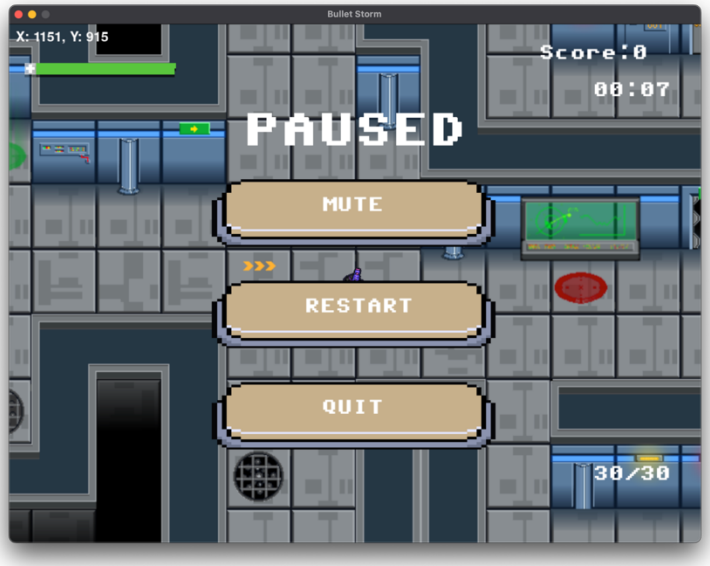
I used the same logic to create a quit button which quits the game when pressed.


```
restart_button = pygame.Rect(300, 380, 420, 100)
restart_text = FONT.render('RESTART', True, WHITE)
if restart_button_pressed:
    restart_button_image = BUTTON_PRESSED
else:
    restart_button_image = BUTTON_IDLE
surface.blit(restart_button_image, (restart_button.x, restart_button.y))
surface.blit(restart_text, (WIDTH / 2 - (75), 400))

if restart_button.collidepoint(mouse_pos):
    if pygame.mouse.get_pressed() == (1, 0, 0):
        BUTTON_CLICK_SOUND.play()
        if not restart_button_pressed:
            restart_button_pressed = True
            time.sleep(0.5)
            restart_game()
    elif pygame.mouse.get_pressed() == (0, 0, 0):
        restart_button_pressed = False
```

Finally, I created the restart button which restarts the game when pressed.

```
def restart_game():
    global player, enemies, score, start_time, paused


    for enemy in EnemyGroup:
        enemy.kill()


    player.score = 0
    player.ammo = MAX_AMMO
    player.pos = pygame.math.Vector2(START_X, START_Y)
    paused = not paused
    start_time = pygame.time.get_ticks()
```

The restart function resets all the variables such as the players score, the time survived and player ammo back to default and sets the players position to the start position. It also kills all enemies and by doing this it puts you back to the exact same position someone who just started would be in.



Now I am going to work on a log in screen where the player can either sign up or login into their account. The purpose of this is so that the player's high score can be saved and allow scores of different players to be displayed with their name next to it on a leaderboard.

```
from datetime import datetime
import hashlib
import json
import os
```

Imported these 4 modules as they will be needed for the code ahead.

```
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
```

Firstly, using the hashlib module, I created a function that will hash the users passwords using SHA256.

```
def username_exists(username):
    return os.path.exists(f"{username.lower()}.json")
```

Next, I created a function that checks if a JSON file in the current directory contains a username that a new user is trying to use. Usernames are converted to lowercase to avoid any mismatches. This function's main purpose is to ensure that two users with the same name cannot both create an account.

```
def create_user(username, password):
    user_data = {
        "username": username,
        "password": hash_password(password)
   }
    with open(f"{username.lower()}.json", "w") as file:
        json.dump(user_data, file)
```

This function's purpose is to store the user's information and the hashed version of their password into a JSON file.

```
def validate_login(username, password):
    if not username_exists(username):
        return False
    with open(f"{username.lower()}.json", "r") as file:
        user_data = json.load(file)
    return user_data["password"] == hash_password(password)
```

This function will be used to validate the log in for the user. First it calls the username exists function to verify that a file with that username exists and then next it compares the hashed value of the password entered by the user with the value stored in the file.

```
def get_text_input(prompt, x, y):
    text = ''
    input_box = pygame.Rect(x, y, 200, 50)
    color_inactive = pygame.Color(WHITE)
    color_active = pygame.Color(0, 0, 255)
    color = color_inactive
    active = False
```

The function above creates an input box which changes colour when selected.

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if input_box.collidepoint(event.pos):
                active = not active
            else:
                active = False
            color = color_active if active else color_inactive

        if event.type == pygame.KEYDOWN:
            if active:
                if event.key == pygame.K_RETURN:
                    return text
                elif event.key == pygame.K_BACKSPACE:
                    text = text[:-1]
                else:
                    text += event.unicode
```

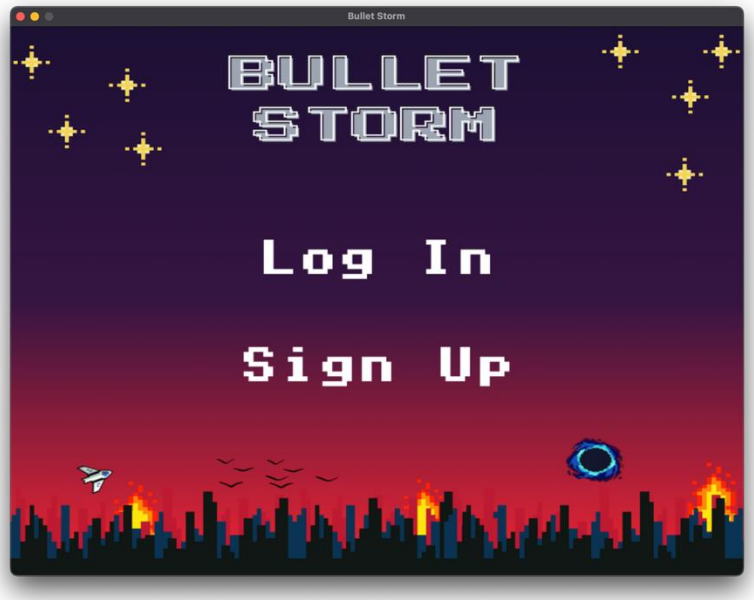
Within the function there is an event handling loop which looks out for when the box is clicked and sets the variable active to True so the player can start typing in the box, press backspace to delete text and enter to confirm details. Currently I cannot view this box as I have not called it into the main loop yet.

```
def sign_in_page(window):
    while True:
        window.fill((0, 0, 0))

        login_text = FONT.render("Log In", True, WHITE)
        signup_text = FONT.render("Sign Up", True, WHITE)

        window.blit(login_text, (WIDTH / 2 - login_text.get_width() / 2, HEIGHT / 2 - 100))
        window.blit(signup_text, (WIDTH / 2 - signup_text.get_width() / 2, HEIGHT / 2 - 50))

        pygame.display.update()
```

I have now created the function which will display these screens. Right now, this function displays two options for the player when they start the game.

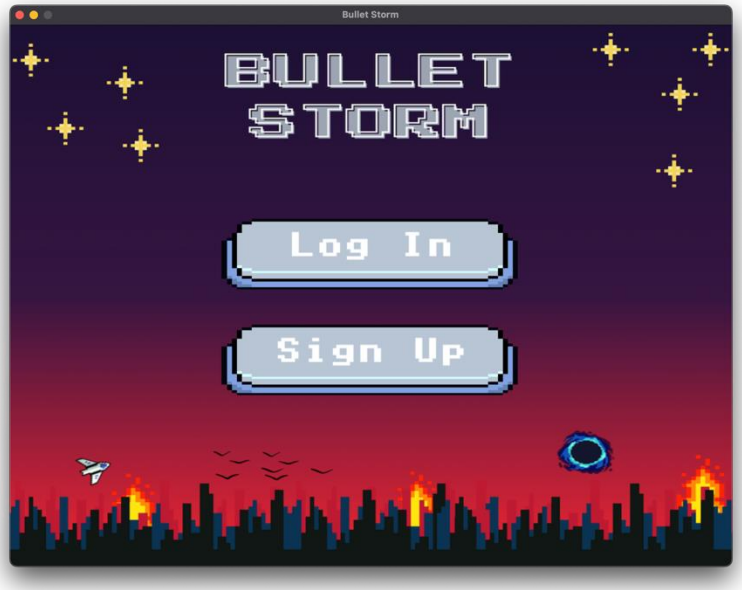

```
def sign_in_page(window):
    while True:
        window.fill((0, 0, 0))
        window.blit(BLANK_MENU, (0, 0))

        login_text = LARGE_FONT.render("Log In", True, WHITE)
        signup_text = LARGE_FONT.render("Sign Up", True, WHITE)

        window.blit(login_text, (WIDTH / 2 - login_text.get_width() / 2, HEIGHT / 2 - 100))
        window.blit(signup_text, (WIDTH / 2 - signup_text.get_width() / 2, HEIGHT / 2 + 50))

        pygame.display.update()
```

I adjusted the code so that the font appears larger, and the background is more appealing and matches the theme of the game.



Next, I will create buttons under the font.

```
sign_up_button = pygame.Rect(300, 275, 420, 100)
log_in_button = pygame.Rect(300, 425, 420, 100)

window.blit(BUTTON_IDLE, (sign_up_button.x, sign_up_button.y))
window.blit(BUTTON_IDLE, (log_in_button.x, log_in_button.y))
```




```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
```

```
    if event.type == pygame.MOUSEBUTTONDOWN:
        if log_in_button.collidepoint(mouse_pos):
            username = get_text_input("Enter username", 200, 275)
            player.name = username
            password = get_text_input("Enter password", 200, 275)
            if validate_login(username, password):
                return username
            else:
pass
```

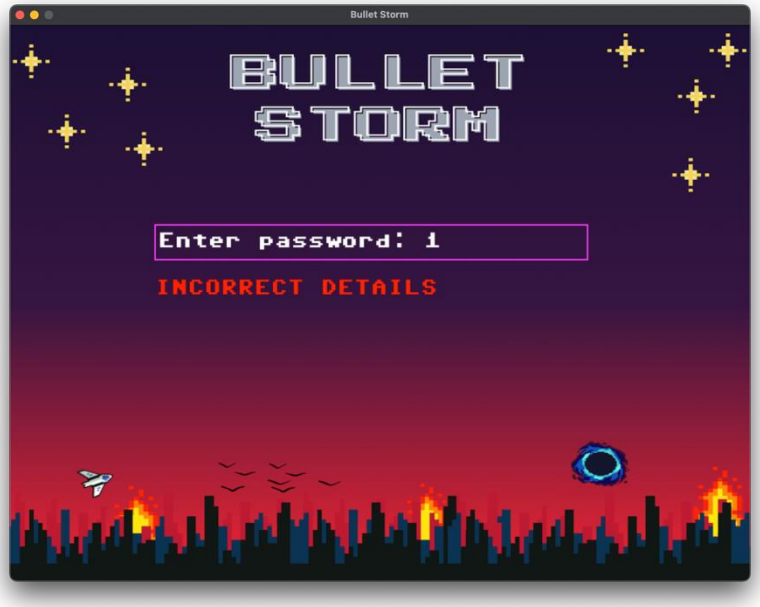
The buttons are now interactable and if you click them, you will be taken to the input text screen where you can enter your information into a text box and the appropriate functions from earlier are used to validate the information.

However if the correct information is not entered, there is no error message informing the player something went wrong. To fix this problem, i will add text on the screen displaying the correct message.

```
error_text1 = FONT.render('INCORRECT DETAILS', True, 'red')
```

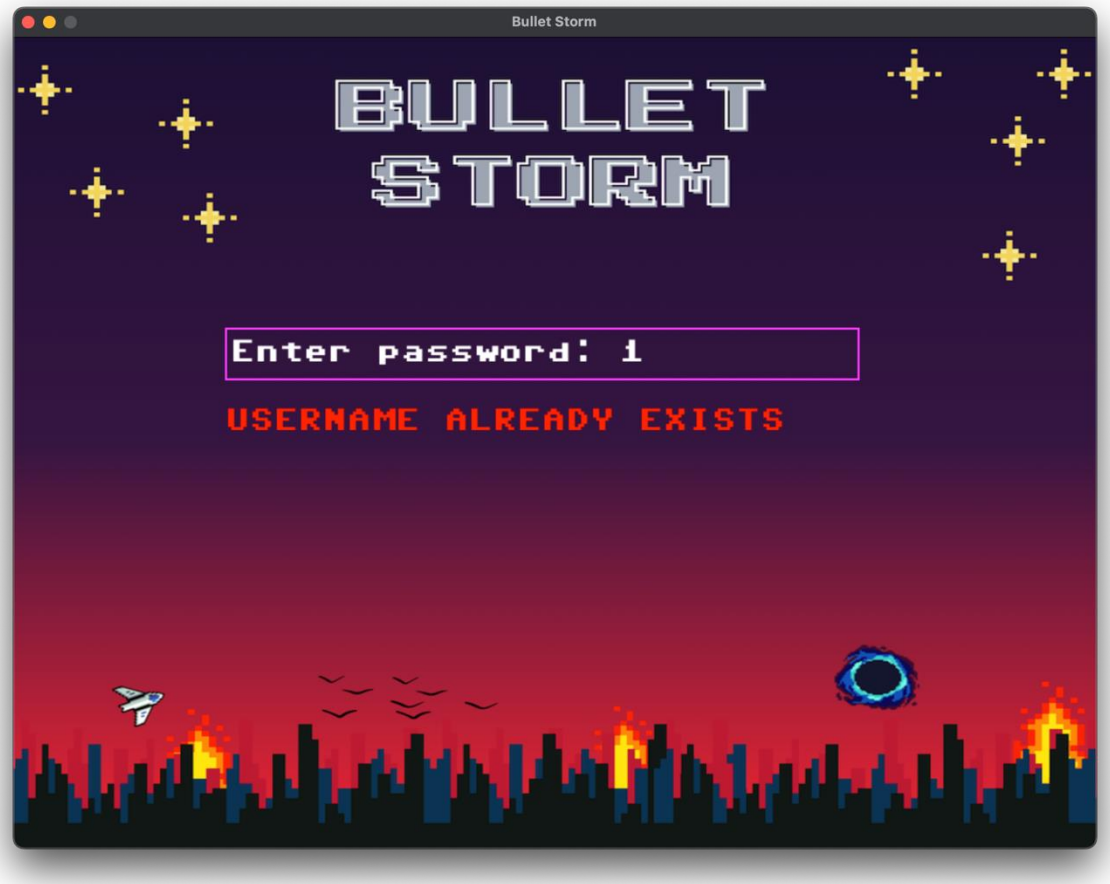
```
else:
    window.blit(error_text1, (200, 345))
    pygame.display.update()
    time.sleep(3)
```

Now a message appears for 3 seconds and then sends you back to the log in page when details are entered incorrectly.

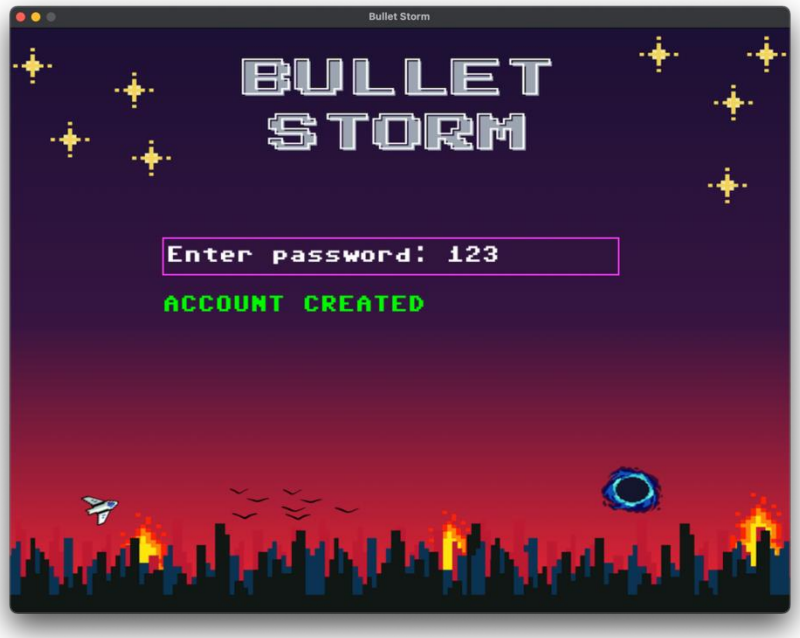



```
if event.type == pygame.MOUSEBUTTONDOWN:
    if sign_up_button.collidepoint(mouse_pos):
        username = get_text_input("Enter username", 200, 275)
        password = get_text_input("Enter password", 200, 275)
        if username_exists(username):
            window.blit(error_text2, (200, 345))
            pygame.display.update()
            time.sleep(3)
        else:
            create_user(username, password)
            window.blit(success_text, (200, 345))
            pygame.display.update()
            time.sleep(3)
            return username
```

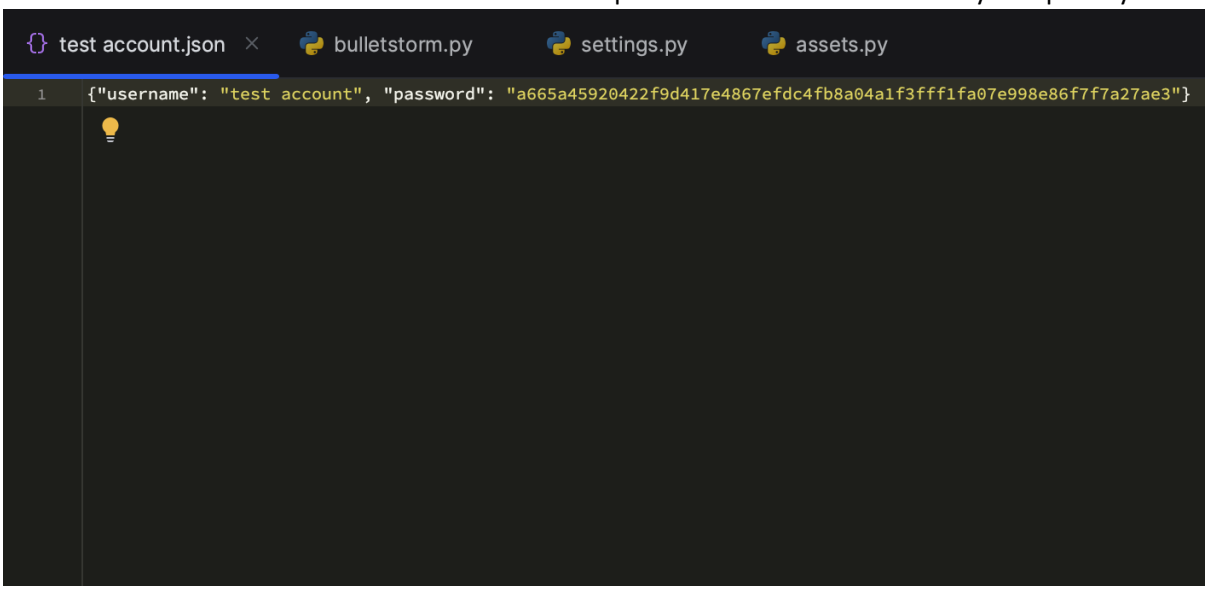
To finish off, I repeated the same steps but for the sign in page and called the different corresponding functions for it. If the username already exists the message below will display.



And if an account is created successfully, this is shown below.



This is then stored in a JSON file in this format. The password is hashed for security and privacy.




```
def save_score(self):
    data = {
        "score": self.score,
        "player": self.name,
        "date": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    }
    try:
        with open("scores.json", "r") as file:
            scores = json.load(file)
    except FileNotFoundError:
        scores = []

    scores.append(data)
    with open("scores.json", "w") as file:
        json.dump(scores, file, indent=4)

def load_scores(self):
    try:
        with open("scores.json", "r") as file:
            scores = json.load(file)
        return sorted(scores, key=lambda x: x["score"], reverse=True)
    except FileNotFoundError:
        return []

def display_leaderboard(self, surface):
    scores = self.load_scores()
    top_scores = scores[:5]
   font = pygame.font.Font(None, 36)

    y_offset = 400
    for rank, entry in enumerate(top_scores, start=1):
        score_text = FONT.render(f"{rank}. {entry['player']} - {entry['score']} ", True,
                                 (255, 255, 255))
        surface.blit(score_text, (WIDTH / 2 - score_text.get_width() / 2, y_offset))
        y_offset += 40
```

Above I have borrowed code from stack overflow and tweaked it so that it fits my program. This code saves every player's score, puts them in a JSON file, then sorts them from highest to lowest and then displays them in a leaderboard format. In the player class, i previously created a player name instance which is automatically set to the username of the player when they log in or create their account.

```
player.name = username
```

This username is then displayed next to the score on the leaderboard. Only the top 5 scores are displayed. The display leaderboard function is called when the game over screen is displayed, and the leaderboard is drawn onto it.

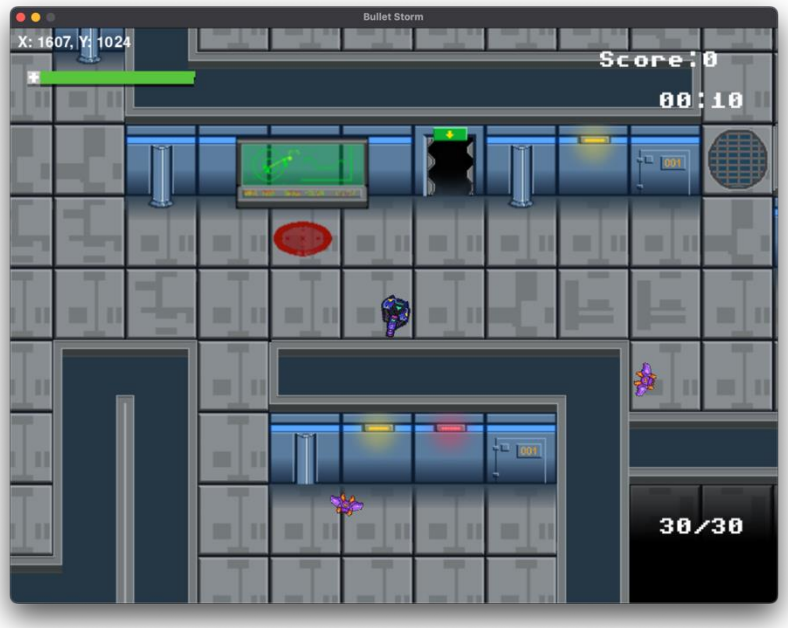
```
if self.dead:
    self.save_score()
    window.fill((0, 0, 0))
    window.blit(self.game_over_image, (0, 0))
    surface.blit(score_message, (WIDTH / 2 - (score_message.get_width() / 2), HEIGHT / 2 +
(score_message.get_height() / 2)-150))
    surface.blit(time_message, (WIDTH / 2 - (time_message.get_width() / 2), HEIGHT / 2 +
(time_message.get_height() / 2)-85))
    self.display_leaderboard(surface)
```


```
class Enemy2(Enemy):
    def __init__(self, position):
        super().__init__(position)

       self.frames = [ENEMY_2_FRAME_1.convert_alpha(),
                      ENEMY_2_FRAME_2.convert_alpha(),
```

```
                        ENEMY_2_FRAME_3.convert_alpha(),
                        ENEMY_2_FRAME_4.convert_alpha(),]
        self.damage = 5
        self.speed = 2
```

I now created a new enemy class which inherits from the main enemy class but has a different image and double the speed but half of the damage of the original enemy. For it to spawn correctly, I modified the Enemy Spawner class to randomly pick between what enemy to spawn.

```
enemy_type = random.choice(["Enemy", "Enemy2"])

if enemy_type == "Enemy":
    new_enemy = Enemy(spawn_point)
elif enemy_type == "Enemy2":
    new_enemy = Enemy2(spawn_point)

EnemyGroup.add(new_enemy)
```



The new enemy is now spawning correctly.

```
class Enemy3(Enemy):
    def __init__(self, position):
        super().__init__(position)

        self.frames = [ENEMY_3_FRAME_1.convert_alpha(),
                       ENEMY_3_FRAME_2.convert_alpha(),
                       ENEMY_3_FRAME_3.convert_alpha(),
                       ENEMY_3_FRAME_4.convert_alpha(),
                       ENEMY_3_FRAME_5.convert_alpha(),
                       ENEMY_3_FRAME_6.convert_alpha(),
                       ENEMY_3_FRAME_7.convert_alpha(),
                       ENEMY_3_FRAME_8.convert_alpha()]
```

```
        self.damage = 20
        self.speed = 1
        self.health = 150
```

Another enemy was created with the same logic. This one has double damage and 50 percent more health.


```
ENEMY_SPAWN_TIME = random.randint(5000, 20000)
```

I changed the logic for the enemy spawn time so that it picks a random time between 5 and 20 seconds.

```
def increase_difficulty():
    global DIFFICULTY_MULTIPLIER, difficulty_start_time
    current_time = pygame.time.get_ticks()

    if current_time - difficulty_start_time >= 60000:
        DIFFICULTY_MULTIPLIER += 0.25
        difficulty_start_time += 60000
```

This new function updates the value of a variable that is used to increase the difficulty of the enemy, the longer the player is alive to add more challenge. The difficulty multiplier variable is multiplied onto the enemies' speed, health and damage.

```
self.speed = ENEMY_SPEED * DIFFICULTY_MULTIPLIER
self.health = ENEMY_HEALTH * DIFFICULTY_MULTIPLIER
self.damage = ENEMY_DAMAGE * DIFFICULTY_MULTIPLIER
```

I also divided the time between spawns by it so that the spawn rate for enemies increases.

```
ENEMY_SPAWN_TIME = random.randint(5000, 20000/DIFFICULTY_MULTIPLIER)
```

```
if current_time - difficulty_start_time >= 60000 and difficulty_start_time <= 36000:
    DIFFICULTY_MULTIPLIER += 0.25
    difficulty_start_time += 60000
```

Then finally i updated the logic in the increase function so that after 6 minutes the difficulty will stop increasing to avoid it being impossible for the player to stay alive.

```
TypeError: 'float' object cannot be interpreted as an integer
```

I ran the code and received this error. Because in my randit function for spawn times, I divided an integer by a float and the function only works for integers.

Upon testing the gameplay, I realised the enemies spawn very slowly even in the later minutes of the game and all enemies do the same damage regardless of what i set their individual damage to be.

```
ENEMY_SPAWN_TIME = random.randint(int(5000/DIFFICULTY_MULTIPLIER), int(20000/DIFFICULTY_MULTIPLIER))
```

To fix the first problem, I decreased the lower bound as well so that there does not have to be a minimum 5 second gap between enemy spawns.

```
class EnemyBullet(pygame.sprite.Sprite):
    def __init__(self, x, y, angle, damage):
```

To fix the second problem, i added a new parameter in the EnemyBullet class and set it to self.damage for the bullet.

```
self.damage = damage
```

Next in the enemy class i created an instance for the damage.

```
self.damage = ENEMY_DAMAGE
```

And finally passed through the value through the bullet instantiation and the enemies are now dealing the correct damage.

```
new_bullet = EnemyBullet(self.rect.centerx, self.rect.centery, angle, self.damage)
```

I added some more final adjustments to the characters so that they are more unique and have different sounds.

```
class Enemy2(Enemy):
    def __init__(self, position):
        super().__init__(position)

        self.frames = [ENEMY_2_FRAME_1.convert_alpha(),
                       ENEMY_2_FRAME_2.convert_alpha(),
                       ENEMY_2_FRAME_3.convert_alpha(),
                       ENEMY_2_FRAME_4.convert_alpha(),]
        EnemyBullet.damage = 3.0 * DIFFICULTY_MULTIPLIER
        self.speed = 2.5 * DIFFICULTY_MULTIPLIER
        self.shoot_cooldown = 300
        self.health = 75.0
        self.gun_sound = GUN_SOUND4

class Enemy3(Enemy):
    def __init__(self, position):
        super().__init__(position)

        self.frames = [ENEMY_3_FRAME_1.convert_alpha(),
                       ENEMY_3_FRAME_2.convert_alpha(),
                       ENEMY_3_FRAME_3.convert_alpha(),
                       ENEMY_3_FRAME_4.convert_alpha(),
                       ENEMY_3_FRAME_5.convert_alpha(),
                       ENEMY_3_FRAME_6.convert_alpha(),
                       ENEMY_3_FRAME_7.convert_alpha(),
                       ENEMY_3_FRAME_8.convert_alpha()]
        self.damage = 25.0 * DIFFICULTY_MULTIPLIER
        self.health = 150.0 * DIFFICULTY_MULTIPLIER
        self.shoot_cooldown = 3000
        self.gun_sound = GUN_SOUND3
```

I still had one problem of enemies being able to see player through walls and start shooting.

```
def line_of_sight(self, player):

    player_vector = pygame.math.Vector2(player.rect.center)
    enemy_vector = pygame.math.Vector2(self.rect.center)

    step = 5

    direction = (player_vector - enemy_vector).normalize()
    distance = self.enemy_distance(player_vector, enemy_vector)

    for i in range(0, int(distance), step):
        point = enemy_vector + direction * i

        for boundary in map_boundaries.boundaries:
            if boundary.collidepoint(point):
                return False

   return True
```

To fix this problem I used code from a YouTube tutorial by 'Coding with Russ' and tweaked it for my program. This code uses a line-of-sight logic. It draws a line between enemies and players and if any obstacles are in the way, such as the map boundary rectangles I created, False will be returned meaning that they will not see the player.

```
def rotate_towards_player(self, player):
    dx = player.rect.centerx - self.rect.centerx
    dy = player.rect.centery - self.rect.centery
    angle = math.degrees(math.atan2(dy, dx))

    if not self.idle and self.line_of_sight(player):
```

```
def shoot_player(self, player):
    current_time = pygame.time.get_ticks()



    if not self.idle and self.line_of_sight(player):
```

```
def follow_player(self):
    player_vector = pygame.math.Vector2(player.hitbox_rect.center)
    enemy_vector = pygame.math.Vector2(self.rect.center)
    distance = self.enemy_distance(player_vector, enemy_vector)
    self.check_idle(player)

    if 70 < distance and not self.idle and self.line_of_sight(player):
```

I added 'and self.line_of_sight(player)' to all these methods and now the enemies can no longer see the player through walls.

The enemy is now not facing towards the player or shooting even though I am close enough as I am standing behind a wall.

Now I am going to work on the Main Menu UI.

```
MAIN_MENU = "main_menu"
SIGN_IN = "sign_in"
GAME = "game"
SETTINGS = "settings"
STATS = "stats"
current_state = SIGN_IN
```

To start off, I declared the 5 menus the game can be in. The first menu you will be in is the sign-up page.

```
def MainMenu(window):
    global current_state
    window.fill((0, 0, 0))
    window.blit(MENU_1, (0, 0))

    START_BUTTON = pygame.Rect(380, 460, 260, 90)
    SETTINGS_BUTTON = pygame.Rect(75, 470, 210, 78)
    STATS_BUTTON = pygame.Rect(735, 470, 210, 78)

    pygame.display.update()
    mouse_pos = pygame.mouse.get_pos()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if START_BUTTON.collidepoint(mouse_pos):
                current_state = GAME
        if event.type == pygame.MOUSEBUTTONDOWN:
            if SETTINGS_BUTTON.collidepoint(mouse_pos):
                current_state = SETTINGS
        if event.type == pygame.MOUSEBUTTONDOWN:
            if STATS_BUTTON.collidepoint(mouse_pos):
                current_state = STATS
```

Next, I create the MainMenu function where if you click a button, you will be taken to a corresponding state.

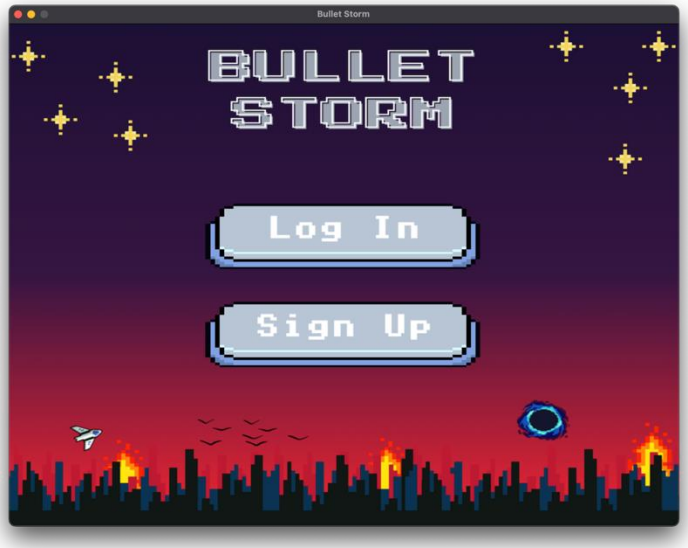
```
if validate_login(username, password):
    current_state = MAIN_MENU
    return username
```

```
else:
    create_user(username, password)
    player.name = username
    window.blit(success_text, (200, 345))
    pygame.display.update()
    time.sleep(3)
    current_state = MAIN_MENU
    return username
```

Then I updated the logic so if you log in or sign up successfully, you will be taken to the main menu screen.

```
def game_loop():
    global paused, start_time, current_state



    while True:
        if current_state == SIGN_IN:
            username = sign_in_page(window)
        elif current_state == MAIN_MENU:
            MainMenu(window)
        elif current_state == GAME:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
```

I then modified the game loop so that the function for each state is called when the state changes allowing for it to transition in between.



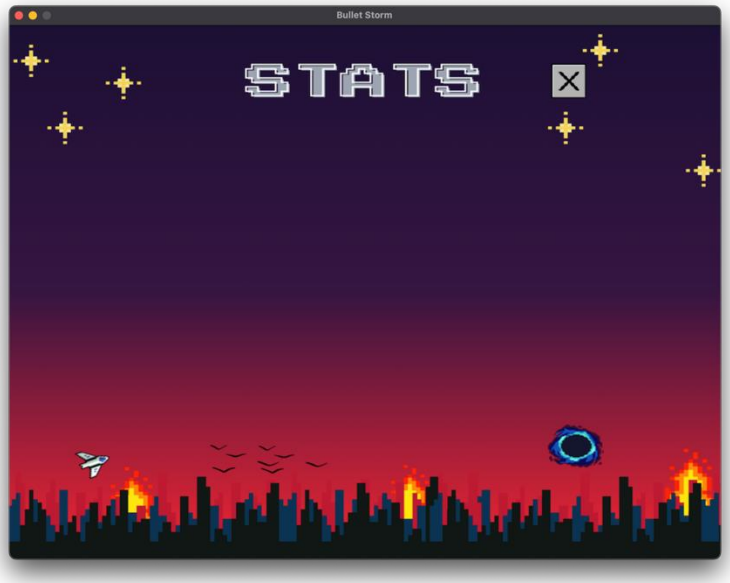
First the login in screen is displayed and after you have logged in or signed up, you are taken to the main menu screen.

If you click the start button, you will be taken into the main game and the game will start. Now I will create the menus for settings and stats.

```
def StatsMenu(window):
    global current_state
    window.fill((0, 0, 0))
    window.blit(MENU_2, (0, 0))
    EXIT_BUTTON = pygame.Rect(700, 55, 100, 100)
    window.blit(CROSS_IMAGE.convert_alpha(), (EXIT_BUTTON.x, EXIT_BUTTON.y))


    pygame.display.update()
    mouse_pos = pygame.mouse.get_pos()


    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()


        if event.type == pygame.MOUSEBUTTONDOWN:
            if EXIT_BUTTON.collidepoint(mouse_pos):
                current_state = MAIN_MENU
```

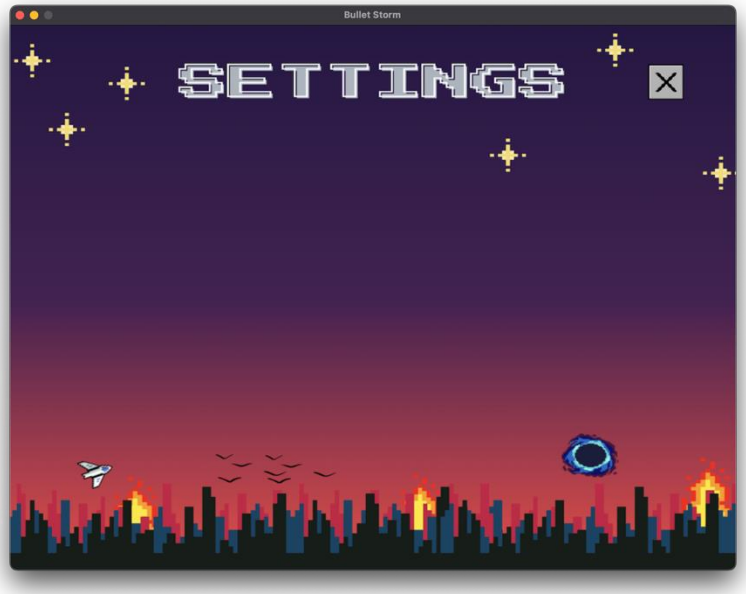
This function is for the stats menu. It contains one interactable button that will take you back to the main menu after you have viewed your stats.

```
elif current_state == STATS:
    StatsMenu(window)
```




```
def SettingsMenu(window):
    global current_state
    window.fill((0, 0, 0))
    window.blit(MENU_3, (0, 0))
    EXIT_BUTTON = pygame.Rect(900, 55, 100, 100)
    window.blit(CROSS_IMAGE.convert_alpha(), (EXIT_BUTTON.x, EXIT_BUTTON.y))


    pygame.display.update()
    mouse_pos = pygame.mouse.get_pos()


    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()


        if event.type == pygame.MOUSEBUTTONDOWN:
            if EXIT_BUTTON.collidepoint(mouse_pos):
                current_state = MAIN_MENU
```

This function is for the settings menu.



The first button in the settings menu will be a colour-blind button.

```
def apply_deuteranopia_filter(screen):
    pixels = pygame.surfarray.pixels3d(screen)
    pixels[:, :, 1] = 0



def apply_protanopia_filter(surface):
    pixels = pygame.surfarray.pixels3d(surface)
    pixels[:, :, 0] = 0



def apply_tritanopia_filter(surface):
    pixels = pygame.surfarray.pixels3d(surface)
    pixels[:, :, 2] = 0
```

First, I used ChatGPT to generate me 3 functions that apply a colour-blind filter onto the screen. This is done by changing the pixel colours.

```
color_blind_list = ['None','Deuteranopia','Protanopia','Tritanopia' ]
pointer = 0
```

Next, I created a list which will be used to cycle between the colour-blind names when clicked and a pointer to access the values.

```
COLOUR_BLIND_BUTTON = pygame.Rect(300, 170, 420, 100)
COLOUR_BLIND_BUTTON_TEXT = FONT.render(color_blind_list[pointer], True, WHITE)

if COLOUR_BLIND_BUTTON_PRESSED:
    COLOUR_BLIND_BUTTON_IMAGE = BUTTON_PRESSED
else:
    COLOUR_BLIND_BUTTON_IMAGE = BUTTON_IDLE
window.blit(COLOUR_BLIND_BUTTON_IMAGE, (COLOUR_BLIND_BUTTON.x, COLOUR_BLIND_BUTTON.y))
```

```
window.blit(COLOUR_BLIND_BUTTON_IMAGE, (COLOUR_BLIND_BUTTON.x, COLOUR_BLIND_BUTTON.y))
window.blit(COLOUR_BLIND_BUTTON_TEXT, (WIDTH/2 - 120, 190))
```

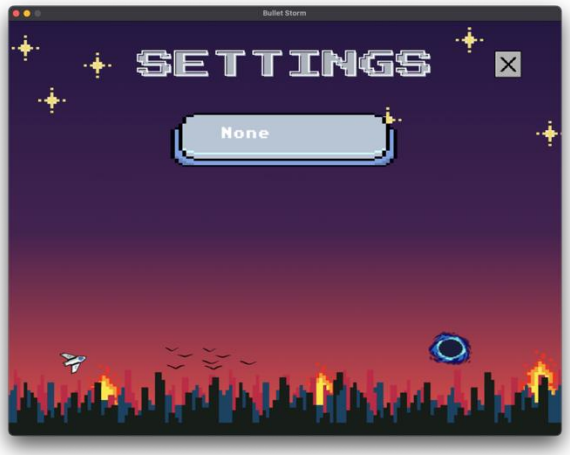
Then I drew on the text and button to the screen. The text will display the corresponding list value.

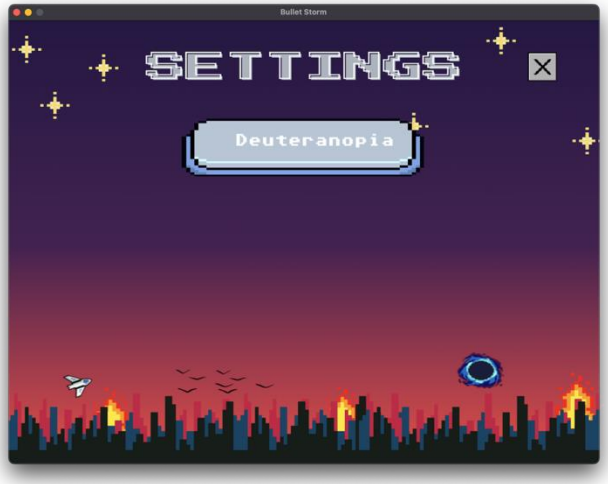
```
if COLOUR_BLIND_BUTTON.collidepoint(mouse_pos):
        COLOUR_BLIND_BUTTON_PRESSED = not COLOUR_BLIND_BUTTON_PRESSED  # Toggle button pressed state


if event.type == pygame.MOUSEBUTTONUP:
    if COLOUR_BLIND_BUTTON.collidepoint(mouse_pos):
        COLOUR_BLIND_BUTTON_PRESSED = False
        if pointer < len(color_blind_list)- 1:
            pointer += 1
        else:
            pointer = 0
```

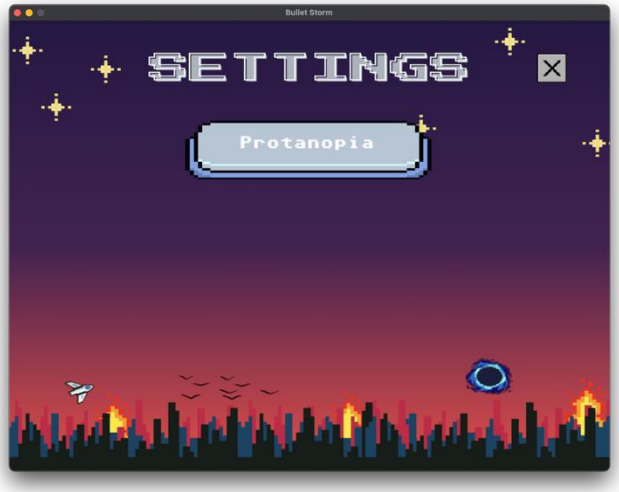
When the button is pressed, the pointer value will increase meaning that the text will change each time when clicked.

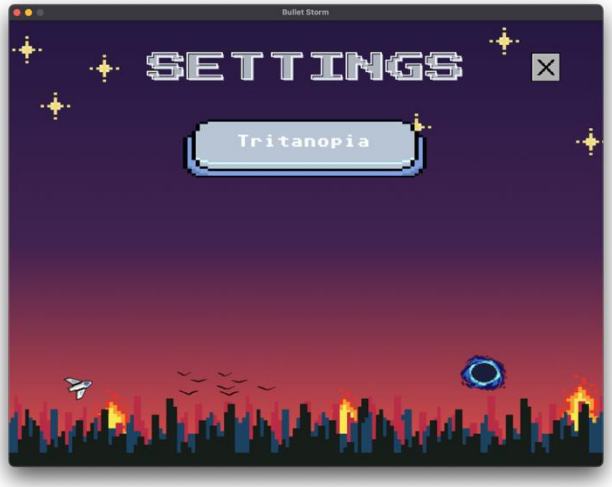
```
if pointer == 1:
    apply_deuteranopia_filter(window)
elif pointer == 2:
    apply_protanopia_filter(window)
elif pointer == 3:
    apply_tritanopia_filter(window)
else:
    pass
```

Finally in the main loop i made it that the pointer value represents each colour-blind mode selected and will call the appropriate function to change the colours.



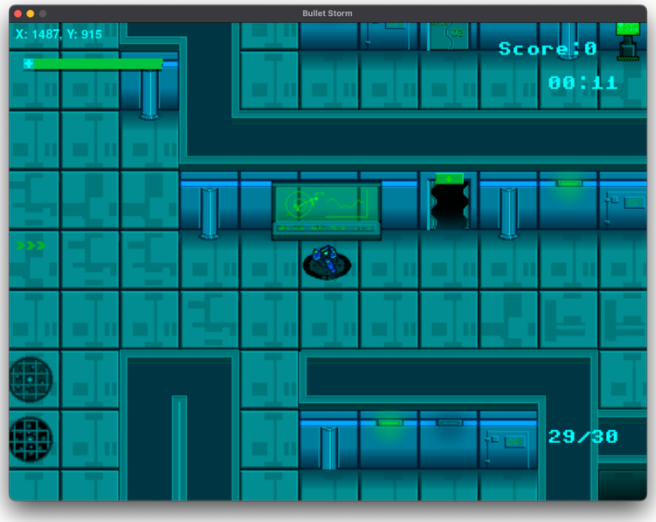

Bullet Storm
SETTINGS
Deuteranopia


Bullet Storm
SETTINGS
Protanopia


Bullet Storm
SETTINGS
Tritanopia

The button changes each time its clicked and loops back to None at the end.


Bullet Storm
X: 1487, Y: 915
Score:120
02:30
21/30


Bullet Storm
X: 1487, Y: 915
Score:0
00:11
29/30

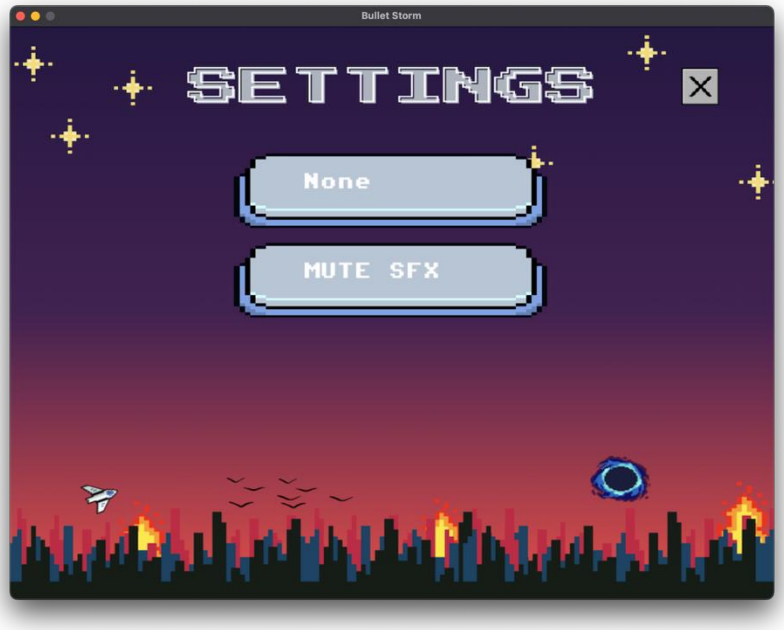

Bullet Storm
X: 1487, Y: 915
Score:0
00:10
29/30

```
MUTE_SFX_BUTTON = pygame.Rect(300, 290, 420, 100)
mute_sfx_text = FONT.render(mute_sfx_message, True, WHITE)
if mute_button_pressed:
    mute_button_image = BUTTON_PRESSED
else:
    mute_button_image = BUTTON_IDLE
window.blit(mute_button_image, (MUTE_SFX_BUTTON.x, MUTE_SFX_BUTTON.y))
window.blit(mute_sfx_text, (WIDTH / 2 - 120, MUTE_SFX_BUTTON.y + 20))
```

The code above draws the mute sound effects button on to the screen.

```
if MUTE_SFX_BUTTON.collidepoint(mouse_pos):
    if pygame.mouse.get_pressed() == (1, 0, 0):
        BUTTON_CLICK_SOUND.play()
        if not mute_button_pressed:
            mute_button_pressed = True
            mute_audio = not mute_audio
            mute_sfx_message = 'UNMUTE SFX' if mute_sfx_message == 'MUTE SFX' else 'MUTE SFX'
            mute_sounds()
    elif pygame.mouse.get_pressed() == (0, 0, 0):
        mute_button_pressed = False
```

If you click the button, its image will change and it the text on it will switch to unmute. It also called the mute sounds function which will mute all sound effects in game and unmute them if the button is pressed again.



The button works correctly and mutes the sound effects.

Next, I will create the mute music button however I need to add music to the game first. One for the menu and one for the game.

```
def game_loop():
    global paused, start_time, current_state

    if (current_state == MAIN_MENU or current_state == STATS or current_state == SETTINGS or
            current_state == SIGN_IN):
        MAIN_MENU_MUSIC.play()
```

The code above plays the main menu music as soon as the program starts. This music will play when the code is not in game state.

```
def game_loop():
    global paused, start_time, current_state

    if current_state != GAME:
        MAIN_MENU_MUSIC.play()
```

I updated the code to simplify it more.

```
def play_game_music():
    global current_state
    if current_state == GAME and game_music_playing:
        GAME_MUSIC_AUDIO.play()
```

This function is to play the game music.

```
elif current_state == GAME:
    MAIN_MENU_MUSIC.stop()
    play_game_music()
```

Finally, I called it in the games while loop and stopped the menu music as soon as the actual game starts. I ran the code, and the music was working correctly, however all game sound effects could no longer be heard, and the game music sound quality was off. This is because I called it in the while loop, so the music keeps on playing repeatedly.

```
game_music_playing = True

def play_game_music():
    global current_state, game_music_playing, music_time
    if current_state == GAME and game_music_playing:
        GAME_MUSIC_AUDIO.play(100)
        game_music_playing = False
```

To fix this problem I added a flag that is set to false as soon as game music starts playing so that in the while lop it can only play once. I also set the game music audio to 100 loops so as soon as the song finishes, it loops and plays again. The sound effects can now be correctly heard.

```
music_muted = False

def mute_music():
    global  music_muted
    if music_muted:
        MAIN_MENU_MUSIC.set_volume(0)
        GAME_MUSIC_AUDIO.set_volume(0)
    elif not music_muted:
        MAIN_MENU_MUSIC.set_volume(1)
        GAME_MUSIC_AUDIO.set_volume(0.1)
```

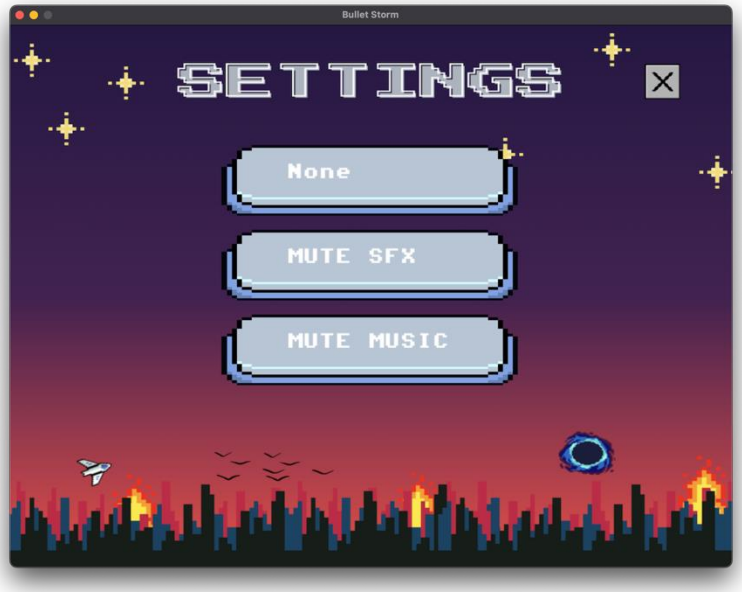
I then created a function that mutes and unmutes the audio depending on if the button was pressed or not.

```
MUTE_MUSIC_BUTTON = pygame.Rect(300, 410, 420, 100)
mute_music_text = FONT.render(mute_music_message, True, WHITE)
if MUTE_MUSIC_BUTTON_PRESSED:
    mute_music_button_image = BUTTON_PRESSED
else:
    mute_music_button_image = BUTTON_IDLE
window.blit(mute_music_button_image, (MUTE_MUSIC_BUTTON.x, MUTE_MUSIC_BUTTON.y))
window.blit(mute_music_text, (WIDTH / 2 - 120, MUTE_MUSIC_BUTTON.y + 20))
```

Next, I drew the button onto the screen.

```
if MUTE_MUSIC_BUTTON.collidepoint(mouse_pos):
    if pygame.mouse.get_pressed() == (1, 0, 0):
        BUTTON_CLICK_SOUND.play()
        if not MUTE_MUSIC_BUTTON_PRESSED:
            MUTE_MUSIC_BUTTON_PRESSED = True
            music_muted = not music_muted
            mute_music_message = 'UNMUTE MUSIC' if mute_music_message == 'MUTE MUSIC' else 'MUTE MUSIC'
            mute_music()
    elif pygame.mouse.get_pressed() == (0, 0, 0):
        MUTE_MUSIC_BUTTON_PRESSED = False
```

Finally, I write the logic for the button which the exact same for the sound effects one but instead calls the mute music function.



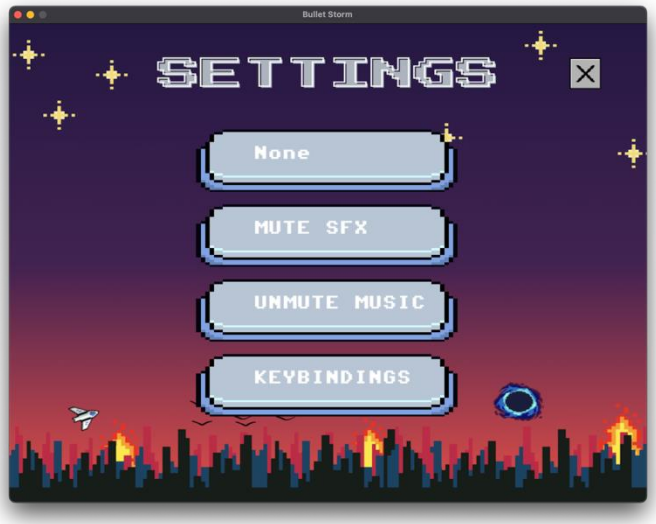
The button is being displayed and functioning correctly.

The final button will be the keybinding buton. This button will take you to a key binding menu which will be a separate state in the game.

```
KEYBINDING = "keybinding"
```

```
KEYBIND_BUTTON = pygame.Rect(300, 530, 420, 100)
keybind_button_text = FONT.render('KEYBINDINGS', True, WHITE)
window.blit(BUTTON_IDLE, (KEYBIND_BUTTON.x, KEYBIND_BUTTON.y))
window.blit(keybind_button_text, (WIDTH / 2 - 120, KEYBIND_BUTTON.y + 20))
```

Next i drew the button.




```
if event.type == pygame.MOUSEBUTTONDOWN:
    if KEYBIND_BUTTON.collidepoint(mouse_pos):
        current_state = KEYBINDING
```

When you click the button the game state will change to the key binding menu.

To create the key binding menu, I first need to create a dictionary for the key binds.

```
keybinds = {
    "move_left": pygame.K_a,
    "move_right": pygame.K_d,
    "move_up": pygame.K_w,
    "move_down": pygame.K_s,
    "shoot": pygame.K_SPACE,
    "reload": pygame.K_r,
    "pause": pygame.K_p
}
```

This dictionary stores all actions next to the key needed to be pressed.

```
def KeybindsMenu(window):
    global keybinds, current_state, pressed, mouse_pos, current_state
    window.fill((0, 0, 0))
    window.blit(MENU_3, (0, 0))


    actions = ["move_left", "move_right", "move_up", "move_down", "shoot", "reload", "pause"]
    labels = ["Move Left", "Move Right", "Move Up", "Move Down", "Shoot", "Reload", "Pause"]


    key_rects = []
```

Next, I created the function for the menu. The actions list corresponds to the dictionary values so the 2 lists can be linked. The labels are what will display on each button. Key_rects is a list that will store all the buttons rectangles.

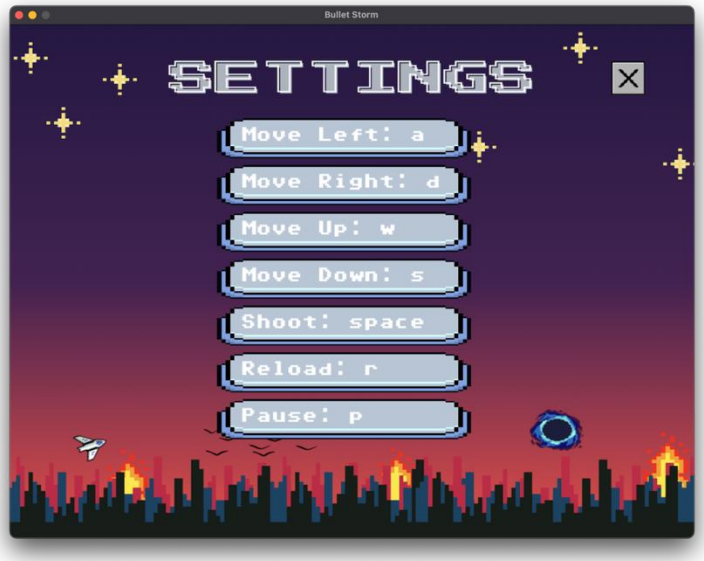
```
for i, action in enumerate(actions):
    if pressed:
        button_image = SMALL_BUTTON_PRESSED
    else:
        button_image = SMALL_BUTTON_IDLE
    label_message = f"{labels[i]}: {pygame.key.name(keybinds[action])}"
    label_text = FONT.render(label_message, True, (255, 255, 255))
    rect = pygame.Rect(300, 135 + i * 70, 300, 40)
    key_rects.append((rect, action))
    window.blit(button_image, (rect.x + 10, rect.y + 5))
    window.blit(label_text, (rect.x + 45, rect.y + 12))
```

The for loop above creates a button for each action in the list and iterates through the labels list for each button so they all have their correct name. It also goes through the key bind dictionary and writes the value of each action (the key bind). Using this for loop will automatically create a new button if I add a new action.

```
EXIT_BUTTON = pygame.Rect(900, 55, 100, 100)
window.blit(CROSS_IMAGE.convert_alpha(), (EXIT_BUTTON.x, EXIT_BUTTON.y))




pygame.display.update()
mouse_pos = pygame.mouse.get_pos()

for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if EXIT_BUTTON.collidepoint(mouse_pos):
                BUTTON_CLICK_SOUND.play()
                current_state == SETTINGS
```

I then drew the exit button which when clicked will take me back to settings. I will now add the key binnding meu to the main loop so that i can see it.

```
elif current_state == KEYBINDING:
    KeybindsMenu(window)
```



I adjusted the gap between the buttons and now the key binds menu is displayed correctly however the exit button is not working.

```
current_state = SETTINGS
```

To fix the problem I removed the double equals and made it a single.

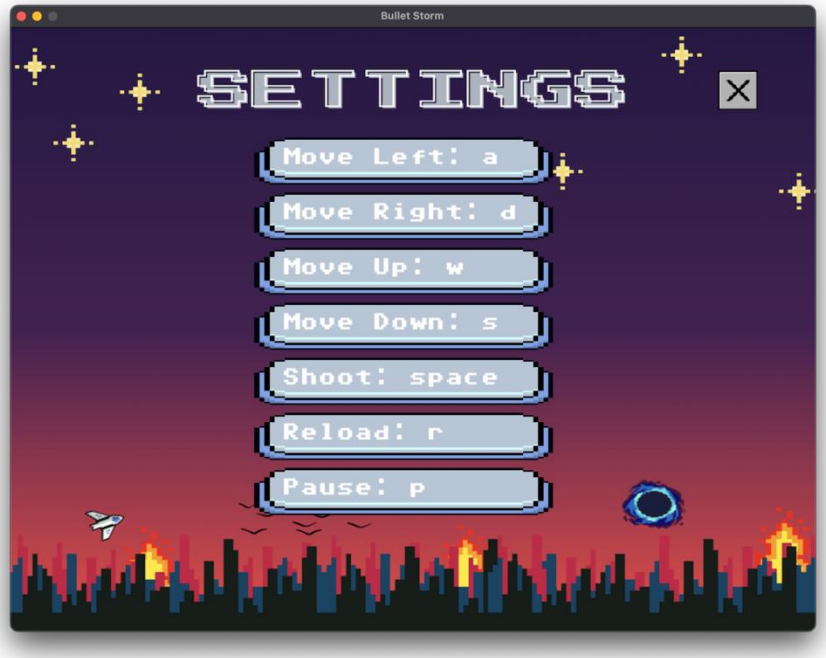
To make the buttons interactable, I had to create a separate function which returns the value of the last key pressed by the user.

```
def get_new_key():
    waiting = True
    while waiting:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            if event.type == pygame.KEYDOWN:
                return event.key
```

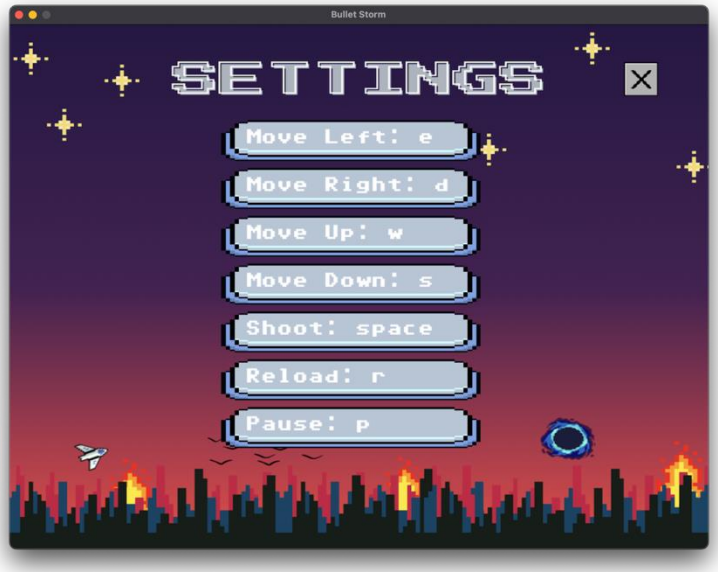
Next I will call this function when the buttons are pressed.

```
if event.type == pygame.MOUSEBUTTONDOWN:
    BUTTON_CLICK_SOUND.play()

    for rect, action in key_rects:
        if rect.collidepoint(mouse_pos):
            new_key = get_new_key()
            if new_key:
                if new_key in keybinds.values():
                    print("key alredy in use")
                else:
                    keybinds[action] = new_key
```

The new key variable holds the returned value of the key pressed and sets it as the actions new value in the dictionary if they key is not already being used. For debugging purposes, I am using a print function to see it will allow 2 actions to have the same value.



I am going to change move left from A to E.



It has worked correctly. Now I will try changing the pause value to E as well.

```
key alredy in use
key alredy in use

Process finished with exit code 0
```

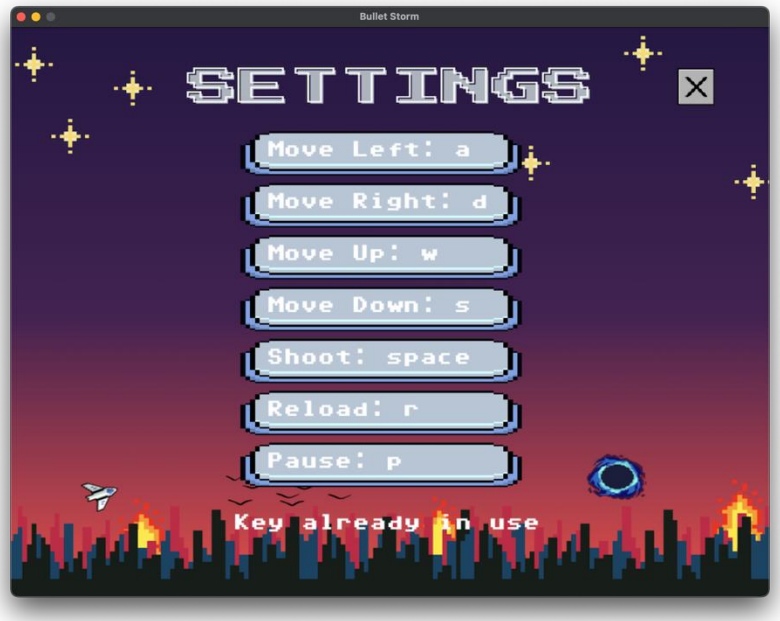
The value did not change, and this is being displayed in the terminal. To display this message temporarily on the screen, I created a new function.

```
def display_message(window, message):
    text_surface = FONT.render(message, True, WHITE)
    window.blit(text_surface, (300, 650))
    pygame.display.update()
    pygame.time.delay(2000)
```

This function will display the message for 2 seconds then come off the screen.

```
if new_key in keybinds.values():
    display_message(window, "Key already in use")
```

I called the function and this is the output.



At the bottom of the screen the message is displayed. To make it more noticeable, I added an error sound.

```
ERROR_SOUND.play()
```

```
if keys_pressed[keybinds['move_left']]:  # LEFT
    self.speed_x -= self.speed
    moved = True
```

```
if keys_pressed[keybinds['move_right']]:  # RIGHT
    self.speed_x += self.speed
    moved = True
if keys_pressed[keybinds['move_up']]:  # UP
    self.speed_y -= self.speed
    moved = True
if keys_pressed[keybinds['move_down']]:  # DOWN
    self.speed_y += self.speed
    moved = True
```

```
if keys_pressed[keybinds['reload']] and self.ammo < self.max_ammo and not self.reloading:
    self.reloading = True
    self.last_reload_time = pygame.time.get_ticks()
    RELOAD_SOUND.play()
```

```
if pygame.mouse.get_pressed() == (1, 0, 0) or keys_pressed[keybinds['shoot']]:
    self.shoot = True
    self.Shooting()
else:
    self.shoot = False
```

Finally, I changed the logic for the controls so that it uses the value stored in the dictionary and the key binds are working now.

For the stats menu, i have decided to add the leaderboard from the game over screen.

```
leaderboard_text = FONT.render('LEADERBOARD', True, WHITE)
window.blit(leaderboard_text, (370, 350))
player.display_leaderboard(window)
```

I drew the leaderboard onto the screen, and you can now view it.

Above the leaderboard I will display the players individual stats. To track player stats i first created 2 variables for the stats declared at the start of the game.

```
high_score = 0
total_enemy_kills = 0
```

Next, I created a function that updates these stats.

```
def update_stats():
    global high_score, total_enemy_kills

    if player.score > high_score:
        high_score = player.score

    if player.kills > total_enemy_kills:
        total_enemy_kills += player.kills
```

The function replaces the high score with the players last score if it was greater and adds on the players kills to total kills. The player.kills instance was created in the player class and set to 0.

This function will be called when the game over screen displays.

```
if self.dead:
    self.save_score()
    window.fill((0, 0, 0))
```

```
    window.blit(self.game_over_image, (0, 0))
    surface.blit(score_message, (WIDTH / 2 - (score_message.get_width() / 2), HEIGHT / 2 +
(score_message.get_height() / 2)-150))
    surface.blit(time_message, (WIDTH / 2 - (time_message.get_width() / 2), HEIGHT / 2 +
(time_message.get_height() / 2)-85))
    self.display_leaderboard(surface)
    update_stats()
```

Next I created a function that will save these scores into a JSON file.

```
def save_stats(player_username):
    global high_score, total_enemy_kills

    try:
        with open("stats.json", "r") as file:
            stats = json.load(file)
    except FileNotFoundError:
        stats = {}

    stats[player_username] = {
        "high_score": high_score,
        "total_enemy_kills": total_enemy_kills
    }

    with open("stats.json", "w") as file:
        json.dump(stats, file, indent=4)
```

I ran the code, and that data was saved correctly when the player died.

```
{
    "1": {
        "high_score": 80,
        "total_enemy_kills": 4
    }
}
```

“1” is the username I used for the account I logged in with.

In my update stats function I realised that I had a logic error. The players kills would only update if the kills were greater than the total kills. I updated the function and set the player kills to 0 in case the player plays again.

```
def update_stats():
    global high_score, total_enemy_kills

    if player.score > high_score:
        high_score = player.score
```

```
    total_enemy_kills += player.kills
    player.kills = 0
```

The code below is a function that will be called at the start of the game and will load the players stats after they login.

```
def load_stats(player_username):
    global high_score, total_enemy_kills

    try:
        with open("stats.json", "r") as file:
            stats = json.load(file)

        if player_username in stats:
            high_score = stats[player_username]["high_score"]
            total_enemy_kills = stats[player_username]["total_enemy_kills"]
    except FileNotFoundError:
        pass
```

```
elif current_state == MAIN_MENU:
    MainMenu(window)
    load_stats(username)
```

I called the function in the main game loop and now the high score and total enemy kils are updating correctly in the file.

```
{
    "1": {
        "high_score": 205,
        "total_enemy_kills": 7
    },
    "waez": {
        "high_score": 65,
        "total_enemy_kills": 3
    }
}
```

```
highscore_text = FONT.render('HIGHSCORE: ', True, WHITE)
totalkills_text = FONT.render('ALL TIME KILLS: ', True, WHITE)
```

```
highscore_value = FONT.render(str(high_score), True, WHITE)
totalkills_value = FONT.render(str(total_enemy_kills), True, WHITE)

window.blit(highscore_text, (300, 200))
window.blit(totalkills_text, (185, 250))

window.blit(highscore_value, (550, 200))
window.blit(totalkills_value, (550, 250))
```

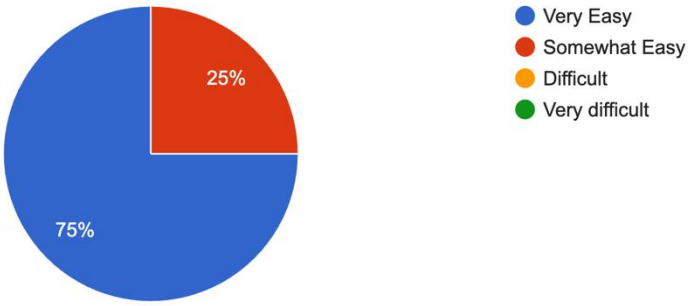
Finally, I drew it onto the screen.

<u>Beta Test Feedback</u>

I will now send my beta test survey out to the students who have now all had a chance to palay the game. The results will be displayed below.

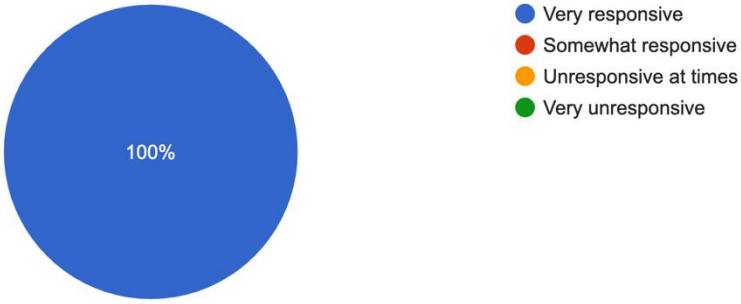
How easy was it to understand the controls?

8 responses



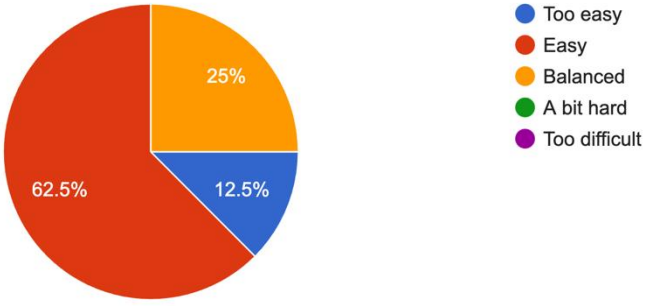
How responsive did the controls feel during gameplay?

8 responses



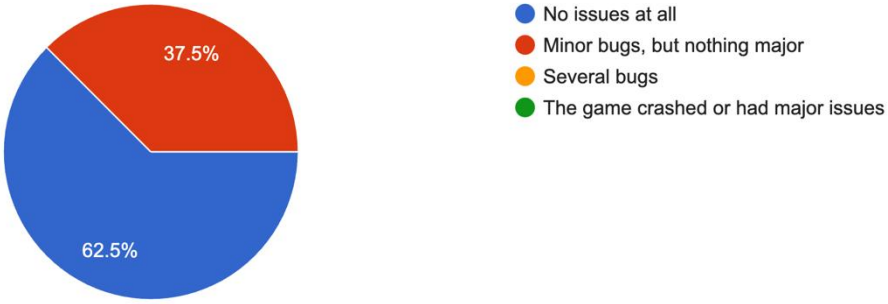
How would you rate the game's difficulty?

8 responses



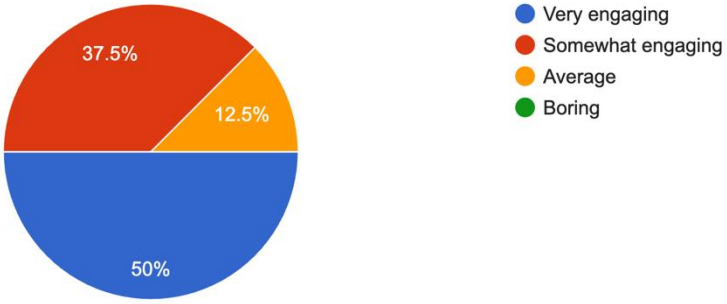
Did you encounter any bugs or technical issues while playing?

8 responses



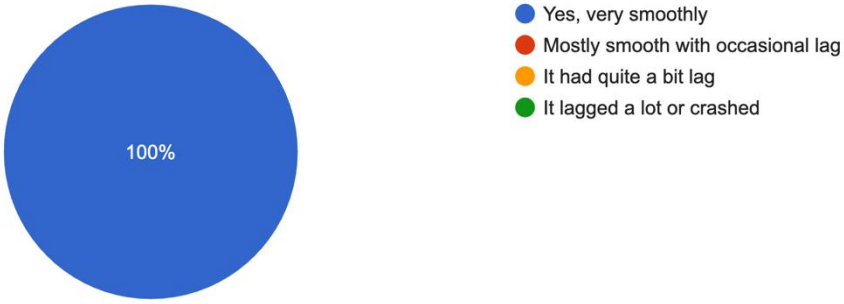
How engaging was the gameplay?

8 responses



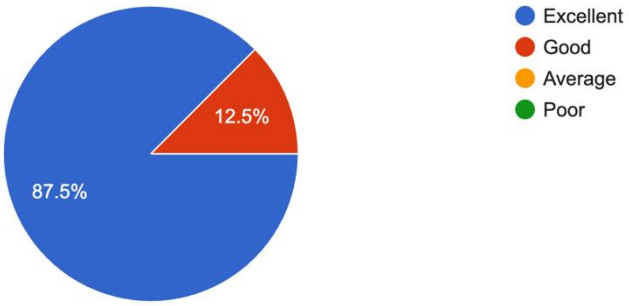
Did the game run smoothly on your device?

8 responses



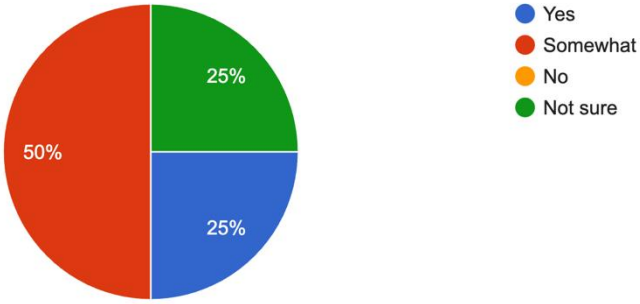
How would you rate the game's visual design?

8 responses



Did the game progressively get harder the longer you were alive for?

8 responses



What needs to improve?

8 responses

Allow enemies to see player from further away
Make enemies get more harder
Display players kills on screen somewhere
Enemies need to get harder and harder
Regenerate a portion of player health after killing an enemy
Spawn enemies and power ups in more places
Add more enemy spawn locations as their spawns get predictable after a while
Increase enemy spawn frequency

Name the bugs you encountered while playing, if found any.

8 responses

n/a
When clicking buttons a sound only plays for some of the buttons
The game timer doesnt start at 0
Didnt find any
When the game starts the player shoots a bullet straight away and some buttons dont have a click sound.
didnt encounter any
N/A
none

From this feedback I can take away a few positives and negatives.

For the positives:

- Everyone is happy with the game's visuals
- Controls are responsive for everyone

- Controls are easy to learn for everyone
- The game ran smoothly for everyone
- Not many bugs were present

The negatives:

- The enemies did not get hard enough
- A few bugs were found
- In game UI doesn't display all essential information
- Not enough enemy spawn locations
- Enemy spawn frequency very low
- Enemies can only see players when player is very close

The main complaint is that the game is not hard enough and the rate the enemy AI increases in difficulty is somewhat unnoticeable to some people. I also received a recommendation to regenerate the players health after killing an enemy and I believe this is a great idea as it will encourage players to camp les when their health is low.

The bugs found from the students were:

- Some buttons don't have a click sound
- The player shoots a bullet as soon as the game starts
- The game timer doesn't start at 0

I will first fix these bugs and then adjust the game settings so it lines up more with what the students would like.

Bug Fixes

To fix the button click sound bug, all I must do is play the click sound whenever the players mouse clicks a button. I already have this for many of the buttons, but I seemed to have missed a few.

```
BUTTON_CLICK_SOUND.play()
```

I added this line of code to each event code where a button is pressed. I tested every button in the game and the sound is now correctly playing.

To stop the player from shooting as soon as the game starts, I will add a small pause to the program as soon as start game is pressed to prevent the click on the button to be registered as a click for shooting.

```
f event.type == pygame.MOUSEBUTTONDOWN:
    if START_BUTTON.collidepoint(mouse_pos):
        BUTTON_CLICK_SOUND.play()
        time.sleep(0.3)
        current_state = GAME
```

When the start button is pressed, time.sleep(0.3) will cause the program to pause for 0.3 seconds and then start the game. This prevents the mouse from still being clicked down when the game starts avoiding an unnecessary bullet to be shot.

Finally, to fix the game timer issue, I had to change the logic of when the start time variable started to track time. Initially it would start the timer as soon as I ran the code however now I have changed it so that it will start the timer as soon as the game state is in 'GAME'. I did the same thing for the difficulty timer.

```
start_time = 0
difficulty_start_time = 0
```

Initially they are set to 0.

```
if event.type == pygame.MOUSEBUTTONDOWN:
    if START_BUTTON.collidepoint(mouse_pos):
        BUTTON_CLICK_SOUND.play()
        time.sleep(0.3)
        start_time = pygame.time.get_ticks()
        difficulty_start_time = pygame.time.get_ticks()
        current_state = GAME
```

As soon as start button pressed, start tracking time.

Game Adjustments

Many students claimed that the game was easy and didn't get hard enough. To fix this I am going to adjust some settings. First, I will start off with the difficulty increasing function.

```
def increase_difficulty():
    global DIFFICULTY_MULTIPLIER, difficulty_start_time
    current_time = pygame.time.get_ticks()

    if current_time - difficulty_start_time >= 60000 and difficulty_start_time <= 36000:
        DIFFICULTY_MULTIPLIER += 0.25
        difficulty_start_time += 60000
```

Currently this function only updates the layer difficulty up to 6 minutes played so to fix this, I will remove the upper bound so that the difficulty increases indefinitely.

```
def increase_difficulty():
    global DIFFICULTY_MULTIPLIER, difficulty_start_time
    current_time = pygame.time.get_ticks()

    if current_time - difficulty_start_time >= 60000:
```

```
        DIFFICULTY_MULTIPLIER += 0.25
        difficulty_start_time += 60000
```

Now the difficulty multiplier will increase by 0.25 every single minute.

```
def restart_game():
    global player, enemies, score, start_time, paused, difficulty_start_time, DIFFICULTY_MULTIPLIER

    for enemy in EnemyGroup:
        enemy.kill()
    for powerup in PowerupsGroup:
        powerup.kill()

    player.score = 0
    player.ammo = MAX_AMMO
    player.pos = pygame.math.Vector2(START_X, START_Y)
    player.kills = 0
    DIFFICULTY_MULTIPLIER = 1.0
```

In the restart game function, I had previously forgotten to reset the difficulty multiplier so I have done that now. Next, I will increase the spawn frequency for the enemy.

```
ENEMY_SPAWN_TIME = random.randint(int(5000/DIFFICULTY_MULTIPLIER), int(10000/DIFFICULTY_MULTIPLIER))
```

I changed the time frame for an enemy to spawn from between 5 and 20 seconds to now 5 and 10 seconds with the value decreasing each round.

```
def damage_taken(self, damage):
    self.health -= damage
    HITMARKER.play()
    if self.health <= 0:
        self.kill()
        ENEMY_DEATH_SOUND.play()
        player.increase_score(self.points)
        player.kills += 1
        player.health += 10
```

For each kill the player gets, their health will increase by 10.

```
if player.is_boosted:
    self.damage = STRENGTH_BOOST_VALUE * DIFFICULTY_MULTIPLIER
```

I have also decided to increase the strength boost value as the enemies get harder as well so that it remains effective. Next I will change the enemies view distance from 400 to 550 so that they can see further.

Before:

```
if distance < 400:
    self.idle = False
else:
    self.idle = True
```

After:

```
if distance < 550:
    self.idle = False
else:
    self.idle = True
```

Upon running the code to test the new distance, I received an unexpected error from the health bar.

```
    filled_bar = self.foreground_image.subsurface((0, 0, filled_width, self.h))
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: subsurface rectangle outside surface area
```

This error is happening because the players health increased over the max health when I killed an enemy as it adds 10 on each kill and I was already on max health.

```
if player.health <= 90:
    player.health += 10
```

I have now fixed the logic, so it only increases the health when it is 90 or less.

I then tested the enemies new view distance and notice a good increase.

```
ENEMY_SPAWN_LOCATIONS = [(1773, 337), (1771, 1293), (620, 1094), (2438, 629), (2643, 915)]
```

Currently there are only 5 possible enemy spawn locations. I am going to add a few more to the list.

```
ENEMY_SPAWN_LOCATIONS = [(1773, 337), (1771, 1293), (620, 1094), (2438, 629), (2643, 915), (2162, 1104),
(1066, 456), (1210, 7968), (1400, 1400), (1966,621), (520, 1000)]
```

I will do the same for the power ups.

Before:

```
POWERUPS_SPAWN_LOCATIONS = [(1579, 1292), (2152, 1005), (1111, 1295), (1012, 426), (1593, 322), (626,
724)]
```

After:

```
POWERUPS_SPAWN_LOCATIONS = [(1579, 1292), (2152, 1005), (1111, 1295), (1012, 426), (1593, 322), (626,
724), (443, 904), (1107, 912), (1695, 630), (2355, 612)]
```

Next, I will display player kills on screen.

```
def draw_kill_count(self, surface):
    kill_text = FONT.render(f'Kills:{self.kills}', True, WHITE)
    surface.blit(kill_text, (WIDTH - 240 , 65))
```

Second Beta Survey

This second survey will ask the students the same questions, but some questions will be removed as they are for things that haven't been changed from the last survey. The questions will ask:

How responsive did the controls feel during gameplay?

- A. Very responsive
- B. Somewhat responsive
- C. Unresponsive at times
- D. Very unresponsive

How would you rate the game's difficulty?

- A. Too easy
- B. Easy
- C. Balanced
- D. A bit hard
- E. Too difficult

Did you encounter any bugs or technical issues while playing?

- A. No issues at all
- B. Minor bugs, but nothing major

- C. Several bugs
- D. The game crashed or had major issues

How engaging was the gameplay?

- A. Very engaging
- B. Somewhat engaging
- C. Average
- D. Boring

Did the game run smoothly on your device?

- A. Yes, very smoothly
- B. Mostly smooth with occasional lag
- C. It was a bit laggy
- D. It lagged a lot or crashed

Did the game progressively get harder the longer you were alive for?

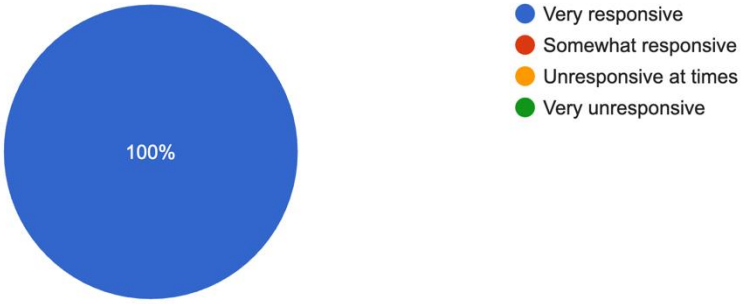
- A. Yes
- B. Somewhat
- C. No
- D. Not sure

Name the bugs you encountered while playing, if found any.

The answers are attached below.

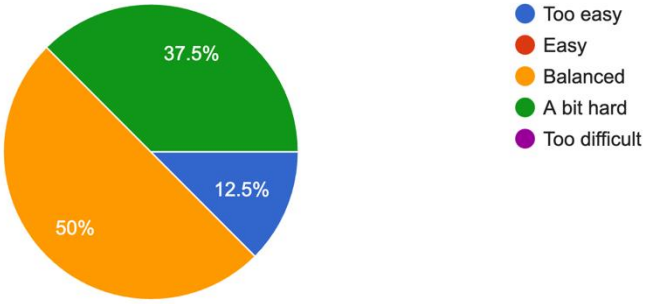
How responsive did the controls feel during gameplay?

8 responses



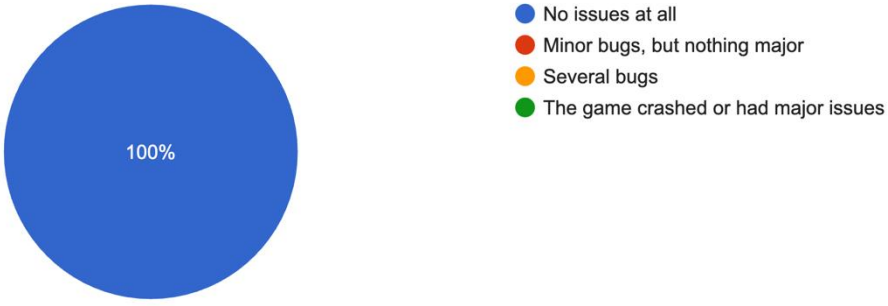
How would you rate the game's difficulty?

8 responses



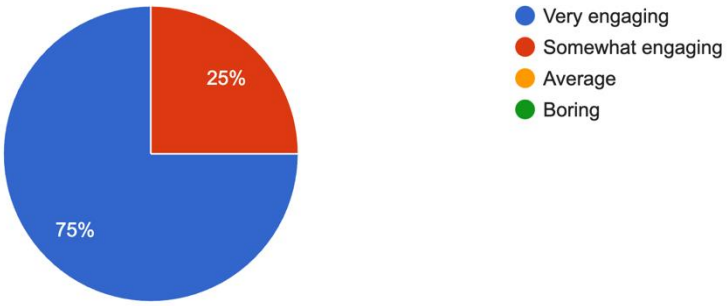
Did you encounter any bugs or technical issues while playing?

8 responses



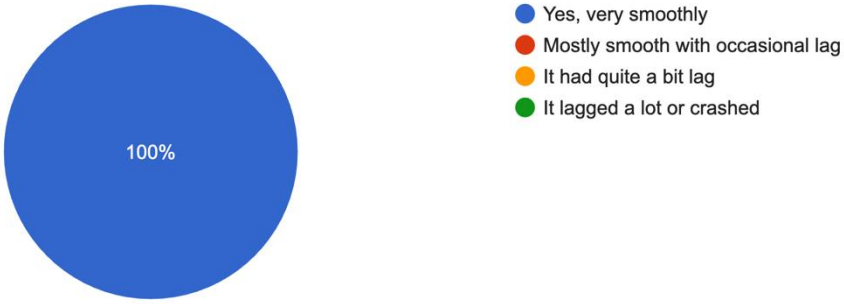
How engaging was the gameplay?

8 responses



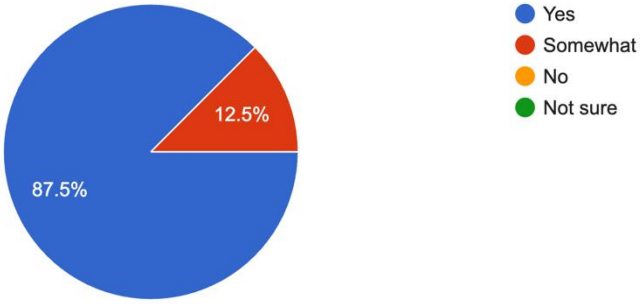
Did the game run smoothly on your device?

8 responses



Did the game progressively get harder the longer you were alive for?

8 responses



No bugs were reported.

Beta Feedback Results Analysis

From these results I can infer that the enemy AI is now increasing its difficulty in a more noticeable rate as almost everyone now selected yes whereas only 25 percent selected it before. The gameplay has also gotten more difficult than before according to the students however a significant increase in students are now saying that the game is more engaging which means the increased difficulty is a good thing. Performance and responsiveness have still stayed good for the students, and no one faced any issues.

Evaluation

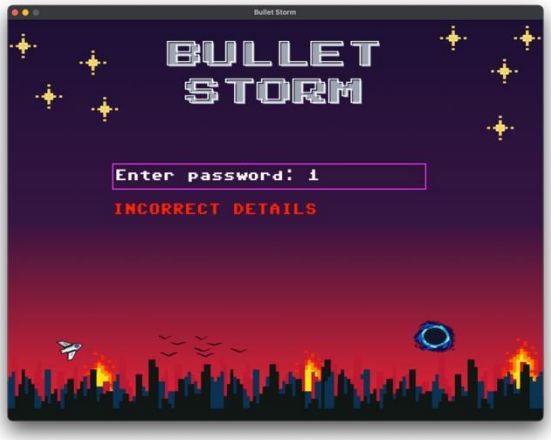
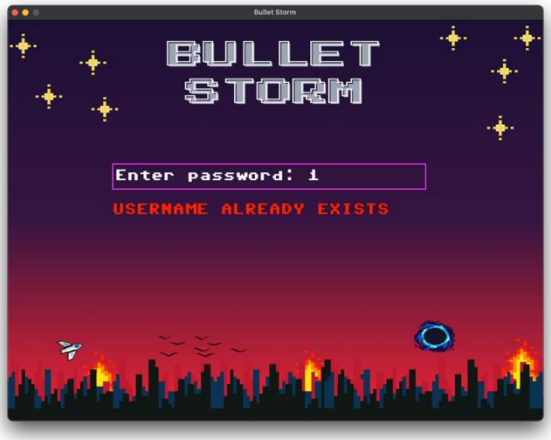
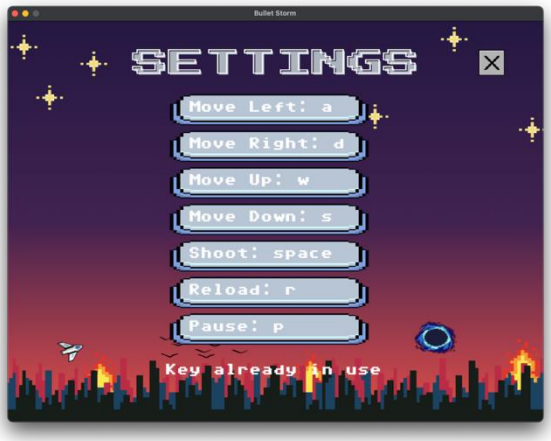
This evaluation will assess how effectively the game meets the design goals and user requirements.

Robustness Testing

I will first evaluate the robustness and usability of the final solution through testing which o carried out at the end of the development process. The aim is to ensure that the solution meets the required functionality, handles errors effectively, and provides a friendly user experience.

Test	Input	Expected Output	Actual Output	Result (Pass/Fail)
Invalid Log in Details	Enter wrong username or password	“Incorrect details” error message displays	Error message displayed	Pass
Creating Account with Existing Username	Create an account with an existing username	“Username already exists” error message displays	Error message displayed	Pass
Bind a key to a key already in use	Bind ‘W’ key to reload function	“Key already in use error message displays” error message displays	Error message displayed	Pass
Rapid Input	Press Pause time rapidly in succession	Game keeps pausing and resuming without crashing or decreasing performance	Game paused and resumed correctly	Pass
Long Button Hold	Hold mouse button down over a button	The button is only clicked once and doesn't register multiple inputs	Button clicked once	Pass

Evidence

Test	Screenshot
Invalid Log In Details	
Creating Account with Existing Username	
Bind a key to a key already in use	

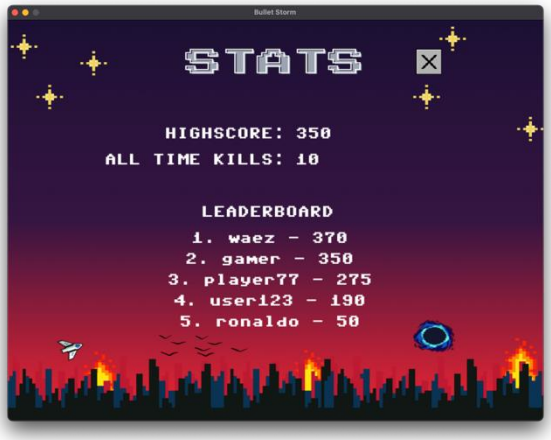
Functionality Testing

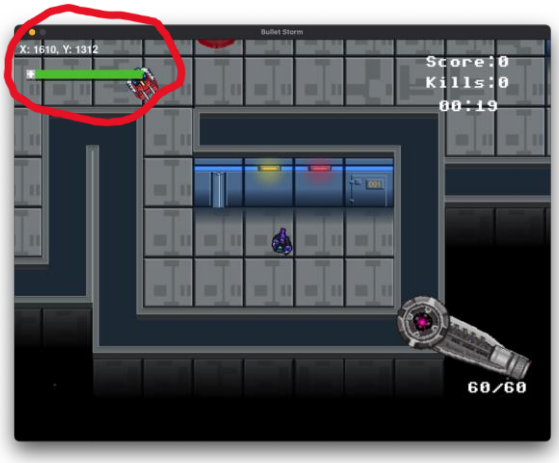
Test Name	Functionality to Test	Steps for Tester	Expected Outcome	Pass or Fail
Main Menu Load Test	Ensure the main menu loads correctly	Launch the game, sign in and wait for the main menu to appear	The main menu should load with all buttons visible	Pass
Start Game Button Test	Verify the "Start" button starts the game	Click on "Start Game" from the main menu	The game should transition to the gameplay screen	Pass
Exit Game Button Test	Check that the "Exit" button exits the game	Click the "Exit" button from the main menu	The game should close without errors	Pass
Pause Menu Load Test	Ensure the pause menu works during gameplay	Start the game, press the pause button	The pause menu should appear, and gameplay should pause	Pass
Resume Game Test	Make sure game can resume	Pause the game, then press pause button again to resume	The game should resume from the exact point where it was paused	Pass
Timer Start Test	Test if the timer starts at the correct moment	Start a new game and check the timer	The timer should start counting from 00:00 when gameplay begins	Pass
High Score Update Test	Check high score updates correctly after a game session	Create a new account, play a game, achieve a new high score, and check the high score in the stats menu	The high score should be updated correctly	Pass
Total Kills Update Test	Check total enemy kills are updated properly	Play a game, kill enemies, and check the stats menu for "Total Kills."	The total kills should update accurately	Pass
Restart Game Test	Test restarting the game from the pause menu.	Pause the game, select "Restart," and observe gameplay.	The game should restart with all elements reset.	Pass
Enemy Spawn Test	Check if enemies spawn correctly during gameplay	Start the game and observe enemy spawns	Enemies should spawn at the correct intervals and locations	Pass
Difficulty Increase Test	Ensure the difficulty	Play the game for several minutes	The game should become	Pass

	increases over time	and observe enemy behavior	progressively harder as time passes	
Power-Up Functionality Test	Test if power-ups spawn and function correctly	Collect power-ups during gameplay and observe their effects	Power-ups should spawn correctly, apply the intended effects, and disappear afterwards.	Pass
Collision Detection Test	Verify collisions between sprites and walls	Move the player into a wall and observe the response	The player should not be able to walk through the wall	Pass
Health Reduction Test	Check if the player's health reduces correctly after taking damage	Take damage from enemies and check the health bar.	The health bar should decrease	Pass
Game Over Test	Ensure the game ends correctly when the player loses all health	Let the player die and observe the transition to the game over screen	The game over screen should appear	Pass
Stats Save Test	Verify that game stats save to a file	Play the game, and check the stats file after game finishes	Stats should remain saved and accurate	Pass
Player Control Test	Check if all controls work	Move the player, shoot the gun and reload the gun	All controls should work	Pass

Evidence

Test	Screenshot
Main Menu Load Test	

Pause Menu Load Test	
High Score Update Test and Total Kills Update Test	Before:  After: 
Health Reduction Test	Before:

	 After: 
Game Over Test	

Usability Testing

Usability testing was performed to assess the user-friendliness of the solution. I had one of the 8 students evaluate the interface, controls, accessibility and gameplay. Below is a table that I made for them, where they fill out if a test either passed or failed and there is a space for extra information.

Test	Purpose	What to check	Pass or Fail	Extra Information
Navigating Menu	Test if players can navigate menus easily	Can you intuitively figure out what each button does and where to find every menu	Pass	All buttons were clearly labelled
Game understanding	Ensure players can easily understand how to play the game	Do you understand the controls and the objective of the game without external help	Pass	Key bind menu displays all the controls so if unsure people can use that
Difficulty	Check if the games difficulty progresses gradually	Does the game gradually get harder or do you suddenly notice a difference at a certain point in the game	Pass	The game got harder the longer and longer I played in a way that felt natural
Score Tracking Test	Test if high scores and stats save correctly.	Do your stats update in the stats menu correctly after each game	Pass	My high score was saved correctly in the stats menu
In-Game UI	Check if all essential game information is displaying during the game	Can you see your score, kills, health, time survived and how, much ammo you have. Is there anything missing?	Pass	All relevant information is displayed in game
Control Response	Test if player controls are responsive.	Does the expected outcome happen instantly as you press a key	Pass	All buttons are extremely responsive
Accessibility	Ensure the game is accessible for all types of players.	Are there colorblind-friendly features? Can you	Pass	All game controls can be binded to something else,

		easily read the text? Can you bind a key to something else to fit your needs?		there are 3 different types of colour blind modes, and all text is easy to read.
Sound and Music	Test the quality and volume of menu and game audio. Test if it can be adjusted to fit the player's needs.	Is the background music too loud or too quiet and does it match the game's theme? Can you turn off certain audio like sound effects or music.	Pass	The is an option to mute music and mute sound effects, and the background music audio is at a good low distracting level.
Performance	Ensure the game runs smoothly on different hardware.	Does the game lag or drop frames? Are loading times fast or slow?	Pass	I ran the game on a school laptop and my own personal laptop. The performance felt pretty much exactly the same
Leaderboard	Test if the leaderboard accurately displays player scores.	Are scores shown from highest to lowest	Pass	Leaderboard is correctly displaying high scores
Pause button	Ensure the game pauses properly.	Does pausing stop all in-game activity and resuming leave you off at exactly where you stopped	Pass	Pause button works as expected
Replay Value	Assess whether players want to replay the game.	Is the game engaging enough to replay or will it get boring after a while	Fail	The game is very engaging and fun however there is no progression system meaning getting good scores is only good for the leaderboard
Graphics	Test if the graphics are clear and easy to see.	Are important game elements visually distinct?	Pass	Graphics are excellent and game is clear and easy to see

Success Criteria Analysis

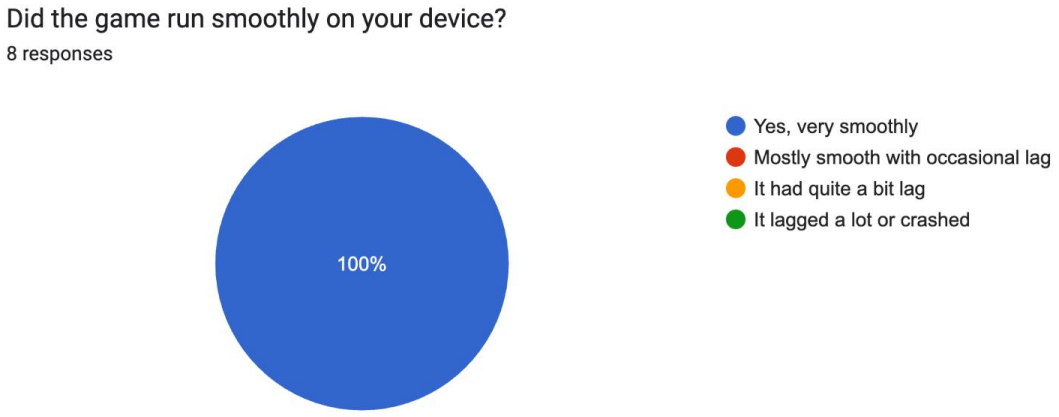
Below are the success criteria I made in my analysis.

- High Performance
- Engaging and Enjoyable Gameplay
- Offline Playability
- Appealing Art Style and Theme
- Player Customization and Power-ups
- Working Enemy AI
- High Engagement
- Intuitive User Interface
- Accessibility

I will now comment on what I achieved and did not achieve from the success criteria above.

Performance

From my testing and the feedback of the beta testers, I believe my game has ticked off high performance criteria. From the beta tester survey, 100 percent of the students claimed they had no performance issues when running the game. One of my beta testers played the game on a low-end computer and a high-end computer and claimed there was no noticeable difference between them.



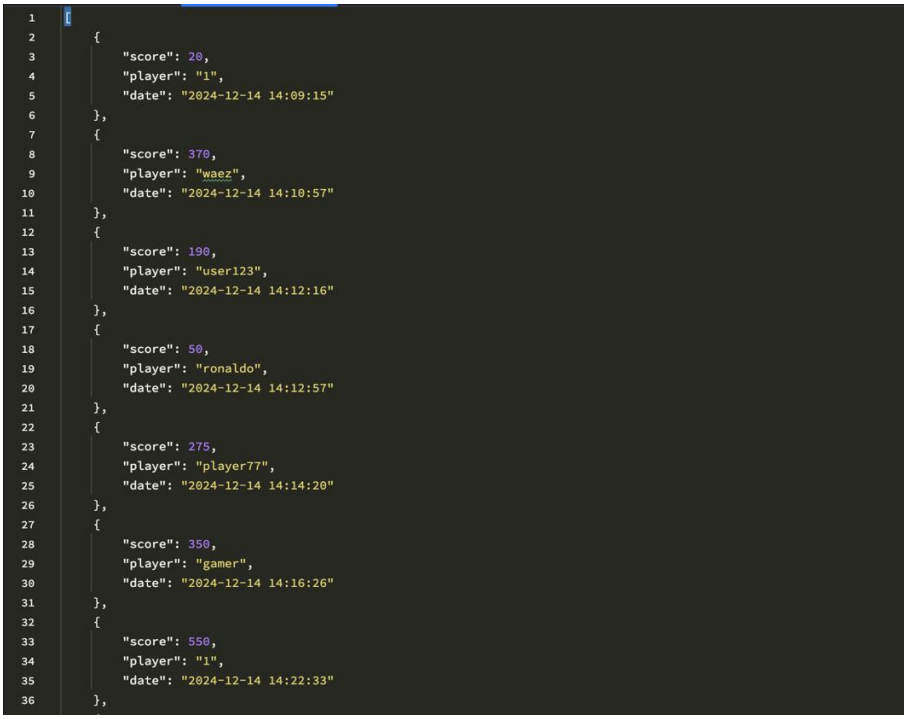
Above is the evidence of the students claiming they had a very good performance.

Engaging and enjoyable gameplay

I believe this criterion is fully met. The game includes fast-paced gameplay with an enemy AI that gradually gets harder and harder the longer the player is alive, which really helps the player stay engaged and not get bored after a certain amount of time. The wide variety of powerups in the game also encourage the player to change their playstyle and meaning that the game does not get repetitive after a while.

Offline Playability

The game uses a JSON file stored locally with the game to store leader board information instead of using a file stored in the cloud. This will allow the leader board to be viewed offline and not require an internet connection allowing the game to be played offline.

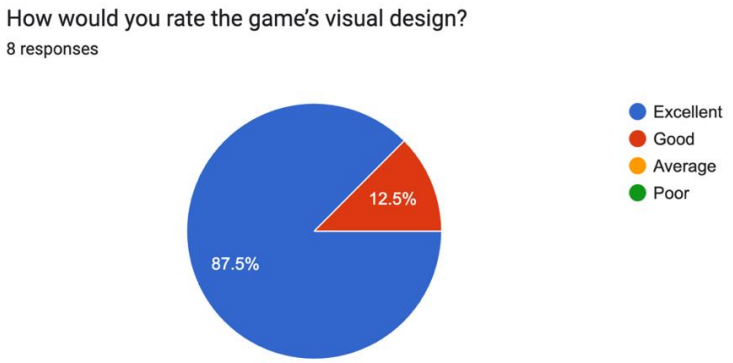


```
[
    {
        "score": 20,
        "player": "1",
        "date": "2024-12-14 14:09:15"
    },
    {
        "score": 370,
        "player": "waez",
        "date": "2024-12-14 14:10:57"
    },
    {
        "score": 190,
        "player": "user123",
        "date": "2024-12-14 14:12:16"
    },
    {
        "score": 50,
        "player": "ronaldo",
        "date": "2024-12-14 14:12:57"
    },
    {
        "score": 275,
        "player": "player77",
        "date": "2024-12-14 14:14:20"
    },
    {
        "score": 350,
        "player": "gamer",
        "date": "2024-12-14 14:16:26"
    },
    {
        "score": 550,
        "player": "1",
        "date": "2024-12-14 14:22:33"
    },
```


The file above is stored locally within the computer and as each player plays, their store will get written onto the same file.

Appealing Art Style and Theme

The feedback from the beta testers helps prove that this has been met.



87.5% of the testers said the games visual design is excellent and I believe it follows a consistent theme throughout.

Player Customisation and Power-ups

The criteria above is not fully met. In the game, there is zero player customisation as you can only play with one character and no in game assets can be changed. The reason this was not met was due to limited time during the development phase and prioritisation of other features in the game. To achieve this next time, I can add an interface where players can select their character before the game starts. However, the power ups were fully implemented, and they work perfectly in game as mentioned above in the functionality test.

Working Enemy AI

This criteria is met as the enemy gradually gets harder as the game goes on but can be built on more to further improve the AI. Currently, the enemy has 2 states, idle and attacking player. Depending on the distance from the player, the enemy will choose one of these states. In idle mode it randomly patrols the map adding a natural feel to the AI. To further enhance the AI's decision making on when to change states, I can run in more parameters in the enemy class methods which can contribute on what state the AI will be in. For example, if the player shoots the AI right now, but the AI is not close enough to the player, it will not start to attack the player. If I add a parameter which the AI can use to tell if it has been shot or not, I can make it attack the player no matter the distance.

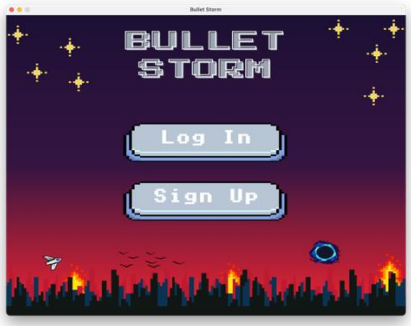
Progression System

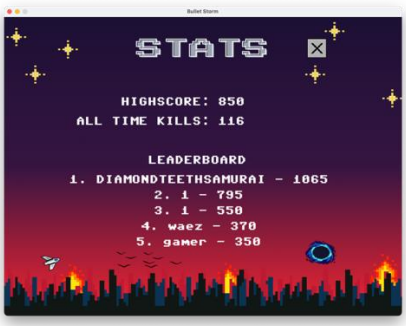
This criterion was not met at all. The game has no progression system, and each player starts off each game with the exact same stats. I decided to leave out this feature purposely because if there was a way to permanently boost player stats, the game would start to get easier, and the leader board wouldn't show off the most skilled players but instead the ones who have played the longest. To include this feature in the future, I can instead have a ranking system which shows a players rank and that can be used to show how much a player has progressed instead of increasing stats.

Intuitive User Interface

This criterion was fully met. From the feedback of the beta testers, it is proven that the UI is simple to navigate and easy to read.

Test	Purpose	What to check	Pass or Fail	Extra Information
Navigating Menu	Test if players can navigate menus easily	Can you intuitively figure out what each button does and where to find every menu	Pass	All buttons were clearly labelled

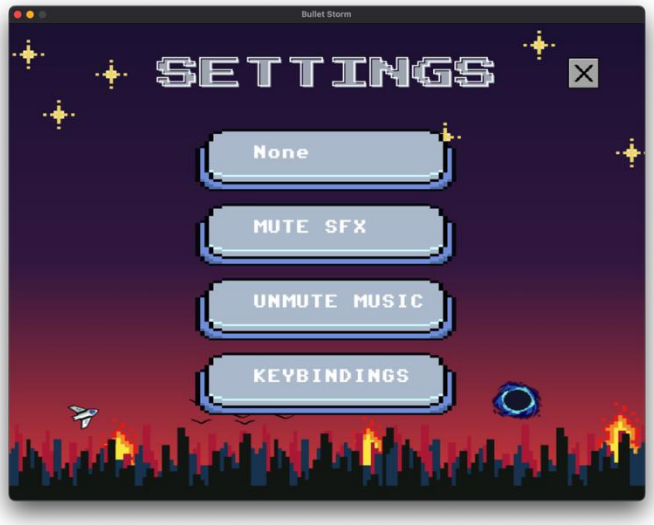




All buttons are clearly labelled and in the main menu they have corresponding icons above for easier understanding.

Accessibility

This criterion has been fully met due to the variety of options in the settings menu The image below shows the settings menu in-game.



There are 3 different types of colour-blind modes players can choose from, players can mute either game sound effects or game music and finally every single control can be remapped into something else.

Maintainability

The code is structured into separate classes for each in game component, such as player, enemies, bullets, and power ups which encapsulate their attributes and behaviours ensuring that changes to one thing, such as the player's movement, do not affect the enemy's movement. The classes also allow a range of characters to be created. For example, if you inherit from the enemy class, you can create a various number of different types of enemies.

The code is easy to read and understand due to appropriate naming conventions. For example, the use of naming functions shoot_player() or damage_taken() makes sure that the purpose of each code is clear. Storing most constants in a separate file called 'settings' allows for easy adjustments without having to search through long lines of code.

It also includes error-handling such as 'try…except' blocks for managing file related errors. However, this error handling is not included anywhere else. However, it does not have a lot of comments which if added can further improve understanding for other programmers.

Currently, when a player creates an account, their data is stored in a newly created file. This is fine for a small, scaled game with not a lot of players however of a lot of people start playing, a numerous number of files will be made causing it to be untidy and a higher risk for data loss. To prevent this next time, all player details should be stored in one file together creating a large database.

Overall, the code gives a high level of maintainability due to its modular design, clear code structure, and file error-handling mechanisms. There are some areas for improvement such as adding more comments, more error -handling and other than that the solution is prepared for future development and debugging.

Limitations and Potential Further Development

Currently, the game only has one endless game mode which can get boring for users who prefer a wider range of game modes. I believe if there was a larger time frame for development, these modes could have been added. For example, adding a round based mode or a story mode will attract a larger audience to the game.

The game also lacks a multiplayer option which can put off players who like to play with friends. I couldn't add this feature right now due to a lack of understanding on bow multiplayer servers work and their costs. Adding a local multiplayer mode would allow friends to play together and further develop their teamwork skills.

There are only 3 enemy types in the game right now. An even wider range of enemies can allow for a more fun gameplay experience. For example, adding a boss enemy after certain time intervals to improve difficulty and add more challenge. A boss enemy can really help increase the games skill gap allowing for more room for improvement in players.

As mentioned earlier, a progression system and customisation can really benefit the game as it will personalise the gaming experience for the player more. This can be included by adding a season pass where players must get a certain amount of score which will help them unlock new cosmetics that they can use in game. The reason I struggled to add a customisation aspect to the game was due to a lack of free top-down pixel sprites available online.