A Level Project

Name:

Centre:

Candidate Number:

Contents

[Analysis 4](#_Toc76372880)

[Introduction 4](#_Toc76372881)

[Computational Suitability 5](#_Toc76372882)

[Current and/or Similar Solutions 5](#_Toc76372883)

[Match Hole 5](#_Toc76372884)

[5](#_Toc76372885)

[Missile Command Clone 6](#_Toc76372886)

[Touch Ball 7](#_Toc76372887)

[Conclusion from my research 8](#_Toc76372888)

[Limitations that I must bear in mind: 8](#_Toc76372889)

[Stakeholders 8](#_Toc76372890)

[Questionnaire 1 8](#_Toc76372891)

[Analysis of questionnaire 1: 10](#_Toc76372892)

[Questionnaire 2 10](#_Toc76372893)

[Analysis of questionnaire 2: 12](#_Toc76372894)

[Final List of Specifications and Requirements. 12](#_Toc76372895)

[Software and Hardware Requirements 15](#_Toc76372896)

[Design 17](#_Toc76372897)

[Overview: 17](#_Toc76372898)

[Design Objectives 17](#_Toc76372899)

[Interface Design Concepts 19](#_Toc76372900)

[Main Menu 21](#_Toc76372901)

[Game Over Screen 22](#_Toc76372902)

[Controls Screen 22](#_Toc76372903)

[Graphic Design: 23](#_Toc76372904)

[Concept of How The Game Would Flow 25](#_Toc76372905)

[Simplified Flowchart Of The Gameplay 25](#_Toc76372906)

[Masking 26](#_Toc76372907)

[Collisions In Pygame 26](#_Toc76372908)

[Rotations In Pygame 26](#_Toc76372909)

[What Is Masking and Why It Would Be Ideal for Me 27](#_Toc76372910)

[Variables and Data Structures 28](#_Toc76372911)

[Class: Enemy 28](#_Toc76372912)

[PsuedoCode Algorithm and flowchart for ChangeState: 28](#_Toc76372913)

[PsuedoCode Algorithm and flowchart for Update(): 29](#_Toc76372914)

[Class: Cursor 29](#_Toc76372915)

[PsuedoCode Algorithm and flowchart for Update(): 29](#_Toc76372916)

[Class: Friendly 30](#_Toc76372917)

[PsuedoCode Algorithm and flowchart for ChangeState: 30](#_Toc76372918)

[PsuedoCode Algorithm and flowchart for Update(): 31](#_Toc76372919)

[Class: Line 31](#_Toc76372920)

[PsuedoCode Algorithm and flowchart for Update(): 32](#_Toc76372921)

[Spawn Procedure 33](#_Toc76372922)

[Destruction and Locking Procedure. 35](#_Toc76372923)

[Collision Procedure 36](#_Toc76372924)

[Test Plan 38](#_Toc76372925)

[Implementation 42](#_Toc76372926)

[Iteration 1 42](#_Toc76372927)

[Requirements being developed 42](#_Toc76372928)

[Errors 42](#_Toc76372929)

[Conclusion 42](#_Toc76372930)

[Iteration 2 42](#_Toc76372931)

[Requirements being developed 42](#_Toc76372932)

[Errors 42](#_Toc76372933)

[Conclusion 42](#_Toc76372934)

[Iteration 3 42](#_Toc76372935)

[Requirements being developed 42](#_Toc76372936)

[Errors 42](#_Toc76372937)

[Conclusion 42](#_Toc76372938)

[Testing 43](#_Toc76372939)

[Evaluation 44](#_Toc76372940)

[Requirements 44](#_Toc76372941)

[Stakeholders 44](#_Toc76372942)

[Usability 44](#_Toc76372943)

[Maintainability 44](#_Toc76372944)

[Future developments 44](#_Toc76372945)

[Bibliography 44](#_Toc76372946)

[Code listing 44](#_Toc76372947)

# Analysis

## Introduction

People of all ages utilise sorts of digital information in order to destress. Many Year 12 and Year 13 students, after working intense hours during and after school in order to gain high qualifications from the exams they are preparing for utilise video-games as a way to relax and destress themselves. Many middle-aged professionals also utilise video-games in order to destress after long working hours. According to research, high levels of stress can lead to physical symptoms including [headaches](https://www.webmd.com/migraines-headaches/default.htm), [upset stomach](https://www.webmd.com/digestive-disorders/digestive-diseases-nausea-vomiting), elevated [blood pressure](https://www.webmd.com/hypertension-high-blood-pressure/guide/diastolic-and-systolic-blood-pressure-know-your-numbers), [chest pain](https://www.webmd.com/pain-management/guide/whats-causing-my-chest-pain), and problems sleeping. Research suggests that stress also can bring on or worsen certain symptoms or diseases. The problem is that the video-games that are utilised to destress do not necessarily allow the player to destress and do not benefit the player.

A good solution to this problem would be a game that would allow the player to destress and benefit them. For example, by improving the players reflexes. Having good reflexes aids performance in sports, exercise, and everyday physical activities like crossing the street, driving and working while poor reflexes can be an underlying reason for faulty movement and injury. Reflexes and reaction time do have a surprisingly large bearing on our ability to avoid any undue injury. Training and challenging your brain can help strengthen the brain and its neural networks. If you frequently train reaction time, the brain's connections will become stronger and healthier, which means that when it comes time to use response time, it will be quicker and require fewer mental resources. Studies have shown that playing video games regularly may increase gray matter in the brain and boost brain connectivity. (Gray matter is associated with muscle control, memories, perception, and spatial navigation.). In a study involving a group of surgeons, researchers found that those who played video games were faster at performing advanced procedures and made 37 percent fewer mistakes than those who didn’t. Special video games have also been used as physical therapy to help stroke victims regain control of their hands and wrists. Strategic and role-playing video games also challenge players to solve complex problems. This virtual problem-solving serves as great practice for real-life hurdles. The American Psychological Association showed in a 2013 study that adolescents who played strategy video games had improved problem-solving abilities and better grades in school. The study showed a correlation between how frequently the participants were playing games and how much improvement they saw — the more strategy games they played, the more their problem-solving skills and school grades improved.

My solution is to create a game that would allow the player to destress and also improve their reflexes. Furthermore, according to my research, creating a game-based solution would cause a number of other different benefits than just improving their reflexes and mental health.

## Computational Suitability

This is primarily due to the fact that the inputs and outputs of the game would be ideal for a game with a computer due to the fact that if, for example, a reflex game was made for a phone or tablet it would be hard to pay the game properly with more than one input as in a mobile device you are limited to inputs from the touch screen. Furthermore, creating a game is much easier on a computer as there is a vast array of resources that can be used to create games in computers such as programs like pygame and unity. Furthermore, on a mobile device in order to allow other users to play the game you would have to publish it onto mobile app stores. This process is long and tedious, and is not present in a computer as you can transfer the game via a file.

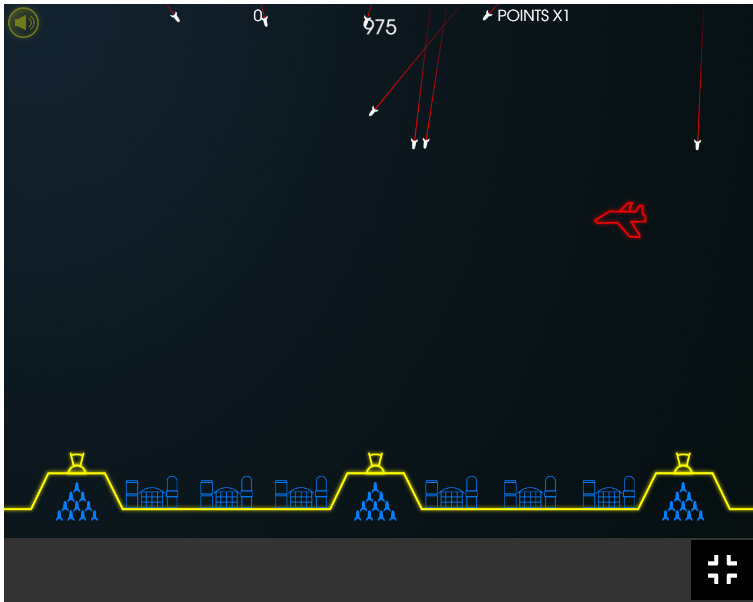
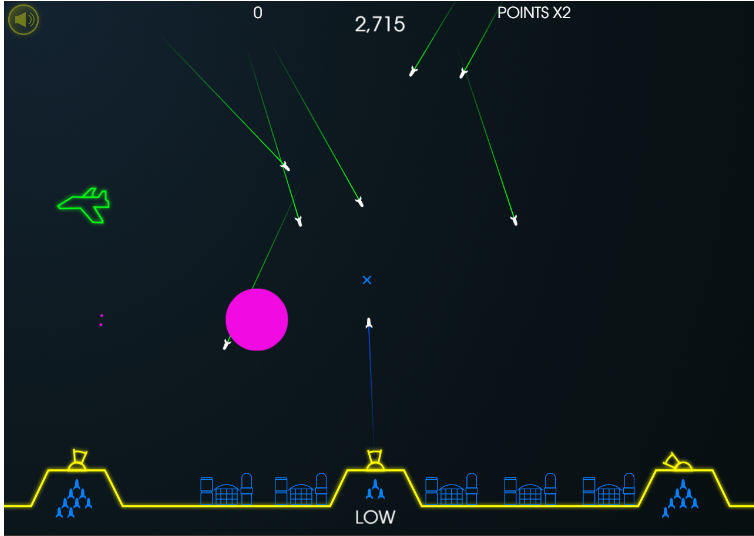
## Current and/or Similar Solutions

### Match Hole

## 

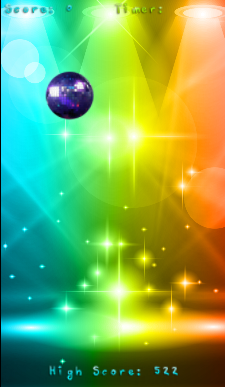
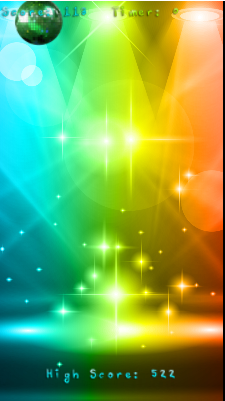
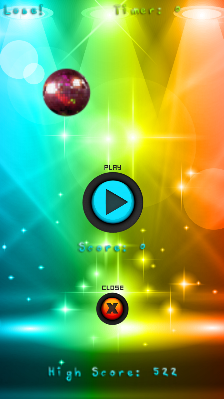
Match Hole is a game in which you have to match a clockwise or anti-clockwise rotating ball onto a randomly determined black ball. When the two balls are on top of each other you click onto the centre of the large circle. If you time this action right you gain a point. If you do not you lose the game, the game then switches to a menu in which it displays your record score and your current score. It then gives you the option to restart or not. The game has a very simple premise’ and does not require a lot of perquisite game knowledge in order to master. Already at the start the white ball moves at a great speed, making it a trickier game than it appears. It is a game that, after playing for a long duration, would improve the reflexes of the person playing the said game. The game also utilises sounds to indicate when you’ve failed or succeeded and has an ambient soundtrack playing in the background. This gives the game a relaxing sense. I believe that the fact that there is light contrast between the colours in the game improves the overall aesthetic of the game. The fact that a high score system is present improves the replayability of the game as the game compares the score you achieved with your high score. This causes the player to aspire to beat his/her high score

## Missile Command Clone



The game above is a clone of the retro classic missile command. The aim of the game is to protect your bases from the incoming missiles. The game has a limited yet contrasting colour pallet, making it easy to distinguish between friend and foe. The user input is registered via the mouse position and mouse press. There is a crosshair that indicates the position of your mouse on the game. When you click the right mouse button, you fire a missile, these missiles are fired from one of 3 missile sites present in the game. The missile site that files the missile is determined via the position of your mouse and the ammunition in the missile sites. When your ammunition is low or empty it is indicated via white text. Your ammunition count is indicated via blue missile symbols under the missile sites. When you fire the missile the missile travels to the position you have pressed on and then explodes at the point. The explosion has a radius, and if this radius comes in contact with a missile, it destroys the missile. When you destroy a missile, you gain points, furthermore depending on the amount od bases that you have remaining by the end of the level you earn extra points. However, if you accidently shoot down a friendly jet, you lose points. The speed of your and the enemy missiles are quite slow, this game does not test your reflexes rather it tests your ability to conserve your missiles.

## Touch Ball



Touch Ball is a game in which a randomly you have to click on a ball before time runs out. The ball’s spawn position is randomly generated. I f you cannot press the ball in a certain amount of time the game over screen appears. In order to indicate that you have successfully pressed the ball the ball turns green. You utilise your mouse cursor as your means of control. Your score increases every time you press the ball however your score depends on how rapidly you press the ball. The game utilises a background soundtrack that I find to be distracting. It utilises a vibrant colour scheme that is engaging. The game allows you to compare your score with your highest score. This game is not complex and is satisfying to play.

## Conclusion from my research

From analysing the games, I extracted features that I plan to add into the questionnaires that I will provide my clients with. These games showed me that you do not need high quality graphics in order to make a game enjoyable or fun. The simplicity and quality of the mechanics that the game utilises makes it engaging and enjoyable. The games do not contain any sort of story driven element. However, their difficulty does increase as time passes. All the games have a score system. All the games allow the player to improve the reflexes and destress. I was most impressed by the missile command clone, and think that incorporating a reflex oriented approach to the game would be and ideal solution

## Limitations that I must bear in mind:

I need to take into account of certain clients that might not be able to properly utilise the software. For example, those with certain visual impairments such as colour-blindness might not be able to distinguish certain shape or colours. That means I must optimise my software in order to take into account these limitations. Such optimisations might include features such as high contrast colours and shapes. This would benefit the game in the whole as it would make it more engaging and easier to understand. I will also have to optimise the inputs my game utilises. As it will most likely use peripherals such as a mouse and keyboard, I must minimise the number of inputs required in order to make it an easier game and make it more accessible to those who might have certain motor skill disabilities.

## Stakeholders

My users consist of students between the ages of 15-18 and professionals between the ages of 40-45. They want a game that is straightforward a game that does not require a significant amount of time and investment in order to fully experience it, they objected to the suggestion that the game should have a story mode of sorts. Nonetheless they also want a game that is also challenging to certain extent as they feel that if it was not challenging, they would not be engaged by the game They want the control scheme of the game to be simple as they agree that utilising a complex control scheme would be tedious. They said that they would want to utilise their personal computers in order to play the game. In order to further determine my clients demand I will conduct a survey.

## Questionnaire 1

I plan to interview 8 of my clients by sending them a questionnaire that I have created.

My questions are:

1. Have you ever plaid reflex-oriented games before?
2. How many unique games have you played in this genre before?
3. What features of the games did you dislike?
4. What features of the games did you like?
5. What features would you have wanted to see in those games?
6. What platform would you prefer to play these games on?
7. What do you think is the most important element of a game?

Q1

Yes (8)

Q2

4 (1)

3 (2)

2 (3)

1 (2)

Q3

Design, i.e it felt too childish,bland (3)

Annoying sound design(4)

Pacing issues (1)

Q4

Simplicity (5)

Difficulty (2)

The fact that the game was relaxing (1)

Q5

Nothing more (4)

More intuitive controls (2)

Randomly generated levels (2)

Q6

PC (8)

Q7

Replayability/engagingness (5)

Fun-factor (2)

Interactiveness (1)

## Analysis of questionnaire 1:

I can observe that the entirety of the respondents to my questionnaire are knowledgeable about the type of software that I am aiming to create. I arrived to this conclusion from the information that I gathered from question 1 and question 2. This will help me a great deal throughout development as my interviewees will be able to offer me suggestions as they have considerable knowledge on the subject. I utilised open ended questions like q3, q4 and q5 in order to gain a wide range of information. This will be beneficial to me as it will provide a significant amount of information that will help me build my specification requirements.

This is the list of base specification requirements that I have created from this questionnaire:

-The game mechanic s will not be complex

-The gameplay will be optimised so that it is engaging

-The user shall control the inputs

-The game shall contain a points system, in which a total score will be created.

-The game will be an endless loop, i.e there shall not be any sort of levels, just a continuous stream.

-The games mechanics will revolve around reflex speed.

-The game’s aesthetic will be engaging and not feel too childish.

-The game will end/ a game over menu will appear when the user fails to complete the reflex-oriented task after a certain amount of time

## Questionnaire 2

My questions are:

1. How you would want to control the character?
2. Would you want music and if so what type?
3. What would you want the overall aesthetic to be like?
4. Would you want abilities or power-ups to be included into the game?
5. Would you want an inventory system to be implemented into the game?
6. Would you want the enemies to have a health bar?
7. Would you want difficulty levels to be implemented into the game?
8. Would you want the difficulty to increase over time?
9. What would you want the window size to be?

Q1

Mouse (5)

Keyboard(4)

Q2

Yes(6)

--ambient

--disco

-electronic

No (2)

Q3

Retro (3)

Simplistic (2)

Futuristic (1)

Arcade (1)

Cant comment (1)

Q4

No(6)

Yes(2)

Q5

No(8)

Q6

No(4)

Don’t know how it would fit with the game (2)

Yes (2)

Q7

Yes(3)

No(5)

Q8

Yes(4)

No(4)

Q9

1000x1000(2)

Smaller than the size of my screen (4)

1920x1080p(2)

## Analysis of questionnaire 2:

This questionnaire gave me a wealth of information regarding the wants of my clients regarding design and gameplay elements. The majority of my clients accepted the suggestion that the game should contain any music, however they offered a variety of music types, which will force me to decide between the different types. Regarding the aesthetic, retro was the most popular one. The majority of my interviewees did not want abilities, an inventory system and health bar systems as the believed that they would unnecessarily overcomplicate the game and make it tedious. They disagreed with the notion of difficulty settings or rise as they stated that it would be unnecessary. They were not able to give proper responses to the screen size, however the majority of them stated that it should be smaller than the resolution of their own screens.

In order to improve the aesthetic of the game and add a unique mechanic, the game will be based around a radar, this radar will detect the enemies, the player will then have to lock and then destroy the target in quick succession

## s

## Final List of Specifications and Requirements.

Must have:

1. A window size shall of 1000x1000

Justification: The answers given in questionnaire 2 question 9

1. A game loop that prioritises the player completing three separate tasks in quick succession

Justification: In order to allow the game have and engaging premise

1. A game loop that revolves around the players reflex speed

Justification: Improving one’s reflex speed is the goal of my project

1. The user moves around a cursor that will be used to destroy the enemy

Justification: This is in order to create an object that registers the user’s inputs

1. The user inputs will be managed by the mouse and keyboard

Justification: The answers given in questionnaire 2 question 1

1. A score system that will record the score achieved by the user

Justification: In order to add a reward element to the gameplay

1. The player move the cursor around via mouse inputs

Justification: The answers given in questionnaire 2 question 1

1. The user utilize keyboard inputs to lock and destroy enemies

Justification: The answers given in questionnaire 2 question 1

1. A base on the screen

Justification: In order to have a visual reference of the location that will cause game over if the enemies to reach it

1. Enemies that move towards the base

Justification: In order to allow the game loop to work as the game revolves around preventing said enemies from reaching the base

1. A system that opens the game over menu when the enemy collides with the base

Justification: In order to allow the game to go to game over after an enemy collides with the base

1. A line spinning around its own axis in the center of the screen

Justification: In order to create a radar-like aesthetic

1. This line should be able to collide with the enemies

Justification: In order to actually create a functioning radar the line must be able to collide with objects

1. A system that allows the enemies to become visible when detected by the radar.

Justification: This is in order to allow the user to see the detected targets, as after collision with the radar, they will be detected

1. A game over screen

Justification: In order to have a screen that indicates the game is over

1. A start screen

Justification: In order to have a screen that displays the title of the game, containing the play menu and the controls menu

1. A controls menu

Justification: In order to allow new players to understand the controls of the game.

Should have

1. A system that stores the scores that the user achieved, that creates a high score

Justification:

1. An aesthetic that is engaging and does not feel too childish.

Justification: The answers given in questionnaire 2 question 3

1. A system that awards the user points when the user destroys enemies.

Justification: In order to allow the score system to function, the player must receive points from a certain system

1. A system that displays the users score and high score in the game over menu

Justification: To add an element of re-playability to the game, as the user can try to constantly beat their previous high scores

1. The enemies spawn in random order at pre-determined locations

Justification: In order for the game to not feel repetitive

1. A system that changes the visuals of the enemy when they are locked

Justification: In order to indicate to the user when they’re able to shoot down the enemies

Could have

1. The base be colored a different color

Justification: In order to allow the user to more easily distinguish the base

1. Retro-electronic music playing in the background

Justification: The answers given in questionnaire 2 question 2

1. A system that creates an audible sound when an enemy has been destroyed

Justification: In order to enhance the immersion of the game

1. A system that creates an audible sound when the game is over

Justification: In order to enhance the immersion of the game

1. A system that creates makes other enemies disappear when an enemy is locked onto

Justification: In order to improve the visual aesthetics of the game

1. A friendly target that you could detect

Justification: In order to introduce another complex mechanic into the game, that will further test one’s reflexes

1. A system that causes the game over screen when you destroy the friendly target

Justification: In order to introduce another complex mechanic into the game, that will further test one’s reflexes

Won’t have

1. Power-ups or abilities

Justification: The answers given in questionnaire 2 question 4

1. Enemies with health bars

Justification The answers given in questionnaire 2 question 6

1. An inventory system

Justification: The answers given in questionnaire 2 question 5

## Software and Hardware Requirements

1. – MacOS, Windows, LinixThe OS required to run game IDE’S

Justification: The answers given in questionnaire 1 question 7

1. Pygame to be downloaded on the player’s computer

Justification: Pygame will be the program that I will utilise to create my game

1. 4GB RAM

Justification: The minimum amount of ram required for game IDE’S

1. 1.5 GHZ+ processor

Justification: The minimum processing power required for game IDE’S

1. 3GB HDD or SSD space

Justification: The minimum amount of storage required for game IDE’S

# Design

## Overview:

I will create a set of design objectives form the specifications that I have created in section 1. With my clients we have settled for a 2D reflex game. This game will be in the form of a radar, in which you have to use your fast reaction speeds in order to destroy the targets before they reach your base. When destroyed the enemies shall increase your score. When the enemies reach your base, the game is over as you have failed.

I will create the game using python, specifically the pygame plugin. In order to achieve the desired looks for the enemies, and the player-controlled cursor I will utilise pixel art, as allows me to create the sprites that I want to create in the specific sizes I want them to be in. I will utilise OOP in order to give different properties to different objects in the game. In order to create a radar like effect I will have to create a spinning sprite that has collisions and does not cause any optimisation issues. As pygame does not have proper rotational functions for sprites this will be quite challenging

List of specifications from Section 1:

-The window size shall be 1000x1000

-User inputs will be managed by the mouse and keyboard

-The user will move around an object that will be used to destroy the enemy

-The game will have a score system, in which it will record the score achieved by the user

-After the enemies cross a certain point in the game a game over screen will come out

-The goal of the player will be to complete two separate tasks in quick succession in order to destroy the enemy

-The games mechanics will revolve around reflex speed.

-The game’s aesthetic will be engaging and not feel too childish.

-When the enemy is destroyed the player will be awarded points as the score would increase

-The game over screen will display the score achieved by the user.

-The enemies will spawn in random order at pre-determined locations.

-The radar will detect the moving enemies

-The player shall control a cursor that will move onto the targets

-Then the player shall use the keyboard to lock and destroy the target

-The game should have a simplistic look.

## Design Objectives

**Visuals:**

Gamescreen:

1. Black background
2. 2 different radius circles that have the same origin point
3. Radar-line
4. Grey coloured background lines that will resemble geographic borders
5. Radar Cursor
6. Enemies
7. Score-Display– green text

Game Over Menu:

1. Score Display – green text
2. High Score Display -green text
3. Retry Button – green text
4. Game over green - text

Main Menu:

1. Play Button
2. Game Title – green text

**User Input:**

Gamescreen:

1. Mouse and key inputs that will guide the cursor and allow the player to destroy the enemies

Game Over Menu:

1. Retry Button that will allow user to retry

Main Menu:

1. Play Button that will allow the user to start playing the game

**Output:**

Gamescreen:

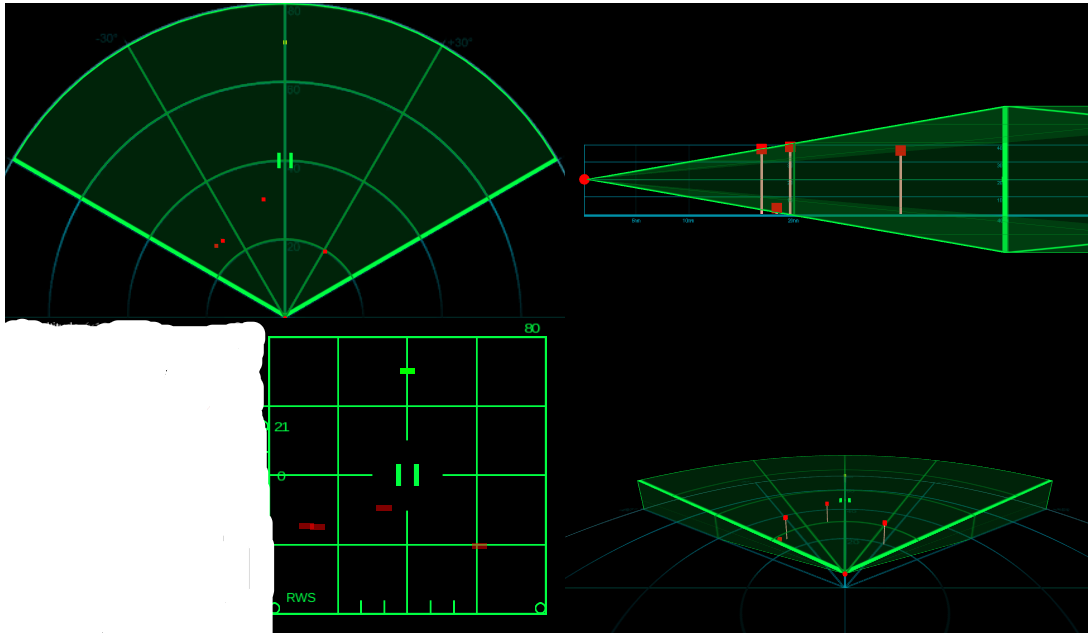
1. Score-Display– green text

Game Over Menu:

1. Score Display – green text
2. High Score Display -green text
3. Retry Button – green text
4. Game over text

Main Menu:

1. Play Button
2. Game Title – green text

Primary Design Influence 

s

s

s

## RADAR screen taken at RAF Neatishead, Norfolk, RAF Air Defence Radar Museum Stock Photo - AlamyInterface Design Concepts

I want my game’s aesthetic and layout to be similar to the image above. I like the minimalist backgrounds that outline the map of the United Kingdom and Northern France which add a sense of immersion without overwhelming the player. I like how the aircraft or objects illuminated by the radar are represented by small dots on the screen. I also like how the screen represents the radar in a circular/retro manner. This would be harder to create than a normal rectangular radar screen however I believe that it would fit the general aesthetic of the game better. I would want my player to feel as if he is controlling such a radar whilst, he/she is playing my game. The screen in the image does not overload the user creating a sense of control and focus that would be essential in a reflex oriented game. However, I believe that the orange colour scheme utilised here would be inferior to a green colour scheme as a contrast is less present with the orange color scheme. I also believe that in the image the size of the blips are too small and they are too numerous, it would hard to differentiate between different objects detected by the radar due to their small size

In the picture above is the radar symbology of an F15 aircraft

I want my visual aesthetic to be similar to the image present as I believe that it will make my game stand out from similar games. The contrast between the green and black of the display is vibrant and will not tire the eye. Furthermore, the iconography on the lower left-hand side of the image is a great point of reference as it could be easily recreated in a 2d environment.

Gamescreen: (above).

Icon of a friendly blip

Like the target it also moves towards the centre

Icon of a target that has been locked

Targets move towards the center of the radar towards the base.

Radar Cursor

It can move in any direction.

Box in which the players current score will be displayed

Radar Line that detects the targets.

The line moves clockwise, spinning around its own axis

Base

The point which will cause the game to end if the enemies cross it

Icon of a target that has been detected by the radar but has not been locked

In order to correlate with the radar aesthetic, the gamescreen shall primarily be composed of black and green. The gamescreen will have a simplistic design. This is in order to meet the specification, which was established from my questionnaires, that the game should have a simplistic design. This is the reason why I have a limited number of text and/or menu elements present on the screen. The score shall be displayed on the bottom-right hand corner, allowing the player to observe their score without being distracted from the game. The size of the window shall be 1000x1000 pixels

The radar line shall constantly spin around its own axis, in a clockwise manner. It will also be coloured green. It will have a small width on order for it to not obscure the enemies while passing over it. This radar will pass over otherwise invisible enemies, and then collide with them. After this collision the targets will be detected and turn into a certain icon.

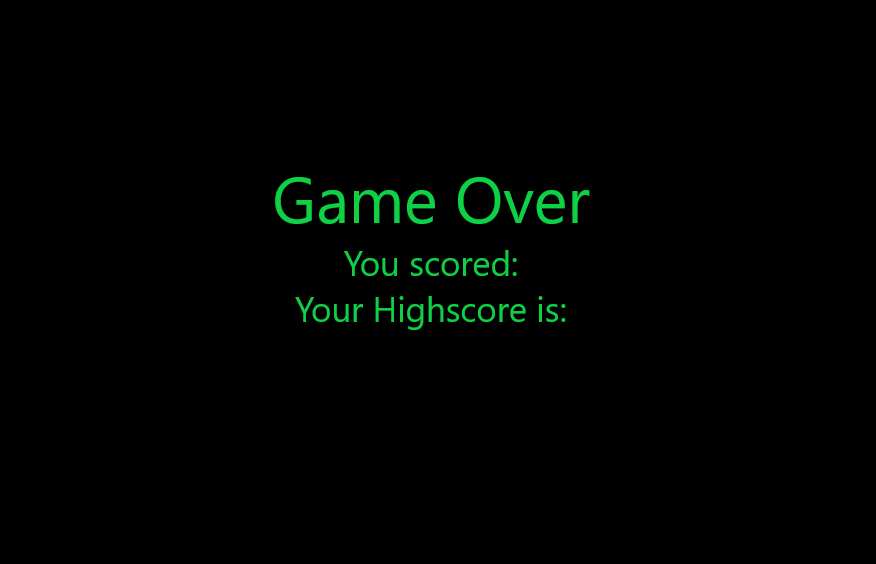
The radar cursor will be wide enough to surround the icon of an unlocked target. After locking the target, the player will be able to destroy the target. The icon of a locked target is far different from the icon of an unlocked target.

The base will be highlighted with a blue outline. This will be done in order to allow the player to distinguish the base from the rest of black-green background.

Main Menu:

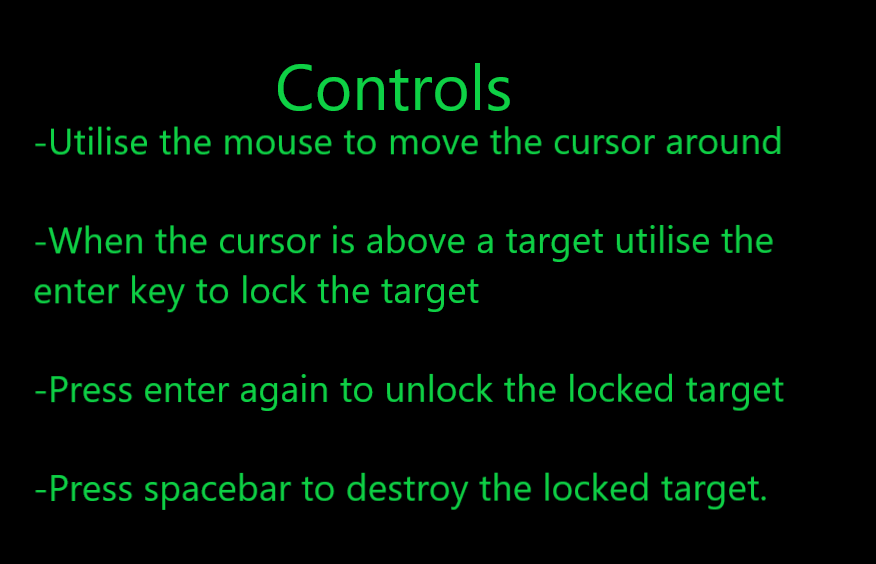


Menu: (above).

This is a quite simple menu, which displays the title of the game and a button prompt. The prompt and title are both coloured green whilst the background is coloured black. After pressing the button, the screen will change to the game screen. There will be two other buttons. One named control and the other named Quit. The controls menu will display the controls of the game and give a brief overview on how the game is played. The quit button will exit the game when it is pressed. The size of the windows will be 1000x1000.

Game Over Screen: (above).

This is a quite simple game over screen. The window size is 1000x1000. It contains 3 lines of text. 1 indicated that the game is over, the other indicates your currents score. The one below displays your highest ever recorded score. This will add an element of competitiveness to the game as the player will be able to compare his/her current score with the previous record, creating a desire to improve.



Controls Screen: (above).

In this screen a simple overview of the controls of the game and how the game is played will be outlined. To be in line with the aesthetic of the game the text will be coloured green.

## Graphic Design:

The icons/characters that I want to create will be influenced from my primary design. In order to create the various states that I want my enemy sprite to be in I will have to manually create certain images in order for them to fit in my creative vision. In order to do this, I will utilise a pixel editor. Pixel editors allow you to create images from individual pixels. are ideal for me as I want to create a 2D game. Furthermore, they are very easy to use and allow you to export your images without cost making it ideal for utilising it to create concept images. I will create 4 unique images. The pixel editor that I will utilise is called Piskel



Image 1: Unlock

Here are the two iterations of the UNLOCKED state of the enemy sprite. One is 40x40 width, the other is 20x20 width. I ended up selecting the 20x20 iterations as I concluded that the size 40x40 would be too large.

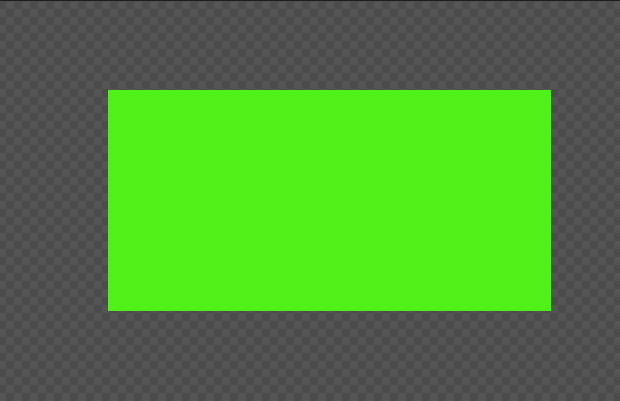
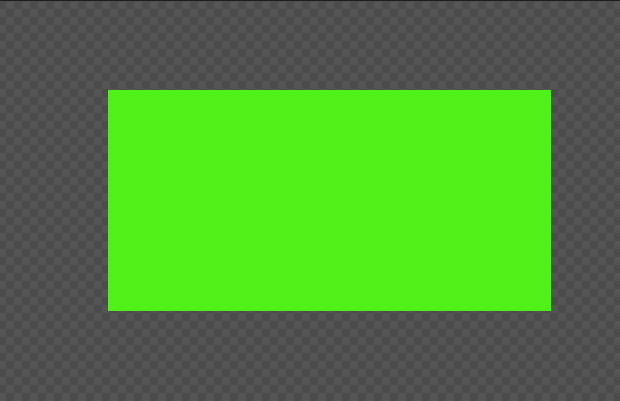
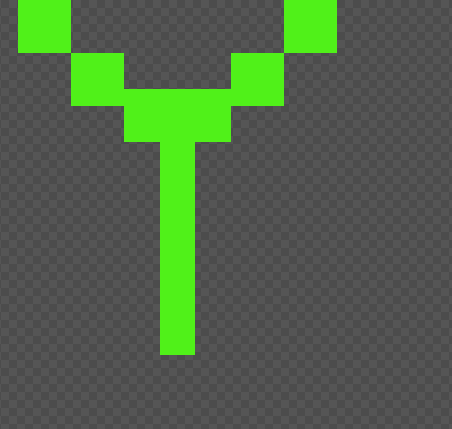
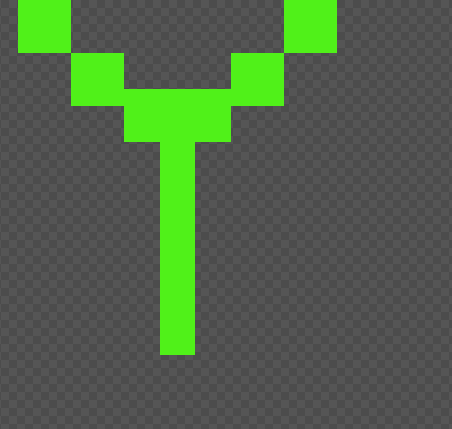
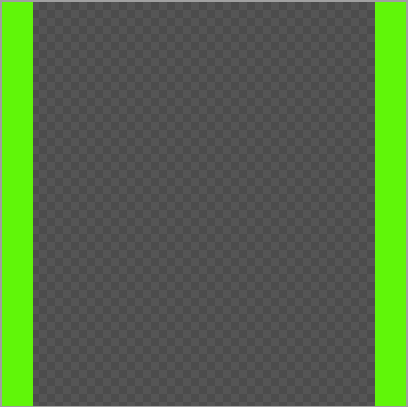


Image 2: Lock



Similarly with the first image I concluded that 40x40 would be too large of a size so I decided to go with a 20x20 configuration. This will be the appearance of the enemy when it is locked by the player.

Image 3: Cursor



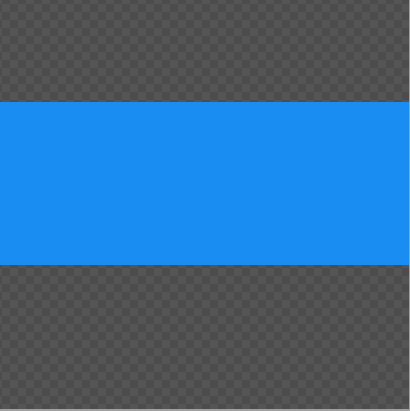
Here is the cursor. This will be the object that the player controls. The player shall bring this ontop of the Unlocked enemy and by using keyboard inputs lock the target. After locking the target, the player will be able to destroy the enemy. The size of the cursor will be 40x40 in order to allow it to encompass the Unlocked enemy.

Image 4: Invisible:



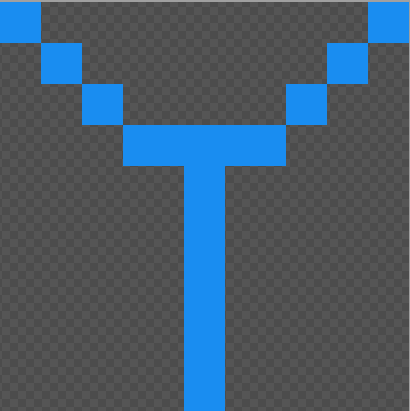
I created a 20x20 size blank sprite. This is in order to create a sprite to represent the enemy whilist it is invisible to the radar. This will bypass the complexity of utilising multiple layers or other means to conceal an object visually via code, as switching between images is a much easier solution.

Image 4: Friendly Unlock:



Here Is the Unlocked states of the friendly object. It is near identical to the unlocked state of the enemy sprite but differs in color. It is colored blue instead of green as blue is commonly used in games to indicate friendly object/units etc. Its different color will hypothetically allow the user to differentiate between the enemy. The fact that is the same shape might confuse the player, which I am intending to do. It also has a size of 20x20 pixels

Image 4: Friendly Lock:



Font Selection:

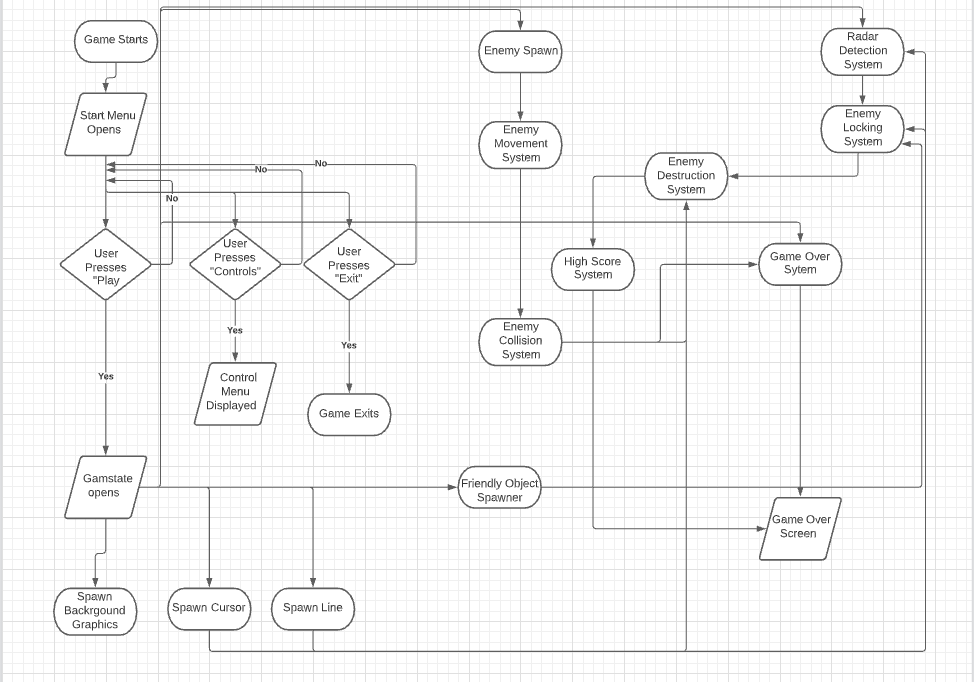
By conducting research online on .ttf froums I have selected the font that I will use in my project. I believe that having a font that fits into the aesthetic of the game would increase the player’s immersion and thus enjoyment The font is called NBP readout. The font is similar to the output of a dot matrix printer. The font has a very retro aesthetic to it and I believe that it would fit my game very well.

Here is the Locked state of the friendly object. Like the friendly unlock the friendly lock only differes from the normal lock sprite in colour. It is also coloured blue. This image is also the size of 20x20 pixels.



## Concept of How The Game Would Flow

### Simplified Flowchart Of The Gameplay



The Flowchart above explains how I conceptualized the game will work.

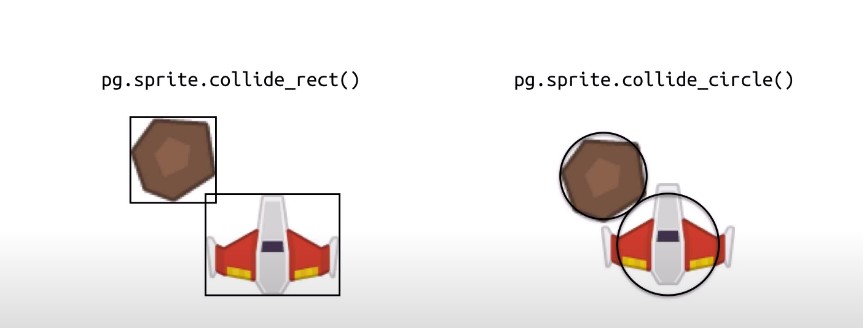
When opened the game will display the start menu. The start menu will give the user 3 buttons to press (Play, Controls, Exit), depending on the User’s input the gamestate might open, the controls menu might open or the game might exit. When the gamestate opens the relevant background graphics will spawn, along with the necessary graphics such as the radar, the enemies, the cursor, the scoreboard.

Different systems will constantly check for certain conditions whilst the game is running and then will carry out certain actions depending on the conditions. These systems will handle processes such as the spawning of the enemies, the movement of the enemies, the detection of the enemies, the locking and destruction of the enemies, movement of the cursor, spawning of the friendly objects, the movement of the friendly objects, the detection and destruction of the friendly objects and whether or not the game is over. When the game is over a screen shall be displayed those outputs the users current and highest ever score. These systems will most likely be represented by my algorithms/functions in my code. The objects present in the flowchart such as the Line, Cursor, Enemies shall be represented by classes in my code

The diagram above will be very helpful for me as I advance through the project as it will provides me a general overview of the tasks I need to complete. It will be beneficial for me in the design stage as it will provide me a point that I will be able to look back at whilst creating my algorithms and class structures.

## Masking

### Collisions In Pygame

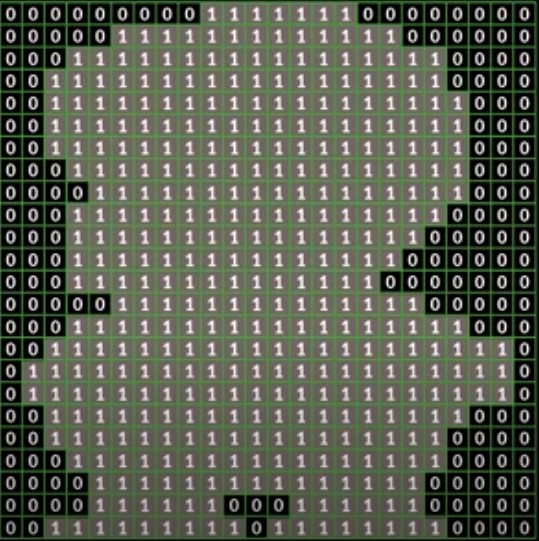
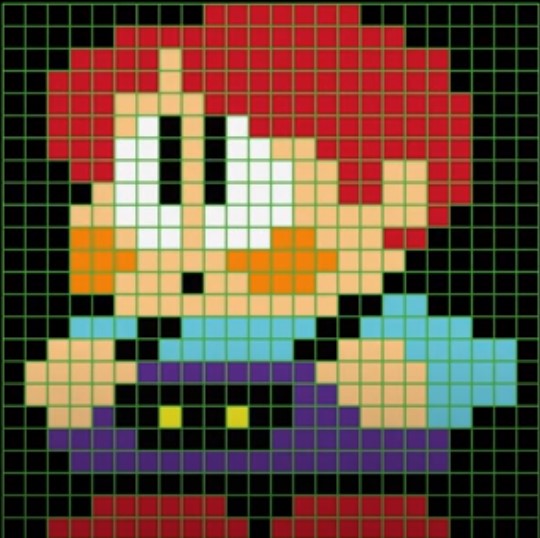


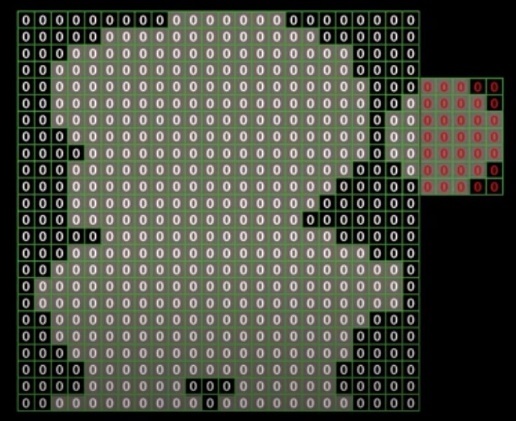
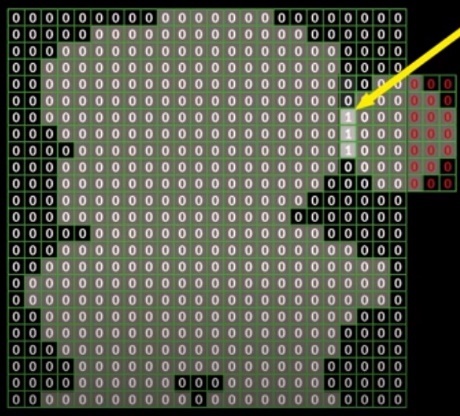
In pygame two types of collision functions are commonly used. These functions check whether or not two different sprites have collided or not. These are collide\_rect and collide\_circle. The default type of collision in Pygame is using the collide\_rect() function, which uses the two sprites’ rect attributes to calculate whether they overlap. This is called AABB collision and it’s very fast and reliable. However, if the sprite images are not rectangles, then you get a situation like in the picture. In the pictures the rectangles overlap, so collide\_rect() is True. The ways they are utilized is that when these sprites are being declared, rough approximations of their shapes are made. As sprites do not tend to just be rectangles or circles these approximations are not perfect. For example, in the image above we can see that the approximations of the surface of the sprites via circles or rectangles is not accurate as the sprite ends up larger/smaller than it actually should be in certain areas. As the objects that the radar will have to detect in my game consist of rectangles one would assume that I could get away with utilizing the collide\_rect() function. However, I will utilize a more precise method to ensure that I will get pixel perfect collisions. This method is called masking

### Rotations In Pygame

In pygame if you were to rotate and image once by an angle around its axis of 10 degrees the object would rotate 10 degrees around its own axis, however the image’s quality would decrease by a negligible amount. This is not problematic if you do not need to constantly rotate the object constantly around its own axis. However, if you were constantly rotating an object around its own axis in pygame you would encounter a number of issues. For example if you had an image with a surface, and then you rotated that surface by one degree using pygame.transform.rotate(image,angle) the rotated version of this image would be painted on the screen, with a indistinguishably low quality difference. If you were to rotate by two degrees, the surface that was created via the rotation of one degree would be recreated but rotated by two degrees. As you can see a small pattern is forming. By every rotation the quality of the image is decreasing, this, as previously stated, would not be noticed when using individual rotations. However, if one was to rotate an object around its axis 1 degrees every frame there would be an exponential decrease in quality every second as normally a game is 30 fps. This is a big problem for me as in order to create the radar line I need a sprite that can rotate around its own axis. However, if I was to create a line and then constantly rotate it around its own axis at the rate that I want it to pygame would crash. A way of solving this problem would be to create new lines every frame, deleting the previous lines, in order to simulate the radar. However, this would create a problem as the surface of the lines would be different from one other due to the way pygame simulates surfaces. This would create problems in collision registration.

### What Is Masking and Why It Would Be Ideal for Me

Masking creates a Mask object from the given surface by setting all the opaque pixels and not setting the transparent pixels. If the surface uses a color-key, then it is used to decide which bits in the resulting mask are set. All the pixels that are not equal to the color-key are set and the pixels equal to the color-key are not set.A mask is a representation of an image that tracks which pixel is transparent and which is not. For every pixel in the image there is bit representation in the mask. If the pixel in image is transparent then the bit is set to 0 or else it is set as 1. Because the only way you can do pixel perfect collision detection is by checking pixel by pixel if the non-transparent parts of the images overlap. And masks are an efficient way of doing this checking. In simple terms a mask is bit representation of an image. Since a comparison can compare 32 bits in a single operation the mask comparison can do faster pixel by pixel comparison than a direct pixel by pixel comparison.

For example, if one was to mask the image on the left, the end result would be the image on the right. The colored bits that are inside the surface are now represented by 1’s, the bits that are not colored are represented by 0’s. Normally the surface would be the rectangle the sprite is in but now via masking the surface is just the area the individual pixels of the sprite take up. As you can see masking results in much more precise approximation of the area encompassed by the sprites by creating a bitmap instead of creating a rough approximation

The images above show how masking makes collisions “pixel perfect”. If masking had not been used the ball and the sprite of the boy would’ve collided prematurely. But now by creating masks of the two objects the premature collision is avoided. This can easily be done in pygame via the mask.collide function. Masking will be very important in creating my radar as , due to the points I previously outlined, pygame does not handle objects rotating around its own axis very well especially the surface’s of the objects. A way to bypass this is to create multiple lines per frame in order to maintain the illusion that a line is rotating around its own axis. However when it is hard to measure the surface and collisions of line in pygame as you create them via a function. In order to create a radar system that has functioning collisions I will create a mask the size of the area of the circle and then update the mask of the circle every time a line is created in the surface of the circle.

## Variables and Data Structures

|  |
| --- |
| Class: Enemy |
| Public procedure new (self, Image, X coordinate, Y coordinate, X acceleration Y acceleration)  Image = png/jpeg  X coordinate= Integer  Y coordinate = Integer  X acceleration =Integer  Y acceleration = Integer  Lock Time = Integer  Mask = bitmap  Endprocedure |
| ChangeState(self, Image)  Update() |

### PsuedoCode Algorithm and flowchart for ChangeState:

#### PseudoCode:

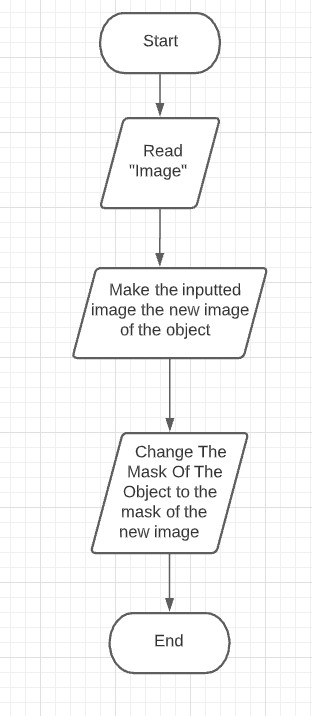
ChangeState(Self,Image):

Self.Image = Image

Self.Mask = mask(image)

endprocedure

#### Flowchart:



Explanation: This procedure is used to change the image that represents an Enemy sprite. This will be used to switch between different states such as being locked, unlocked or invisible to radar. After the image is changed the object is re masked so that it will be able to simulate collisions after the image has changed

Explanation: This method is used in order to move an enemy object. The acceleration in the x and y axis will be set via the spawning function. This method will be called every frame

### PsuedoCode Algorithm and flowchart for Update():

#### PseudoCode:

X coordinate += X acceleration

Y coordinate += Y acceleration

While Lock Time>0:

Lock Time

|  |
| --- |
| Class: Cursor |
| Image = png/jpeg  X coordinate= Integer  Y coordinate = Integer  Mask = bitmap |
| Update() |

### PsuedoCode Algorithm and flowchart for Update():

#### PseudoCode:

X coordinate= mouse.get\_pos\_x

Y coordinate= mouse.get\_pos\_y

|  |
| --- |
| Class: Friendly |
| Public procedure new \_\_(self, Image, X coordinate, Y coordinate, X acceleration Y acceleration)  Image = png/jpeg  X coordinate= Integer  Y coordinate = Integer  X acceleration =Integer  Y acceleration = Integer  Lock Time = Integer  Mask = bitmap  Endprocedure  (this could also inherit from the enemy class as they are near identical |
| ChangeState(self, Image)  Update() |

### PsuedoCode Algorithm and flowchart for ChangeState:

#### PseudoCode:

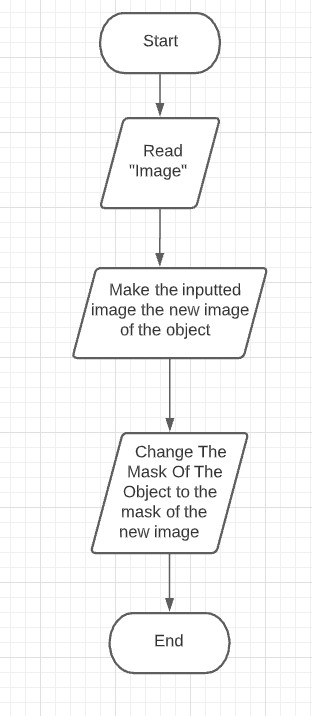
ChangeState(Self,Image):

Self.Image = Image

Self.Mask = mask(image)

endprocedure

#### Flowchart:



Explanation: This procedure is used to change the image that represents an Friendly sprite. This will be used to switch between different states such as being locked, unlocked or invisible to radar. After the image is changed the object is re masked so that it will be able to simulate collisions after the image has changed

Hggg

### PsuedoCode Algorithm and flowchart for Update():

Explanation: This method is used in order to move an enemy object. The acceleration in the x and y axis will be set via the spawning function. This method will be called every frame

#### PseudoCode:

X coordinate += X acceleration

Y coordinate += Y acceleration

|  |
| --- |
| Class: Line |
| init\_\_(self, Image, X coordinate, Y coordinate, X acceleration Y acceleration)  Image = surface(800,800)  X coordinate= Integer  Y coordinate = Integer  Angle = integer  Color = (0,255,0) |
| Update() |

### Vectors | Boundless PhysicsPsuedoCode Algorithm and flowchart for Update():

#### PseudoCode:

angle +=4

if angle >=360:

deg=0

dx = x/2 + x/2\* math.cos(math.radians(self.angle-.1))

dy = y/2 + x/2 \* math.sin(math.radians(self.angle-.1))

image.fill(0)

draw.aaline(self.image, (0, int(255/(1)), 0), (int(x/2), int(y/2)), (dx, dy),5)

mask = pygame.mask.from\_surface(self.image)

Explanation: The code on the left might seem quite complex. In layman’s terms this code creates a line that spins around its own axis. The sprite will be updated every frame. The angle increases by 4 degrees per second. X and y are two variables that have the value 800 and represent the length and width of the surface. The midpoint of the surface, therefore the centre of the circle is x and y divided by tow. I have set up two variables inside this function called dx and dy. These variables represent the y and x endpoints of the line. In order these endpoints I will have to utilise my maths and trig knowledge.

One way of achieving this task is by visualising the distance between every endpoint and the canter of the circle as a vector. When you visualise the vertical and horizontal components of vector a right-angled triangle will form. The hypotenuse of this triangle will be the radius of the circle, the angle between the hypotenuse and the horizontal component from the radius will the value of the angle variable that we have declared. We will take the radius of the circle to be 400 as it is half the length of the surface that we have declared. The horizontal component of the vector will take the form dx (an abbreviation of distance x) and the vertical component of the vector will take the form of dy (an abbreviation of distance y). Using trig you would obtain the vertical component by multiplying the magnitude of the vector (the radius) with the sine of its angle, similarly to obtain the horizontal component you would multiply the magnitude of the vector with the cos of its angle. After obtaining these components you would add these values to the centre points (400,400) and thus obtain the an endpoint(dx,dy). In order to give a more authentic look I utilised anti-aliased lines instead of normal lines. Instead of printing the lines onto the surface of the screen as one would normally do in a game, I print the lines onto the surface of the image. This is in order to utilise masking. After I print the line into the surface of the image I mask the image, which then allows for collisions to be simulated with the line as the other black pixels in the 800x800 image are not part of the mask. In order to prevent the game from crashing by the green lines from printed every frame I fill the image with the colour black, which in essence deletes the previous line before a new one is printed as the mask does not recognise the previous line due to the fact that it has been filled over with the black colour. Utilising black also allows this process to go unnoticed as the background will also be black.

### Spawn Procedure

#### Pseudo Code of the Spawn Procedure

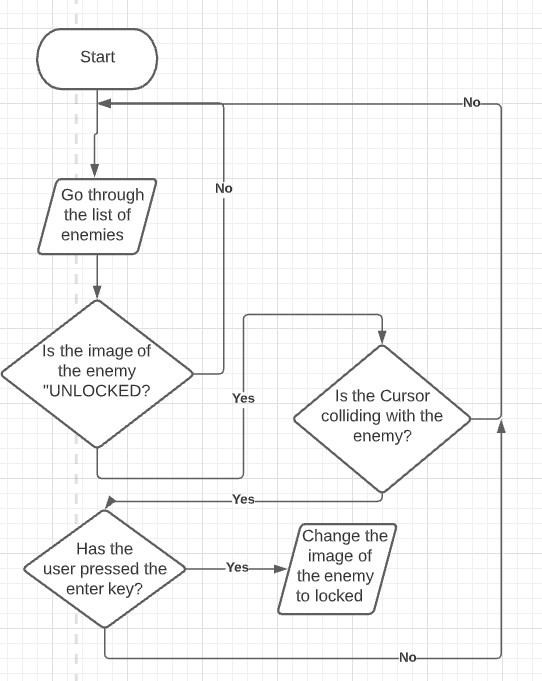
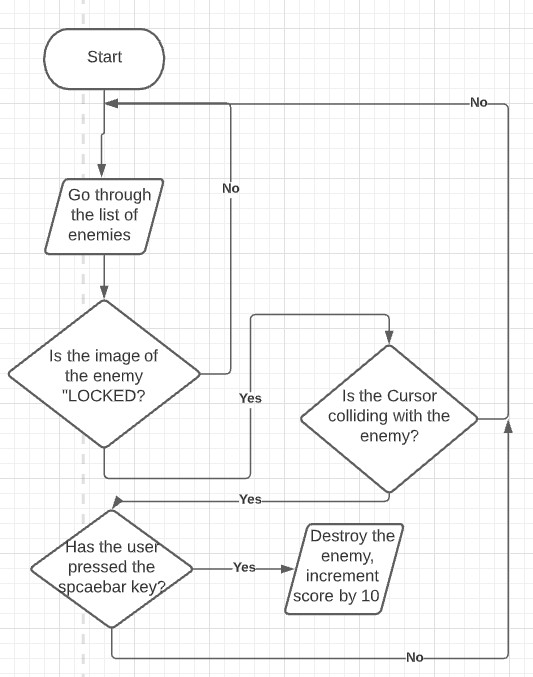
EnemySpawn(number)

|  |
| --- |
|  |
|  | locations=[]  global enemy\_count |
|  | locations = random.sample(range(8),number) |
|  | for i in range(number): |
|  | if locations[i]==1: |
|  | enemy=Enemy(UNLOCK,400,100,0,1) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1  friendly=Friendly(UNLOCK\_F,150,100,1,1)  all\_sprites\_list.add(friendly)  friendly\_list.add(friendly) |
|  | elseif locations[i] == 2: |
|  | enemy=Enemy(UNLOCK,150,100,1,1) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1 |
|  | elseif locations[i] == 3: |
|  | enemy=Enemy(UNLOCK,100,400,1,0) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1  friendly=Friendly(UNLOCK\_F,150,600,1,-1)  all\_sprites\_list.add(friendly)  friendly\_list.add(friendly) |
|  | elseif locations[i] == 4: |
|  | enemy=Enemy(UNLOCK,150,600,1,-1) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1 |
|  | elseif locations[i] == 5: |
|  | enemy=Enemy(UNLOCK,400,700,0,-1) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1 |
|  | elseif locations[i] == 6: |
|  | enemy=Enemy(UNLOCK,600,600,-1,-1) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_count =+ 1 |
|  | enemy\_list.add(enemy) |
|  | elseif locations[i] == 7: |
|  | enemy=Enemy(UNLOCK,700,400,-1,0) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1  friendly=Friendly(UNLOCK\_F,650,150,-1,1)  all\_sprites\_list.add(friendly)  friendly\_list.add(friendly) |
|  | elseif locations[i] == 8: |
|  | enemy=Enemy(UNLOCK,650,150,-1,1) |
|  | all\_sprites\_list.add(enemy) |
|  | enemy\_list.add(enemy) |
|  | enemy\_count =+ 1 |
|  | locations=[]  endprocedure |

Explanation: Above is the pseudocode for a simple spawning procedure. The procedure takes the variable number as a parameter. The number represents how many enemies will be spawned. The procedure also utilises enemy count, in order to keep track of the number of enemies that will be present on the screen. In order to make the enemies spawn in a random order, I generate a number amount of random integers and store them in the locations array. Each of these integers correspond to a certain spawning location. Depending on the relative positions of the locations to the circle the enemies have certain x and y acceleration values. I iterate through the locations array and spawn enemies in the corresponding locations. In order to also spawn friendly units, I have randomly selected some locations where they could spawn. In order to prevent them from spawning ontop of the enemies, which would create a lot of problems, I just make them spawn at a different position from the enemy. I am making the enemies follow a pre-determined path instead of utilising an A\* algorithm or any other path finding algorithm as I believe that it would be unnecessarily complex. I of course add the sprites that I have created to their respective lists as not doing so would create problems when trying to create collisions.

### Destruction and Locking Procedure.

#### Flowchart



#### Pseudocode

For enemy in enemy\_list:

If enemy.Image=”UNLOCKED”:

If mask\_collide(Cursor,enemy) ==True:

If key\_pressed == “ENTER”:

Enemy.Image=LOCKED

For enemy in enemy\_list:

If enemy.Image=”LOCKED”:

If mask\_collide(Cursor,enemy) ==True:

If key\_pressed == “SPACEBAR”:

Enemy\_list.remove(enemy)

Score+=10

Above is a simple outline of the destruction and locking system that I will utilise. The code will scan through the list of enemies, checking for enemies that have the “UNLOCKED” image i.e the enemies that have been detected by the radar. It then checks if the enemies that have been detected by the enemies have collided with the cursor, if so it checks if the user has pressed the enter key, the key that locks the enemy. If so, the enemy’s image is changed to that of a locked image. The second part of the function scans through enemies that have the LOCKED image, i.e enemies that have been locked. It then checks if the cursor is above the enemy, and then checks if the enemy has pressed spacebar. If all these conditions are valid it removes the enemy from the list, essentially deleting it.

### Collision Procedure

For enemy in enemy\_list:

If enemy.Image=”INVIS”:

If mask\_collide(Line,enemy) ==True:

nemy.Timer

Score+=10

Above is a simple outline of the collision/detection system that I will utilise. This will allow the enemies and friendlies to be detected by the spinning radar line.

K

## Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Description** | **Input** | **Expected Outcome** | **Justification** |
| 1 | Screen appears and is 1000 by 1000 pixels | Run the code | Screen appears on the display | In order to test whether or not the display is working and if it fits the parameters, I determined |
| 2 | Does The Cursor move right? | Move the mouse to the right of the screen | Cursor image moves towards the right of the screen | In order to test if the cursor can move. The cursor is a crucial part of my game, if it cannot move the game will not properly function |
| 3 | Does The Cursor move left? | Move the mouse to the left of the screen | Cursor image moves towards the left of the screen | In order to test if the cursor can move. The cursor is a crucial part of my game, if it cannot move the game will not properly function |
| 4 | Does the Cursor move up ? | Move the mouse to the top of the screen | Cursor image moves towards the top of the screen | In order to test if the cursor can move. The cursor is a crucial part of my game, if it cannot move the game will not properly function |
| 5 | Does the Cursor move down ? | Move the mouse to the bottom of the screen | Cursor image moves towards the bottom of the screen | In order to test if the cursor can move. The cursor is a crucial part of my game, if it cannot move the game will not properly function |
| 6 | Does the sprite image of the enemy’s changes when locked? | Press the enter key | The image of the enemy would switch from unlocked to locked | In order to test whether the backbone of my locking system is working, as later the image must switch when it is locked via the cursor |
| 7 | Does the sprite image of the friendly changes when locked ? | Press the enter key | The image of the friendly would switch from unlocked to locked | In order to test whether the backbone of my locking system is working, as later the image must switch when it is locked via the cursor |
| 8 | Does the sprite image of the enemy’s changes when unlocked? | Press the enter key | The image of the enemy would switch from locked to unlocked | In order to test whether the backbone of my locking system is working, the player must be able to also unlock the targets they lock |
| 9 | Does the sprite image of the friendly changes when unlocked? | Press the enter key | The image of the friendly would switch from locked to unlocked | In order to test whether the backbone of my locking system is working, the player must be able to also unlock the friendlies they lock especially as if they do not unlock and destroy the friendlies the game will be over. |
| 10 | Do enemies appear on the screen? | Run the code | An enemy object appears | In order to test whether or not my enemies can be spawned onto the screen, as they are a crucial part of my game them not being able to spawn would be a big problem |
| 11 | Do friendlies appear on the screen? | Run the code | An friendly object appears | In order to test whether or not my friendlies can be spawned onto the screen, as they are a crucial part of my game them not being able to spawn would be a big problem |
| 12 | Does the Base appears in the middle of the screen? | Run the code | The Base appears in the middle of the screen | As the base is an important part of my game, the player must be able to see it in order to know where he is defending from the enemies. If the base does not spawn a crucial part of my game would not be present |
| 13 | Do the background aesthetics appear? | Run the code | The background aesthetics appear | The background aesthetics are not crucial to the gameplay, however their presence of these background aesthetics is essential to the players immersion |
| 14 | Does the scoreboard appears in the left-hand corner of the screen? | Run the code | The scoreboard appears in the left-hand corner of the screen | In order to show the player how many points he/she has accumulated whilst playing |
| 15 | Can an Enemy be destroyed? | Press the enter key then press spacebar | The enemy would be deleted from the screen | In order to test whether or not I can destroy enemy objects as this is an essential part of the gameplay. If I cannot destroy the enemies the game would not function properly. |
| 16 | Does the score increase when an enemy is destroyed? | Destroy the enemy/ Press the enter key then press spacebar | The score displayed on the left hand corner of the screen would be incremented by 10 | In order to test whether or not my score system is working. If the score did not increment then I would not be able to create a high score system nor would I be able to add any replay value to my game |
| 17 | Does the game detect when the enemies have collided with the base? | Add code that causes the game to exit when a collision occurs | When an enemy reaches the base the game exits | This is in order to simply show to me that the game can recognize the collisions between the enemies and the base. If the game cannot recognize the collisions then the game would go on forever. |
| 18 | Can a Friendly be destroyed? | Press the enter key then press spacebar | The Friendly would be deleted from the screen | In order to test whether or not I can destroy friendly objects as this is an essential part of the gameplay. If I cannot destroy the friendlies the game would not function properly. |
| 19 | Does the game end when a friendly is destroyed? | Destroy the enemy/ Press the enter key then press spacebar | The game would exit. | This is in order to simply show to me that the game can recognize the fact that it is game over when the user destroys the friendly object. |
| 20 | Can the Gane Over Screen be displayed? | Cause a game over event i.e enemy hits the base, player destroys friendly. | The Game Over Screen Would be Displayed. | This is in order to check whether or not I am able to switch between different screens (star,game and game over screen). It is also to check whether the game properly recognizes if the game is over. |
| 21 | Does the game store a high score? | Record a high score, then beat that high score, | The high score would change after the player beat it | This is in order to ensure that the competitive element of my game will function. If this was not to function then there would be no reason to play this game for an extended duration. |
| 22 | Does the Start button work? | Press the start button in the opening menu | The game would begin | This is in order to ensure the user is able to start playing the game. If the start button did not work then the game would be stuck on the opening menu |
| 23 | Does the Controls button work? | Press the control button in the opening menu | The control menu would open. | This is in order to ensure the user is able to access the controls menu. The controls menu is essential to the game, as it shows the player the controls of the game, how to play the game and also how the game flows. |
| 24 | Does the Quit button work? | Press the quit button in the opening menu | The game would quit/exit | This is in order to ensure the user is able to quit the game in the opening menu. As this button one of the first things the user sees upon launching the game, this button not working would leave a bade impression |
| 25 | Does the radar line spin around its own axis? | Run the game | The line would spin around its own axis | As utilizing a radar is a big part of the game that I intend to develop, I need to create a physical object that resembles/looks like a real radar. |
| 26 | Can the radar line collide with the objects in the game? | Add code that causes the game to exit when a collision occurs | When the line collides with and enemy/friendly the game would exit | In order for the enemies to be detected by the radar the program must be able to detect the collisions between the radar line and the objects. If the collisions do not work, creating the game would be hard as it would be impossible to implement the detection mechanics. |
| 27 | Do the friendlies move towards the center of the screen? | Run the game | The friendlies would move towards the center of the screen. | This is in order to test if the friendlies are moving towards the point, they should, if they do not move towards the center the game would not properly function as they would not mimic the movement of the enemies. |
| 28 | Do the enemies move towards the center of the screen? | Run the game | The enemies would move towards the center of the screen. | This is in order to test if the enemies are moving towards the point, they should, if they do not move towards the center the game would not properly function as they would not collide with the base |
| 29 | Can the enemies be invisible to the radar | Run the game and spawn enemies. | The enemies will not appear on the screen | This is in order test if the detection mechanics are working or not. This would be achieved by changing the sprite image to the transparent one and checking if that hides it or not. |
| 30 | Does audio play in the background? | Run the game | You would be able to hear the music in the background/ the music would be playing. | My shareholder/clients requested that music play in the background of the game so I will need to test if the background music playing or not. |

# Implementation

## Iteration 1

### 

### The requirements that I am to develop:

Requirement 1: A window shall have the size of 1000x1000 pixels

Requirement 4: The user moves around a cursor that will be used to destroy the enemy

Requirement 5: The user inputs will be managed by the mouse and keyboard

Requirement 7: The player move the cursor around via mouse inputs

Requirement 8: The user utilize keyboard inputs to lock and destroy enemies

Requirement 12: A line spinning around its own axis in the center of the screen

Requirement 13: This line should be able to collide with the enemies

Requirement 14: A system that allows the enemies to become visible when detected by the radar.

Requirement 23: A system that changes the visuals of the enemy when they are locked

### Justification:

This first iteration will allow me to set up the core elements of the game that I want to create. User input handling is very important for the game and is one of the first things that must be delt with when developing. The enemy and friendly changing visual states is very important for my game as it is the backbone of my detection mechanic. Furthermore, creating a line that spins around its own axis and can detect collisions is very important for the radar mechanic of the game.

I will create small test files that will act of proof of concepts for individual or multiple requirements in order to prevent creating complex files that could be derailed with one bad line of code.

import pygame

# Define some colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

class enemy(pygame.sprite.Sprite):

    """ The class is the player-controlled sprite. """

    def \_\_init\_\_(self, color):

        """Constructor function"""

        # Call the parent's constructor

        super().\_\_init\_\_()

        # Set height, width

        self.image = pygame.Surface([40, 40])

        self.image.fill(color)

        self.color=color

        self.move=1

        self.move\_y=1

        # Make our top-left corner the passed-in location.

        self.rect = self.image.get\_rect()

    def change\_color(self, color):

        self.color = color

        self.image.fill(self.color)

    def update(self):

        self.rect.x += self.move\_y

pygame.init()

# Set the width and height of the screen [width, height]

size = (700, 500)

screen = pygame.display.set\_mode(size)

pygame.display.set\_caption("press left arrow for it to lose track")

# Loop until the user clicks the close button.

done = False

# Used to manage how fast the screen updates

clock = pygame.time.Clock()

all\_sprites\_list = pygame.sprite.Group()

enemy = enemy((0,255,0))

all\_sprites\_list.add(enemy)

# -------- Main Program Loop -----------

while not done:

    # --- Main event loop

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            done = True

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_LEFT:

                if enemy.color == (0,255,0):

                    enemy.change\_color((0,0,0))

                elif enemy.color == (0,0,0):

                    enemy.change\_color((0,255,0))

    # --- Game logic should go here

    all\_sprites\_list.update()

    # --- Screen-clearing code goes here

    # background image.

    screen.fill(BLACK)

    # --- Drawing code should go here

    all\_sprites\_list.draw(screen)

    # --- Go ahead and update the screen with what we've drawn.

    pygame.display.flip()

    # --- Limit to 60 frames per second

    clock.tick(60)

# Close the window and quit.

pygame.quit()

import os

import random

import pygame

# Define some colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

UNLOCK = pygame.image.load('pixilart-drawing.png')

LOCK = pygame.image.load('pixil-frame-0.png')

CURSOR = pygame.image.load('pixil-frame-0 (1).png')

INVIS = pygame.image.load('pixil-frame-0 (2).png')

class enemy(pygame.sprite.Sprite):

    """ The class is the player-controlled sprite. """

    def \_\_init\_\_(self, image):

        """Constructor function"""

        # Call the parent's constructor

        super().\_\_init\_\_()

        # Set height, width

        self.image = pygame.Surface([40, 40])

        self.image=image

        # Make our top-left corner the passed-in location.

        self.rect = self.image.get\_rect()

        self.rect.y = 100

        self.rect.x = 70

    def change\_state(self, image):

        self.image = image

class cursor(pygame.sprite.Sprite):

    """ The class is the player-controlled sprite. """

    def \_\_init\_\_(self, image,x ,y):

        """Constructor function"""

        # Call the parent's constructor

        super().\_\_init\_\_()

        # Set height, width

        self.image = pygame.Surface([60, 60])

        self.rect = self.image.get\_rect()

        self.rect.y = y

        self.rect.x = x

        self.change\_x = 0

        self.change\_y = 0

        self.image=image

        # Make our top-left corner the passed-in location.

pygame.init()

# Set the width and height of the screen [width, height]

size = (700, 500)

screen = pygame.display.set\_mode(size)

pygame.display.set\_caption("press enter to lock the target")

# Loop until the user clicks the close button.

done = False

# Used to manage how fast the screen updates

clock = pygame.time.Clock()

all\_sprites\_list = pygame.sprite.Group()

enemy = enemy(UNLOCK)

cursor = cursor(CURSOR,340,240)

all\_sprites\_list.add(enemy)

all\_sprites\_list.add(cursor)

# -------- Main Program Loop -----------

while not done:

    # --- Main event loop

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            done = True

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_LEFT:

                cursor.rect.x -= 10

            elif event.key == pygame.K\_RIGHT:

                cursor.rect.x += 10

            elif event.key == pygame.K\_UP:

                cursor.rect.y -= 10

            elif event.key == pygame.K\_DOWN:

                cursor.rect.y += 20

            elif event.key == pygame.K\_RETURN:

                if pygame.sprite.collide\_rect(cursor,enemy) == True:

                    enemy.change\_state(LOCK)

                else:

                    enemy.change\_state(UNLOCK)

    # --- Game logic should go here

    all\_sprites\_list.update()

    # --- Screen-clearing code goes here

    # Here, we clear the screen to white. Don't put other drawing commands

    # above this, or they will be erased with this command.

    # If you want a background image, replace this clear with blit'ing the

    # background image.

    screen.fill(BLACK)

    # --- Drawing code should go here

    all\_sprites\_list.draw(screen)

    # --- Go ahead and update the screen with what we've drawn.

    pygame.display.flip()

    # --- Limit to 60 frames per second

    clock.tick(60)

# Close the window and quit.

pygame.quit()

import math

import pygame

class SpriteObject(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y, image):

        super().\_\_init\_\_()

        self.image = image

        self.rect = self.image.get\_rect(center = (x, y))

        self.mask = pygame.mask.from\_surface(self.image)

    def update(self):

        self.rect.center = pygame.mouse.get\_pos()

class Line(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.image = pygame.Surface((200, 200))

        self.image.set\_colorkey((0, 0, 0))

        self.rect = self.image.get\_rect(center = (x, y))

        self.angle = 0

    def update(self):

        vec = round(math.cos(self.angle \* math.pi / 180) \* 100), round(math.sin(self.angle \* math.pi / 180) \* 100)

        self.angle = (self.angle + 1) % 360

        self.image.fill(0)

        pygame.draw.line(self.image, (255, 255, 0), (100 - vec[0], 100 - vec[1]), (100 + vec[0], 100 + vec[1]), 5)

        self.mask = pygame.mask.from\_surface(self.image)

pygame.init()

window = pygame.display.set\_mode((500, 500))

clock = pygame.time.Clock()

sprite\_image = pygame.image.load('CURSOR.png').convert\_alpha()

moving\_object = SpriteObject(0, 0, sprite\_image)

line\_object = Line(\*window.get\_rect().center)

all\_sprites = pygame.sprite.Group([moving\_object, line\_object])

red = 0

run = True

while run:

    clock.tick(60)

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            run = False

    all\_sprites.update()

    if pygame.sprite.collide\_mask(moving\_object, line\_object):

        red = min(255, red+4)

    else:

        red = 0

    window.fill((red, 0, 0))

    all\_sprites.draw(window)

    pygame.display.flip()

pygame.quit()

exit()

import sys

import pygame

import math

import random

from time import sleep

from pygame.locals import \*

UNLOCK = pygame.image.load('UNLOCK.png')

LOCK = pygame.image.load('LOCK.png')

CURSOR = pygame.image.load('CURSOR.png')

INVIS = pygame.image.load('INVIS.png')

# Define some colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

(x,y) = (800,800)

deg = 0

enemy\_count= 0

pygame.init()

pygame.display.set\_mode((1000, 1000), 0, 32)

screen = pygame.display.get\_surface()

pygame.init()

all\_sprites\_list = pygame.sprite.Group()

enemy\_list = pygame.sprite.Group()

class SpriteObject(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y, image):

        super().\_\_init\_\_()

        self.image = image

        self.rect = self.image.get\_rect(center = (x, y))

        self.mask = pygame.mask.from\_surface(self.image)

    def update(self):

        self.rect.center = pygame.mouse.get\_pos()

class Line(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.image = pygame.Surface((800, 800))

        self.image.set\_colorkey((0, 0, 0))

        self.rect = self.image.get\_rect(center = (400, 400))

        self.angle = 0

    def update(self):

        self.angle +=1

        if self.angle >=360:

            deg=0

        i =1

        dx = x/2 + x/2 \* math.cos(math.radians(self.angle-.1\*i))

        dy = y/2 + x/2 \* math.sin(math.radians(self.angle-.1\*i))

        f = 1\*.1

        self.image.fill(0)

        pygame.draw.aaline(self.image, (0, int(255/(1+f)), 0), (int(x/2), int(y/2)), (dx, dy),5)

        self.mask = pygame.mask.from\_surface(self.image)

pygame.display.set\_caption("Missile Command Redux")

# Loop until the user clicks the close button.

done = False

# Used to manage how fast the screen updates

clock = pygame.time.Clock()

moving\_object = SpriteObject(0, 0, CURSOR)

all\_sprites\_list.add(moving\_object)

radar =Line(400,400)

all\_sprites\_list.add(radar)

while not done:

    # --- Main event loop

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            done = True

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_LEFT:

                cursor.rect.x -= 10

            elif event.key == pygame.K\_RIGHT:

                cursor.rect.x += 10

            elif event.key == pygame.K\_UP:

                cursor.rect.y -= 10

            elif event.key == pygame.K\_DOWN:

                cursor.rect.y += 20

    if pygame.sprite.collide\_mask(moving\_object, radar):

        exit()

    else:

        red = 0

    all\_sprites\_list.update()

    pygame.display.update()

    clock.tick(60)

    screen.fill((0, 0, 0, 0))

    # --- Screen-clearing code goes here

    # Here, we clear the screen to white. Don't put other drawing commands

    # above this, or they will be erased with this command.

    all\_sprites\_list.draw(screen)

    # If you want a background image, replace this clear with blit'ing the

    # background image.

    # --- Drawing code should go here

    # --- Go ahead and update the screen with what we've drawn.

    pygame.display.flip()

    # --- Limit to 60 frames per second

# Close the window and quit.

pygame.quit()

import os

import sys

from typing import Tuple

import pygame

import math

import random

from time import sleep

from pygame.locals import \*

## simplifiying image declerations so code doesnt become swamped

UNLOCK = pygame.image.load('UNLOCK.png')

LOCK = pygame.image.load('LOCK.png')

CURSOR = pygame.image.load('CURSOR.png')

INVIS = pygame.image.load('INVIS.png')

BASE = pygame.image.load('base.png')

# Define some colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (0, 255, 0)

BLUE=(0,0,255)

RED = (255, 0, 0)

(x,y) = (800,800)

deg = 0

enemy\_count= 0

game\_end= True

pygame.init()

font = "NbpReadout-RBVA.ttf"

score\_font = pygame.font.Font('NbpReadout-RBVA.ttf',12)

pygame.display.set\_mode((1000, 1000), 0, 32)

screen = pygame.display.get\_surface()

pygame.init()

## global declaration for easier access in loops

global score

score= 10

all\_sprites\_list = pygame.sprite.Group()

enemy\_list = pygame.sprite.Group()

radar\_list = pygame.sprite.Group()

class Cursor(pygame.sprite.Sprite):

    """ The class is the player-controlled sprite. """

    def \_\_init\_\_(self, image,x ,y):

        """Constructor function"""

        # Call the parent's constructor

        super().\_\_init\_\_()

        # Set height, width

        self.image = pygame.Surface([40, 40])

        self.rect = self.image.get\_rect()

        self.rect = self.image.get\_rect(center = (x, y))

        self.image=image

    def update(self):

        self.rect.center = pygame.mouse.get\_pos()

class Base(pygame.sprite.Sprite):

    def \_\_init\_\_(self, image,x ,y):

        """Constructor function"""

        # Call the parent's constructor

        super().\_\_init\_\_()

        # Set height, width

        self.image = pygame.Surface([100, 100])

        self.rect = self.image.get\_rect()

        self.rect = self.image.get\_rect(center = (x, y))

        self.image=image

class Enemy(pygame.sprite.Sprite):

    def \_\_init\_\_(self, image, x\_pos, y\_pos,x\_d,y\_d):

        super().\_\_init\_\_()

        # Set height, width

        self.image = pygame.Surface([20, 20])

        self.image= image

        self.mask = pygame.mask.from\_surface(self.image)

        self.y\_d = y\_d

        self.x\_d = x\_d

        self.lock\_time = 0

        self.rect = self.image.get\_rect()

        self.rect.y = y\_pos

        self.rect.x = x\_pos

        self.detected = False

        # function that allows the enemy to switch between detection modes

    def change\_state(self, image):

        #print ('test')

        self.image = image

        self.mask = pygame.mask.from\_surface(self.image)

    def update(self):

        self.rect.x += self.x\_d

        self.rect.y += self.y\_d

class Friendly(Enemy):

    pass

#the line is the radar cursour that is spinning

class Line(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        self.image = pygame.Surface((800, 800))

        self.image.set\_colorkey((0, 0, 0))

        self.rect = self.image.get\_rect(center = (400, 400))

        self.angle = 0

    def update(self):

        self.angle +=4

        if self.angle >=360:

            deg=0

#

#mathematical equations that allow the lines to spin a circle whilist varying its length

        dx = x/2 + x/2 \* math.cos(math.radians(self.angle-.1))

        dy = y/2 + x/2 \* math.sin(math.radians(self.angle-.1))

        f = .1

        self.image.fill(0)

        pygame.draw.aaline(self.image, (0, int(255/(1+f)), 0), (int(x/2), int(y/2)), (dx, dy),5)

        self.mask = pygame.mask.from\_surface(self.image)

def MAP():

    pygame.draw.circle(screen, (102, 255, 102), (int(x/2), int(y/2)), int(x/2), 1)

    pygame.draw.circle(screen, (102, 255, 102), (int(x/2), int(y/2)), int(x/4), 1)

    pygame.draw.line(screen, (76, 82, 76), (250, 190), (200, 55))

    pygame.draw.line(screen, (76, 82, 76), (300, 200), (250, 190))

    pygame.draw.line(screen, (76, 82, 76), (350, 300), (300, 200))

    pygame.draw.line(screen, (76, 82, 76), (350, 300), (785, 300))

    pygame.draw.line(screen, (76, 82, 76), (340, 360), (0, 400))

    pygame.draw.line(screen, (76, 82, 76), (350, 300), (300, 600))

    pygame.draw.line(screen, (76, 82, 76), (300, 600), (350, 675))

    pygame.draw.line(screen, (76, 82, 76), (370, 796), (350, 675))

# a better alterantive to pygame.font that makes it easier in the long run to print text.

def text\_format(message, textFont, textSize, textColor):

    newFont=pygame.font.Font(textFont, textSize)

    newText=newFont.render(message, 0, textColor)

    return newText

# code that I hoped would create a local highscore file but did not work

#def loadscore():

#    global highscore

#    dir = path.dirname(\_\_file\_\_)

#    try:

#        #try to read the file

#        with open(path.join(self.dir, HS\_FILE), 'r+') as f:

#            highscore = int(f.read())

#    except:

#        #create the file

#        with open(path.join(self.dir, HS\_FILE), 'w'):

#            highscore = 0

clock = pygame.time.Clock()

def main\_menu():

    menu=True

    controls=False

    selected="start"

    while menu:

        for event in pygame.event.get():

            if event.type==pygame.QUIT:

                pygame.quit()

                quit()

            if event.type==pygame.KEYDOWN:

                if event.key==pygame.K\_UP:

                    selected="start"

                elif event.key==pygame.K\_RIGHT:

                    selected="quit"

                elif event.key==pygame.K\_LEFT:

                    selected="Controls"

                if event.key==pygame.K\_RETURN:

                    if selected=="start":

                        menu = False

                    if selected=="Controls":

                        menu = False

                        controls=True

                    if selected=="quit":

                        pygame.quit()

                        quit()

        # Main Menu UI

        screen.fill(BLACK)

        title=text\_format("Missile Command Redux", font, 50, GREEN)

        if selected=="start":

            text\_start=text\_format("START", font, 40, WHITE)

        else:

            text\_start = text\_format("START", font, 40, GREEN)

        if selected=="Controls":

            text\_controls=text\_format("CONTROLS", font, 40, WHITE)

        else:

            text\_controls = text\_format("CONTROLS", font, 40, GREEN)

        if selected=="quit":

            text\_quit=text\_format("QUIT", font, 40, WHITE)

        else:

            text\_quit = text\_format("QUIT", font, 40, GREEN)

        text\_tutorial1= text\_format("USE THE MOUSE BUTTON TO MOVE THE CURSOR", font, 23, GREEN)

        text\_tutorial2= text\_format("BRING THE CURSOR ONTOP OF THE ENEMIES, THE GREEN BLIPS", font, 23, GREEN)

        text\_tutorial3= text\_format("PRESS THE ENTER KEY TO LOCK THEM", font, 23, GREEN)

        text\_tutorial4= text\_format("WHEN LOCKED PRESS SPACEBAR TO SHOOT THEM DOWN", font, 23, GREEN)

        text\_tutorial5= text\_format("IF THE ENEMIES REACH YOUR BASE IT IS GAME OVER", font, 23, GREEN)

        text\_tutorial6= text\_format("DO NOT SHOOT THE FRIENDLIES", font, 23, BLUE)

        # the reason why there is so many stacked text declarations is because of an error I faced being solved by doing this

        global text\_message

        text\_message= text\_format("HQ IS DOWN", font, 46, GREEN)

        global text\_message2

        text\_message2= text\_format("YOU HAVE FAILED", font, 46, GREEN)

        #text\_back= text\_format("Back", font, 35, GREEN)

        #text\_score= text\_format("Score: "+ str(highscore), font, 35, GREEN)

        title\_rect=title.get\_rect()

        start\_rect=text\_start.get\_rect()

        quit\_rect=text\_quit.get\_rect()

        # Main Menu Text

        screen.blit(title, (1000/2 - (title\_rect[2]/2), 80))

        screen.blit(text\_start, (1000/2 - (start\_rect[2]/2), 300))

        screen.blit(text\_controls, (600/2 - (start\_rect[2]/2), 360))

        screen.blit(text\_quit, (1250/2 - (quit\_rect[2]/2), 360))

        #screen.blit(text\_score, (1000 , 80))

        pygame.display.update()

        clock.tick(60)

    # controls section of the menu

    while controls:

        for event in pygame.event.get():

            if event.type==pygame.QUIT:

                pygame.quit()

                quit()

            if event.type==pygame.KEYDOWN:

                if event.key==pygame.K\_UP:

                    selected="back"

                if event.key==pygame.K\_RETURN:

                    if selected=="back":

                        main\_menu()

                        controls = False

        # Main Menu UI

        screen.fill(BLACK)

        if selected=="back":

            text\_back=text\_format("Back", font, 40, WHITE)

        else:

            text\_back = text\_format("Back", font, 40, GREEN)

        title=text\_format("Controls", font, 50, GREEN)

        text\_quit = text\_format("USE THE MOUSE BUTTON TO MOVE THE CURSOR", font, 20, GREEN)

        # Main Menu Text

        screen.blit(text\_back, (1000/2 - (title\_rect[2]/2), 80))

        screen.blit(title, (1000/2 - (title\_rect[2]/2), 160))

        #screen.blit(title2 , (1000/2 - (title2[2]/2), 80))

        screen.blit(text\_tutorial1, (1000/2 - (480), 300))

        screen.blit(text\_tutorial2, (1000/2 - (480), 360))

        screen.blit(text\_tutorial3, (1000/2 - (480), 420))

        screen.blit(text\_tutorial4, (1000/2 - (480), 480))

        screen.blit(text\_tutorial5, (1000/2 - (480), 540))

        screen.blit(text\_tutorial6, (1000/2 - (480), 600))

        pygame.display.update()

        clock.tick(60)

def Enemy\_Spawn(number):

    # 0 and 1 correspons to negative or positive changes in the x or y axis of movement of the sprites. They spawn at predetermined positions as an excessive amount of algebra would be required to create concentric cirles of spawn positions.

    global enemy\_count

    #for i in range(number):

    if number==1:

        enemy=Enemy(INVIS,400,100,0,1)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 2:

        enemy=Enemy(INVIS,150,100,1,1)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 3:

        enemy=Enemy(INVIS,100,400,1,0)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 4:

        enemy=Enemy(INVIS,150,600,1,-1)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 5:

        enemy=Enemy(INVIS,400,700,0,-1)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 6:

        enemy=Enemy(INVIS,600,600,-1,-1)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 7:

        enemy=Enemy(INVIS,700,400,-1,0)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

    elif number == 8:

        enemy=Enemy(INVIS,650,150,-1,1)

        all\_sprites\_list.add(enemy)

        enemy\_list.add(enemy)

        enemy\_count =+ 1

def detection():

    for e in enemy\_list:

        if (pygame.sprite.collide\_mask(e, radar)) and (e.detected == False) :

            e.change\_state(UNLOCK)

            e.detected= True

    for e in enemy\_list:

        if pygame.sprite.collide\_mask(e, cursor):

            e.change\_state(LOCK)

            score =+ 10

    for e in enemy\_list:

        if pygame.sprite.collide\_mask(e, base):

            game\_end = False

def game\_loop():

    global clock

    clock= pygame.time.Clock()

    global dt

    dt=1

    global timer

    timer=30

    radar =Line(400,400)

    radar\_list.add(radar)

    cursor =Cursor(CURSOR,372,375)

    base = Base(BASE,400,400)

    all\_sprites\_list.add(radar)

    all\_sprites\_list.add(cursor)

    all\_sprites\_list.add(base)

    game\_end=True

    while game\_end:

        # --- Main event loop

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

            elif event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_LEFT:

                    cursor.rect.x -= 10

                elif event.key == pygame.K\_RIGHT:

                    cursor.rect.x += 10

                elif event.key == pygame.K\_UP:

                    cursor.rect.y -= 10

                elif event.key == pygame.K\_DOWN:

                    cursor.rect.y += 20

                elif event.key == pygame.K\_RETURN:

                    for e in enemy\_list:

                        if pygame.sprite.collide\_mask(e, cursor):

                            e.change\_state(LOCK)

                elif event.key == pygame.K\_SPACE:

                    for e in enemy\_list:

                        if e.image == LOCK:

                            e.kill()

                            global score

                            score = score + 10

        # --- Game logic should go here

        #MAP()

        if game\_end == True:

            pygame.draw.circle(screen, (102, 255, 102), (int(x/2), int(y/2)), int(x/2), 1)

            pygame.draw.circle(screen, (102, 255, 102), (int(x/2), int(y/2)), int(x/4), 1)

            pygame.draw.line(screen, (76, 82, 76), (250, 190), (200, 55))

            pygame.draw.line(screen, (76, 82, 76), (300, 200), (250, 190))

            pygame.draw.line(screen, (76, 82, 76), (350, 300), (300, 200))

            pygame.draw.line(screen, (76, 82, 76), (350, 300), (785, 300))

            pygame.draw.line(screen, (76, 82, 76), (340, 360), (0, 400))

            pygame.draw.line(screen, (76, 82, 76), (350, 300), (300, 600))

            pygame.draw.line(screen, (76, 82, 76), (300, 600), (350, 675))

            pygame.draw.line(screen, (76, 82, 76), (370, 796), (350, 675))

        timer-=dt

        if timer <= 0:

            Enemy\_Spawn(random.randrange(1,9))

            timer=60

            #pygame.draw.line(screen, (0, 200, 0), (int(x/2), 0), (int(x/2), y))

        #for enemy in enemy\_list:

            #if pygame.sprite.collide\_mask(enemy,radar):

                #enemy.change\_state(LOCK)

                #enemy.lock\_time = pygame.time.get\_ticks()

        #for enemy in enemy\_list:

            #if pygame.spritecollide\_mask(enemy,radar):

        #detection()

        #

        #currently not working will have to make game end global via forcing it

        #

        #

        for e in enemy\_list:

            if (pygame.sprite.collide\_mask(e, radar)) and (e.detected == False) :

                e.change\_state(UNLOCK)

                e.detected= True

        for e in enemy\_list:

            if pygame.sprite.collide\_mask(e, base):

                game\_end = False

        all\_sprites\_list.update()

        pygame.display.update()

        #OPTIMISATION ISSUE

        #dt=clock.tick(60)/600

        clock.tick(60)

        screen.fill((0, 0, 0, 0))

        # --- Screen-clearing code goes here

        # Here, we clear the screen to white. Don't put other drawing commands

        # above this, or they will be erased with this command.

        all\_sprites\_list.draw(screen)

        # If you want a background image, replace this clear with blit'ing the

        # background image.

        # --- Drawing code should go here

        # --- Go ahead and update the screen with what we've drawn.

        pygame.display.flip()

    scores.append(score)

    score=0

def retry\_menu():

    retry = True

    selected="quit"

    while retry:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit()

            if event.type==pygame.KEYDOWN:

                if event.key==pygame.K\_UP:

                    selected="return"

                elif event.key==pygame.K\_DOWN:

                    selected="quit"

                if event.key==pygame.K\_RETURN:

                    if selected=="return":

                        retry=False

                    if selected=="quit":

                        pygame.quit()

                        quit()

        max\_score=scores[0]

        if playcount>=1:

            for x in range((playcount+1)):

                if scores[x] > max\_score:

                    max\_score= scores[x]

        text\_score= text\_format("You Scored: " + str(scores[playcount]), font, 23, GREEN)

        text\_high\_score= text\_format("Your High Score Is: " + str(max\_score), font, 23, GREEN)

        text\_return= text\_format("Return to Main Menu", font, 40, GREEN)

        text\_quit= text\_format("Quit", font, 40, GREEN)

        if selected == "return":

            text\_return= text\_format("Return to Main Menu", font, 40, WHITE)

        if selected == "quit":

            text\_quit= text\_format("Quit", font, 40, WHITE)

        screen.fill((0, 0, 0, 0))

        screen.blit(text\_high\_score, (1000/2 - (480), 250))

        screen.blit(text\_score, (1000/2 - (480), 300))

        screen.blit(text\_message, (1000/2 - (480), 50))

        screen.blit(text\_message2, (1000/2 - (480), 150))

        screen.blit(text\_return, (1000/2 - (480), 350))

        screen.blit(text\_quit, (1000/2 - (480), 400))

        pygame.display.flip()

def main():

    global playcount

    global scores

    playcount = 0

    scores=[]\*100

    while not done:

        if(playcount > 1):

            for e in enemy\_list:

                e.kill()

            for e in radar\_list:

                e.kill()

        score=0

        main\_menu()

        game\_loop()

        retry\_menu()

        playcount = playcount+1

current\_time=0

# Set the width and height of the screen [width, height]

#def radar\_detection

#def lock

pygame.display.set\_caption("Missile Command Redux")

# Loop until the user clicks the close button.

done = False

# Used to manage how fast the screen updates

# -------- Main Program Loop -----------

##black=(0,0,0)

##

##end\_it=False

##

##while (end\_it==False):

##    screen.fill(black)

##    myfont=pygame.font.SysFont("NBP Readout", 40)

#

#    nlabel=myfont.render("Welcome ", 1, (255, 0, 0))

#

#    for event in pygame.event.get():

#

#        if event.type==MOUSEBUTTONDOWN:

#

#            end\_it=True

#

#    screen.blit(nlabel,(200,200))

#    pygame.display.flip()

main()

    # --- Limit to 60 frames per second

# Close the window and quit.

pygame.quit()

### Errors

### Conclusion

## 

## Iteration 2

### Requirements being developed

### Errors

### Conclusion

## 

## Iteration 3

### Requirements being developed

### Errors

### Conclusion

# Testing

Using the test plan from the design show the outcome of the test and whether it passed or failed. This will probably be a table with screenshots or printouts.

# Evaluation

## Requirements

List all requirements and say whether they are met, partially met or not met. Say which test demonstrated this. You should say why they are not or are partially met.

## Stakeholders

What did the stakeholders think of the game?

## Usability

How usable is the game? What instructions are there on-screen

## Maintainability

How did you write the code to help to maintain the code? OOP .. etc.

## Future developments

Say what you think you would do next. Give at least ideas and say how you think you would implement them

# Bibliography

What site or books have you used for th project. Reserahc, help with coding etc

# Code listing

A list of the final code