# Section 2: 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

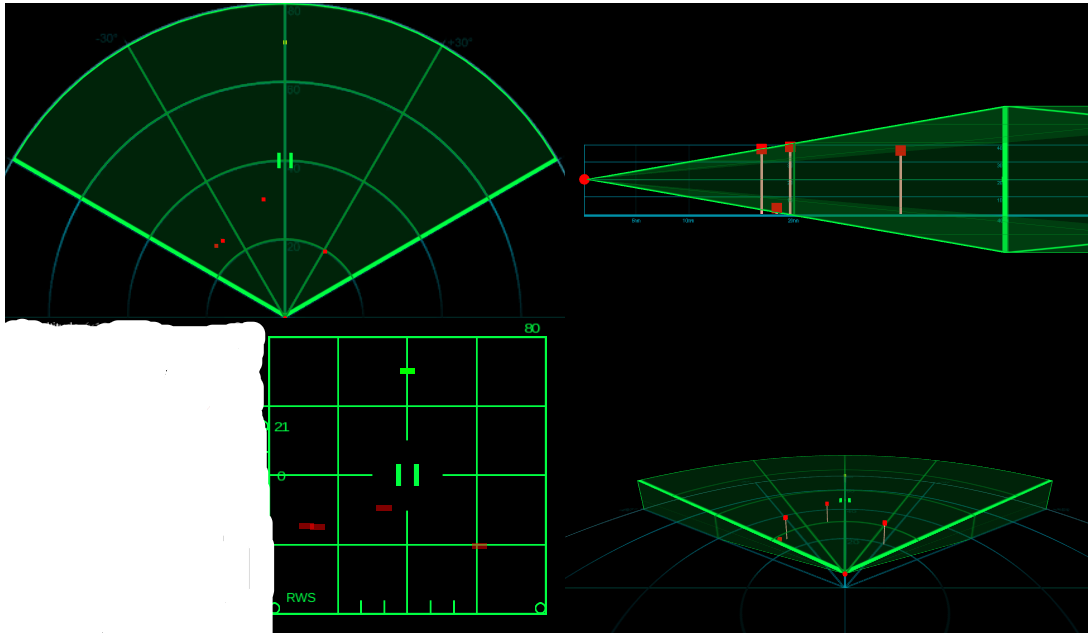
Game Over Menu:

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



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. In order to implement this I will have to utilise rotation and collision functions in pygame. The contrast between the green and black of the display is vibrant and will not tire the eye.

# Interface Design Concepts

Icon of a target that has been locked

Targets move towards the center of the radar

Gamescreen: (above).

Base

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

Radar Line that detects the targets.

The line moves clockwise, spinning around its own axis

Score Display

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

Radar Cursor

It can move in any direction

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.

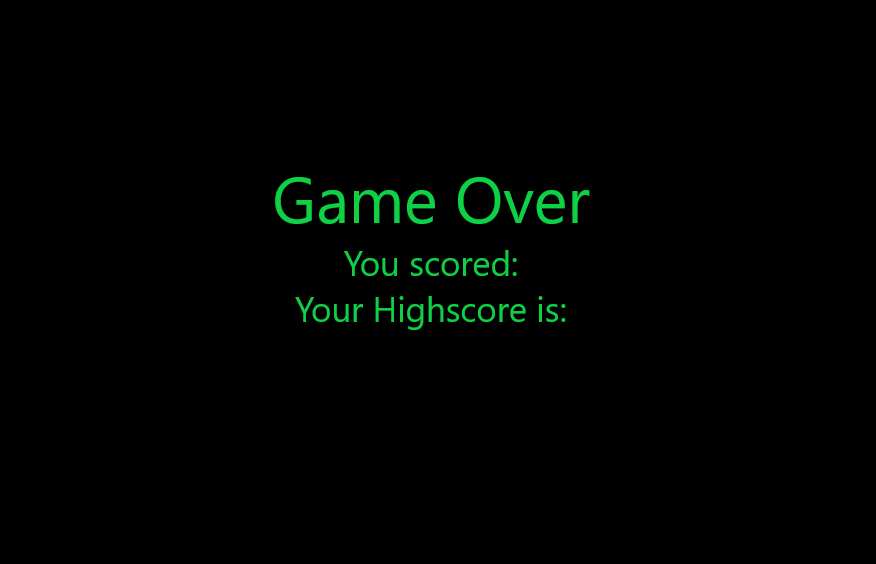
The radar cursor will be wide enough to surround the icon of an unlocked target. After locking the target, the player will be able to destroy the target. The icon of a locked target is far different from the icon of an unlocked target.

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. The size of the windows will be 1000x1000.



Game Over Screen: (above).

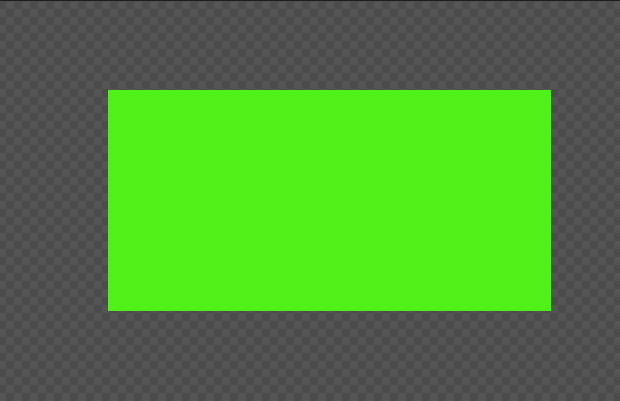
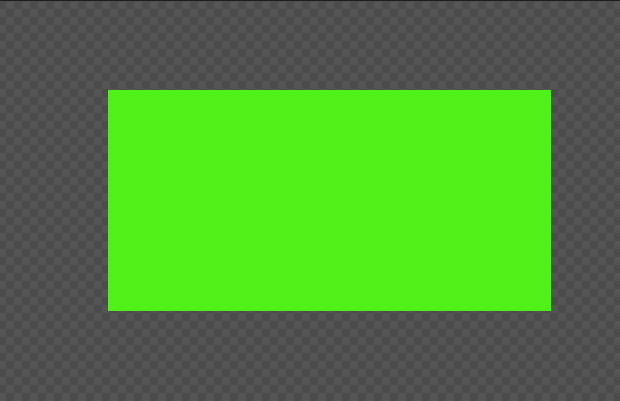
This is a quite simple game over screen. The window size is 1000x1000. It contains 3 lines of text. 1 indicated that the game is over, the other indicates your currents score. The one below displays your highest ever recorded score. This will add an element of competitiveness to the game as the player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

# Graphic 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. I will create 4 unique images. The pixel editor that I will utilise is called Piskel

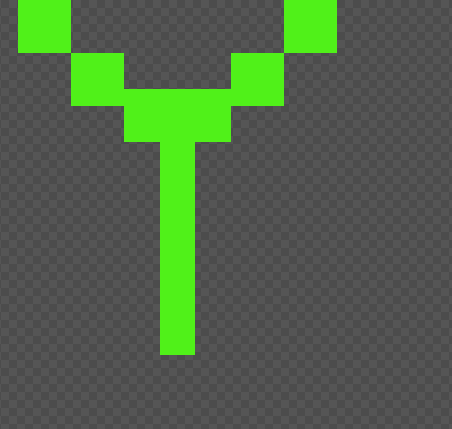
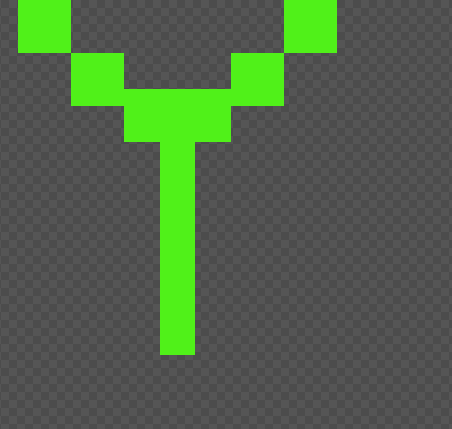


Image 1: Unlock



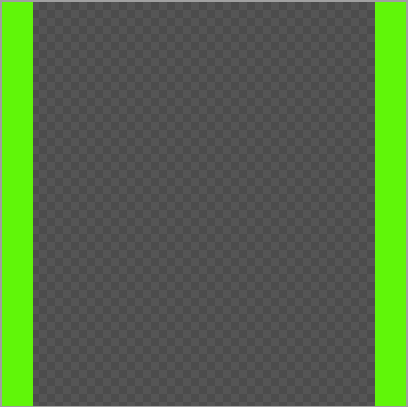
Here are the two iterations of the UNLOCKED state of the enemy sprite. One is 40x40 width, the other is 20x20 width. I ended up selecting the 20x20 iterations as I concluded that the size 40x40 would be too large

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. I shall use pygame code in order to switch between the 3 images created for the enemy in order to properly represent radar deteection.

Data Design:

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.y = y

        self.rect.x = x

        self.change\_x = 0

        self.change\_y = 0

        self.image=image

This represents the design of my cursor, in the beta stage I shall use my keyboard to control the cursor, evident from the change\_x and change\_y values. The initial x and y coordinates will be determined by myself depending on the orientation of the radar

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

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

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

        self.rect.y = y\_pos

        self.rect.x = x\_pos

    def change\_state(self, image):

        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

The speed of the enemy class will be determined by the the player as seen from the y\_d and x\_d variables. I have added masking in order for improved collisions with the rotating radar array. The change state method that I will utilise will allow the image of the Enemy to change. The update method will allow to move on itself based on increasing/decreasing x and y values set depending on the spawn location of the Enemy.

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

        for i in range(1, 10):

            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 = i\*.1

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

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

For the line I have utilised masking in order to allow the collisions to be simulated with the line istead of the 800x800 surface. This is very complex in pygame as pygame does not siumlate the roations of any sort of images. By utilising mathematical functions such as cos and sin I allow the the lines to spin around its own axis in the update function. In order to add a colour gradient I utilse the f variable which decreases the green value of every line which would add a visually pleasing effect.