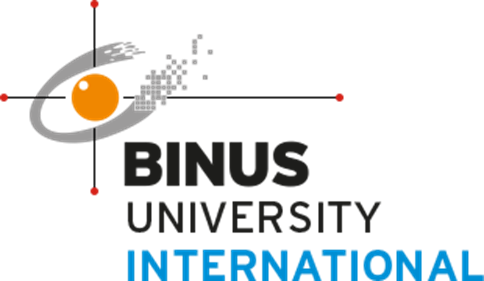
Odd Semester (2023)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Algorithm and Programming Final Project**

**Fantasy Fighters**

**Student Information:**

**Surname:** Komala **Given Name:** Ryan Patrick Komala **Student ID:** 2702334853

**Course Code :** COMP6047001 **Course Name:** Algorithm and Programming

**Class :** L1AC **Lecturer :** Jude Joseph Lamug Martinez, MCS

**Type of Assignment :** Final Project Report

**Submission Pattern**

**Due Date :** 12 January 2024 **Submission Date :** 12 January 2024

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating**

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality**

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Ryan Patrick Komala

Table of Contents

[A. Introduction 4](#_Toc155961470)

[B. Project Specification 4](#_Toc155961471)

[1. Background 4](#_Toc155961472)

[2. Gameplay 4](#_Toc155961473)

[3. Modules 5](#_Toc155961474)

[4. Game Folders and Files 5](#_Toc155961475)

[5. Assets Used 6](#_Toc155961476)

[C. Solution Design 8](#_Toc155961477)

[D. Essential Algorithms 8](#_Toc155961478)

[E. Screenshots 8](#_Toc155961479)

[ Main Menu 8](#_Toc155961480)

[ Countdown Intro 9](#_Toc155961481)

[ Player 1 Round Win 9](#_Toc155961482)

[ Player 2 Round Win 10](#_Toc155961483)

[ Game Over Screen 10](#_Toc155961484)

[F. Evaluation and Reflection 11](#_Toc155961485)

# **Introduction**

The main point of this final project is to challenge the students to go above and beyond the lessons that has been previously discussed in the Algorithm and Programming Course. Each student is expected to implement all of the topics that has been taught in class as well as research and discover new Python concepts that has not been discussed in class yet.

From the very beginning, I was interested in understanding how games are created and how the code works in creating the game logics, this is why I ended up choosing to create a game for this project. I made a game similar to “Flappy Bird” which only takes a single input, the mouse left-click, to play the game. But after further thought, I decided that the game might be too simple and generic so I ended up making a 2-player fighting game which I have named “Fantasy Fighters”. This game dives deeper into the pygame module and makes use of new functions and features of pygame that has not been discussed in class.

This project can be accessed through my GitHub repository in,  
<https://github.com/RyanPK528/Fantasy-Fighters>

# Project Specification

## Background

Fantasy Fighters is a 2-player fighting game, which means the game requires two different users to control each fighter/character with their own inputs. The theme that I chose for the game is ‘*Fantasy*’, because of this I wanted to set the background stage to something otherworldly, after some digging through multiple art online, I came across a fiery stage with tall towering structures. This stage paired with the sprites for the fighters achieves the sense of sword and sorcery in every fantasy setting that we all know and love.

## Gameplay

The gameplay is relatively simple, one person controls the fighter on the left side of the screen and another controls the fighter on the right side of the screen. The objective of the game is to beat the other player and be the first to win 3 rounds.

Player 1 can control their fighter using the following keys:

1. ‘a’ – to move to the left
2. ‘d’ – to move to the right
3. ‘w’ – to jump
4. ‘q’ – to initiate attack 1
5. ‘e’ – to initiate attack 2

Player 2 can control their fighter using the following keys:

1. ‘j’ – to move to the left
2. ‘l’ – to move to the right
3. ‘i’ – to jump
4. ‘u’ – to initiate attack 1
5. ‘o’ – to initiate attack 2

After the discussion with Sir Jude during my project presentation regarding the damage output for the different types of attack, where I previously made it so that both attacks deal the same amount of damage, I decided to make the attacks different by making it deal different amount of damage and have different cooldowns.

## Modules

* Pygame – a Python module for creating video games using Python as the main programming language. It includes functions for computer graphics, sound libraries and so on, in the case of my game, pygame was mainly used in creating the game window, handling user input, drawing images on the screen, and playing music or sound effects.

## Game Folders and Files

* *main.py* – The main file where the game is initialized and run. Functions for creating the game’s main menu, game over screen, drawing the user interface and loading the assets can be found here.
* *fighter.py* – The file containing functions for the overall logic of how the fighters work inside the game.
* *settings.py* – The file containing self-defined variables used through out the game.
* *‘assets’ folder* – The folder that contains all of the art/images, audio/sound effects, and the fonts used inside the game.
* *‘Documents’ folder* – The folder that contains the report file (pdf), screenshots of the game, video demonstration as well as the diagrams for this final project.

## Assets Used

All of the assets used in this game was not made by me nor does it belong to me.

1. Game Images/Art:
   1. Background stage

Taken from one of the background stages in Street Fighter III   
(<https://i.pinimg.com/originals/3f/3c/9a/3f3c9a96c1f9cf268707e606eb758a6f.png>)

* 1. Fighter 1 (Warrior)

Credits to LuizMelo in itch.io,

(<https://luizmelo.itch.io/martial-hero-3>)

* 1. Fighter 2 (Samurai)

Credits to LuizMelo in itch.io,

(<https://luizmelo.itch.io/martial-hero>)



1. Game Audio:
   1. Background music

Credits to sonatina in itch.io,

“Passion and Precision”

(<https://sonatina.itch.io/sibz-selection>)

* 1. Sword sound effect

Credits to Herkules92 in freesound.org,

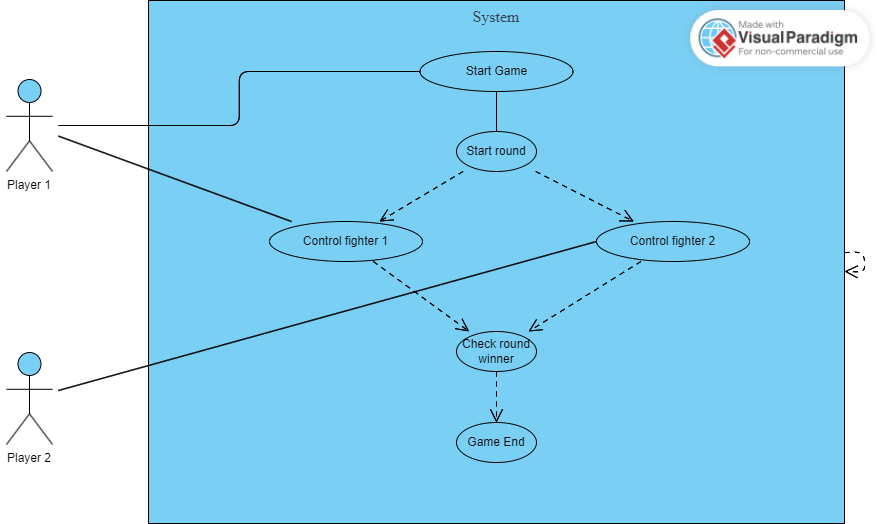
(<https://freesound.org/people/Herkules92/sounds/547600/>)

1. Game Font:
   1. Font Type

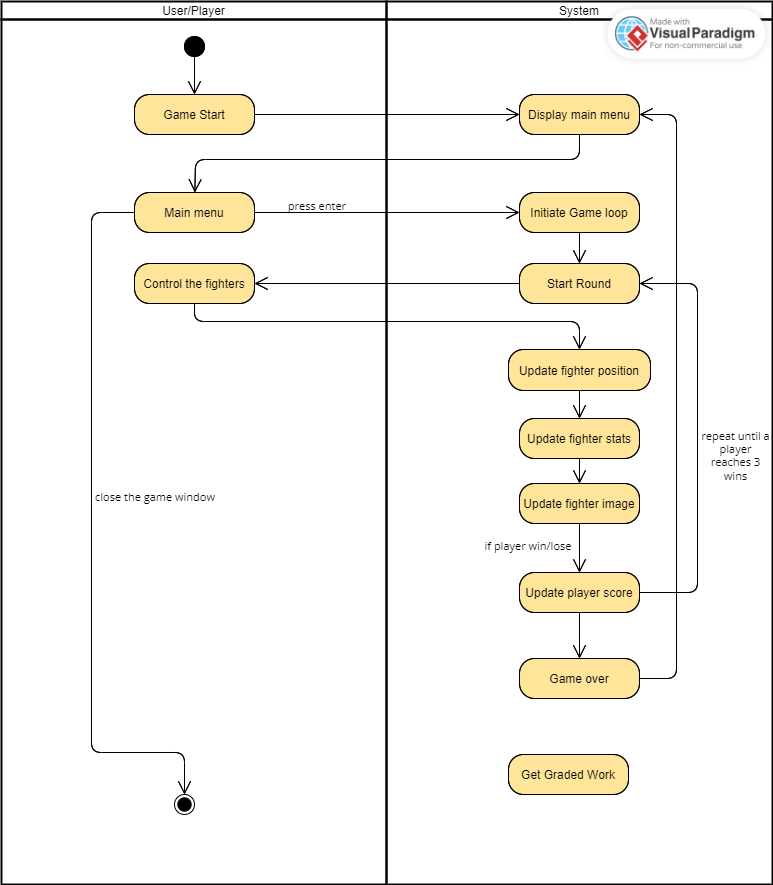
(<https://fontmeme.com/fonts/turok-font/>)

# Solution Design

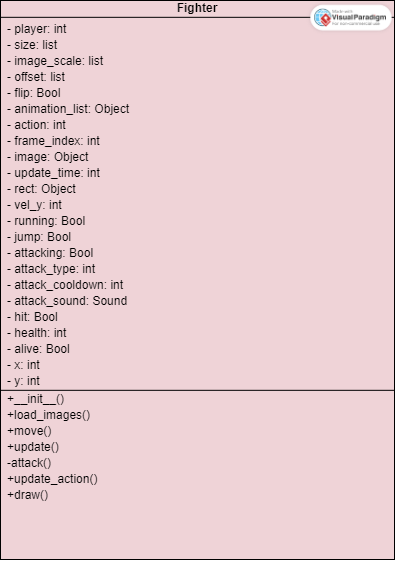
## Use Case Diagram



## Activity Diagram



## Class Diagram



# Essential Algorithms

## settings.py,

Contains all the self-defined variables used in the game

* Import pygame library, to be used in a variable involving time

import pygame

* Variables for the displayed window (size and name) as well as the game’s running frame rate

# Window settings

SCREEN\_WIDTH = 1000

SCREEN\_HEIGHT = 600

GAME\_NAME = "Fantasy Fighters"

# FPS

FPS = 60

* Variables for RGB values of different colors

# Color settings

RED = (255, 0, 0)

YELLOW = (255, 255, 0)

GREEN = (0, 255, 0)

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

* Variables for the game functions, such as the movement values, cooldown or countdown timers

# Game function variables

SPEED = 10 # movement value (move by 10px)

GRAVITY = 2 # movement value (move down by 2px)

intro\_count = 4 # 4s countdown before each round

last\_count\_update = pygame.time.get\_ticks()

score = [0, 0] # [P1, P2]

round\_over = False

round\_over\_cooldown = 2000 # 2s

winner = 0 # Player who won

wins\_required = 3

* Variables for the fighter model or art,

# Fighter variables

WARRIOR\_SIZE = 126

WARRIOR\_SCALE = 3

WARRIOR\_OFFSET = [60, 20]

WARRIOR\_DATA = [WARRIOR\_SIZE, WARRIOR\_SCALE, WARRIOR\_OFFSET]

SAMURAI\_SIZE = 200

SAMURAI\_SCALE = 3

SAMURAI\_OFFSET = [80, 60]

SAMURAI\_DATA = [SAMURAI\_SIZE, SAMURAI\_SCALE, SAMURAI\_OFFSET]

# Number of frames in each animation

WARRIOR\_ANIMATION\_STEPS = [10, 8, 3, 7, 9, 3, 11]

SAMURAI\_ANIMATION\_STEPS = [8, 8, 2, 6, 6, 4, 6]

## fighters.py,

The file containing functions for the overall logic of how the fighters work inside the game.

* Import modules and values from another file

# Modules

import pygame.sprite

from settings import \*

* Fighter class with the self-variables

class Fighter(pygame.sprite.Sprite):

    def \_\_init\_\_(self, player, x, y, flip, data, sprite\_sheet, animation\_steps, sound):

        super().\_\_init\_\_()

        self.player = player # Player 1 (left) or 2 (right)

        self.size = data[0] # Sprite size

        self.image\_scale = data[1] # Scaling

        self.offset = data[2] # Image offset to get the right position

        self.flip = flip # Flip iamge to make sure fighters face each other

        self.animation\_list = self.load\_images(sprite\_sheet, animation\_steps) # Animation from spritesheet

        self.action = 0  # 0: idle #1: run #2: jump #3: attack1 #4: attack2 #5: hit #6: death

        self.frame\_index = 0 # Animation frame

        self.image = self.animation\_list[self.action][self.frame\_index] # Image displayed based on animation frame and action

        self.update\_time = pygame.time.get\_ticks()

        self.rect = pygame.Rect((x, y, 80, 180)) # Hitbox

        self.vel\_y = 0 # movement value in y axis

        self.running = False

        self.jump = False

        self.attacking = False

        self.attack\_type = 0

        self.attack\_cooldown = 0

        self.attack\_sound = sound

        self.hit = False

        self.health = 100

        self.alive = True

* Function to separate the fighter’s spritesheet into each singular image frame for the different state of action. The spritesheet contains all of the fighter’s images, the first row for idle action, second row for running action, third for jumping, fourth for attack 1, fifth for attack 2, sixth for taking hit and seventh for death. This code will go through each row of the spritesheet and with the animation\_steps, it will go through the available frames for each action.

def load\_images(self, sprite\_sheet, animation\_steps):

        # Get seperate images from spritesheet

        animation\_list = []

        for y, animation in enumerate(animation\_steps):

            temp\_img\_list = []

            for x in range(animation):

                temp\_img = sprite\_sheet.subsurface(x \* self.size, y \* self.size, self.size, self.size)

                temp\_img\_list.append(pygame.transform.scale(temp\_img, (self.size \* self.image\_scale, self.size \* self.image\_scale)))

            animation\_list.append(temp\_img\_list)

        return animation\_list

* Function to allow movement and actions of the fighters

def move(self, screen\_width, screen\_height, surface, target, round\_over):

        dx = 0

        dy = 0

        self.running = False

        self.attack\_type = 0

        # Get keypresses

        key = pygame.key.get\_pressed()

* (Inside the move function)

Take inputs from the players and set the state of action to True. For horizontal movement, it will cause a change in x by the SPEED. For jumping, it will set a negative y velocity to make the fighter move upwards by 30px. Then based on the input of attack, it will set the value of attack\_type to either 1 or 2 and initiate the attack function towards the enemy.

# Allow other actions if not attacking

        if self.attacking == False and self.alive == True and round\_over == False:

            # Player 1 controls

            if self.player == 1:

                # Move

                if key[pygame.K\_a]:

                    dx = -SPEED

                    self.running = True

                if key[pygame.K\_d]:

                    dx = SPEED

                    self.running = True

                # Jump

                if key[pygame.K\_w] and self.jump == False:

                    self.vel\_y = -30

                    self.jump = True

                # Attack

                if key[pygame.K\_q] or key[pygame.K\_e]:

                    # Attack type

                    if key[pygame.K\_q]:

                        self.attack\_type = 1

                        self.\_\_attack(target)

                    if key[pygame.K\_e]:

                        self.attack\_type = 2

                        self.\_\_attack(target)

* (Inside the move function)

Set a constant downward movement value for the fighter by GRAVITY

# Gravity

self.vel\_y += GRAVITY # Constant downward position change

dy += self.vel\_y

* (Inside the move function)

Made sure the fighters don’t go beyond the screen by limiting the fighter’s x values to not go below 0 or beyond the screen width. For the y position, the minimum y value is 70 so it stays on top of the ‘floor’.

 # Limit player position within screen

 if self.rect.left + dx < 0:

    dx = -self.rect.left

 if self.rect.right + dx > screen\_width:

     dx = screen\_width - self.rect.right

 if self.rect.bottom + dy > screen\_height - 70:

     self.vel\_y = 0

    self.jump = False

     dy = screen\_height - 70 - self.rect.bottom

* (Inside the move function)

Made sure the players face each other by flipping their image based on their relative position to each other.

# Ensure players face each other

if target.rect.centerx > self.rect.centerx:

  self.flip = False

else:

  self.flip = True

* (Inside the move function)

Countdown for the attack cooldown

# Attack cooldown

if self.attack\_cooldown > 0:

  self.attack\_cooldown -= 1

* (Inside the move function)

Update the fighter’s position based on the change made with the dx or dy value.

# Update player position

self.rect.x += dx

self.rect.y += dy

* Function to update

# Animation updates

    def update(self):

        if self.health <= 0:

            self.health = 0

            self.alive = False

            self.update\_action(6)  # 6: Death

        elif self.hit:

            self.update\_action(5)  # 5: Hit

        elif self.attacking:

            if self.attack\_type == 1:

                self.update\_action(3)  # 3: Attack1

            elif self.attack\_type == 2:

                self.update\_action(4)  # 4: Attack2

        elif self.jump:

            self.update\_action(2)  # 2: Jump

        elif self.running:

            self.update\_action(1)  # 1: Run

        else:

            self.update\_action(0)  # 0: Idle

        animation\_cooldown = 60 # duration of each frame

* (Inside the update function)

Update and change the image of the fighter to go through the entire animation sequence. In certain cases, an action’s animation has more priority, if a fighter is dead, it will instantly start the death animation despite being in the middle of another animation and stop the animation. Resets the action values to False after playing the animation (attacking and hit) then add the cooldown.

# Update fighter image

        self.image = self.animation\_list[self.action][self.frame\_index]

        # Change to next frame after cooldown

        if pygame.time.get\_ticks() - self.update\_time > animation\_cooldown:

            self.frame\_index += 1

            self.update\_time = pygame.time.get\_ticks()

        # Check if the animation sequence has finished

        if self.frame\_index >= len(self.animation\_list[self.action]):

            # If fighter is dead then end the animation

            if not self.alive:

                self.frame\_index = len(self.animation\_list[self.action]) - 1

            else:

                self.frame\_index = 0

                # Check if an attack was executed

                if self.action == 3:

                    self.attacking = False

                    self.attack\_cooldown = 20

                elif self.action == 4:

                    self.attacking = False

                    self.attack\_cooldown = 45

                # Check if damage was taken

                if self.action == 5:

                    self.hit = False

                    # If the player was in the middle of an attack, then the attack is stopped

                    self.attacking = False

                    self.attack\_cooldown = 30

* Function for the fighter’s attack, it will only work if there is no attack cooldown, as this function is initiated, it will set the attacking to True so the action can be updated and it will play the sound effect. There is an attack rectangle that will expand towards the enemy and if it reaches and collides with the target’s rectangle, it will consider it as a hit and damage will be done. Attack 1 deals 10 health damage but cooldown is faster than attack 2, but attack 2 deals twice the damage but also has more than twice the cooldown.

def \_\_attack(self, target):

        if self.attack\_cooldown == 0:

            # Execute an attack

            self.attacking = True

            self.attack\_sound.play()

            attacking\_rect = pygame.Rect(self.rect.centerx - (2 \* self.rect.width \* self.flip), self.rect.y, 2 \* self.rect.width, self.rect.height)

            if self.attack\_type == 1:

                if attacking\_rect.colliderect(target.rect):

                    target.health -= 10

                    target.hit = True

            elif self.attack\_type == 2:

                if attacking\_rect.colliderect(target.rect):

                    target.health -= 20

                    target.hit = True

* Function to update the fighter’s action in the case of an action change

def update\_action(self, new\_action):

        # Check if the new action is different from the previous one

        if new\_action != self.action:

            self.action = new\_action

            # Update the animation settings

            self.frame\_index = 0

            self.update\_time = pygame.time.get\_ticks()

* Function to draw the fighter’s image

def draw(self, surface):

        img = pygame.transform.flip(self.image, self.flip, False)

        surface.blit(img, (self.rect.x - (self.offset[0] \* self.image\_scale), self.rect.y - (self.offset[1] \* self.image\_scale)))

## main.py,

The main file where the game is initialized and run. Contains functions for creating the game’s main menu, game over screen, drawing the user interface and loading the assets can be found here.

* Import modules and class from other files and initializing them

# Modules used for the game

import pygame

from pygame import mixer

from settings import \*

from fighter import Fighter

mixer.init()

pygame.init()

* Function to initialize the game window

# Function to initialize the game

def initialize\_game():

    # Window settings

    screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

    pygame.display.set\_caption(GAME\_NAME)

    clock = pygame.time.Clock()

    return screen, clock

* Function to load all of the game assets and return them as a variable

# Function to load the game assets

def load\_assets():

    # In-game audio

    pygame.mixer.music.load("assets/audio/music.wav")

    pygame.mixer.music.set\_volume(0.4)

    pygame.mixer.music.play(-1, 0.0, 5000)

    sword\_fx = pygame.mixer.Sound("assets/audio/sword.wav")

    sword\_fx.set\_volume(0.45)

    # Background image

    bg\_image = pygame.image.load("assets/images/background/background\_stage.png").convert\_alpha()

    # Load fighter spritesheets

    warrior\_sheet = pygame.image.load("assets/images/warrior/Sprites/warrior.png").convert\_alpha()

    samurai\_sheet = pygame.image.load("assets/images/samurai/Sprites/samurai.png").convert\_alpha()

    return bg\_image, warrior\_sheet, samurai\_sheet, sword\_fx

* Function to wait for a key input

# Function to wait for a key press

def wait\_for\_key():

    waiting = True

    while waiting:

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

            if event.type == pygame.KEYDOWN:

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

                    waiting = False

* Function to display the main menu

# Function to display the main menu

def main\_menu():

    screen.fill(BLACK)

    draw\_text("Press ENTER to start", count\_font, WHITE, 120, 250)

    pygame.display.update()

    wait\_for\_key() # Start the game after pressing 'ENTER'

* Function to display the game over screen

# Function to display the game-over screen

def game\_over():

    screen.fill(BLACK)

    draw\_text("GAME OVER", count\_font, WHITE, 325, 250)

    pygame.display.update()

    pygame.time.delay(2000)  # Display for 2 seconds

    main\_menu()  # Go back to the main menu

* Function to draw text anywhere in the screen depending on the x and y values

# Function to draw text in the game

def draw\_text(text, font, color, x, y):

    img = font.render(text, True, color)

    screen.blit(img, (x, y))

* Function to draw the background stage

# Function to draw background

def draw\_bg():

    scaled\_bg = pygame.transform.scale(bg\_image, (SCREEN\_WIDTH, SCREEN\_HEIGHT)) # Adjust image proportions to fit in window

    screen.blit(scaled\_bg, (0, 0))

* Function to draw the health bar for each fighter

# Function to draw fighter's health bars

def draw\_health\_bar(health, x, y):

    ratio = health / 100

    pygame.draw.rect(screen, BLACK, (x - 2, y - 2, 408, 38)) # Shadow border

    pygame.draw.rect(screen, RED, (x, y, 400, 30)) # Health loss

    pygame.draw.rect(screen, GREEN, (x, y, 400 \* ratio, 30)) # Health left

* Pre-loop initialization

# Initialize game

screen, clock = initialize\_game()

# Load assets

bg\_image, warrior\_sheet, samurai\_sheet, sword\_fx = load\_assets()

# Set text font

count\_font = pygame.font.Font("assets/fonts/turok.ttf", 80)

score\_font = pygame.font.Font("assets/fonts/turok.ttf", 30)

* Initialize the main menu loop

# Main menu loop

main\_menu()

* Create the fighters, set the values for the variables in their functions and set their position on the screen

# Create two instances of fighters

fighter\_1 = Fighter(1, 200, 350, False, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx)

fighter\_2 = Fighter(2, 700, 350, True, SAMURAI\_DATA, samurai\_sheet, SAMURAI\_ANIMATION\_STEPS, sword\_fx)

* The start of the main game loop, setting the FPS, and drawing the heads-up display (HUD) and background

# Game loop

run = True

while run:

    clock.tick(FPS)

    # Draw background

    draw\_bg()

    # Show player stats

    draw\_health\_bar(fighter\_1.health, 20, 20)

    draw\_health\_bar(fighter\_2.health, 580, 20)

    draw\_text("Player 1: " + str(score[0]), score\_font, WHITE, 20, 60)

    draw\_text("Player 2: " + str(score[1]), score\_font, WHITE, 580, 60)

* The initial round countdown, the intro\_count will go down from 4 to 0 every second and this countdown will be displayed with the draw\_text function. In this countdown state, the fighters are unable to move or act, the fighter will only be able to move when the intro\_count is finished or has the value 0, in this case the move function is initialized and the players can move the fighters.

# Update countdown

    if intro\_count <= 0:

        # Move fighters

        fighter\_1.move(SCREEN\_WIDTH, SCREEN\_HEIGHT, screen, fighter\_2, round\_over)

        fighter\_2.move(SCREEN\_WIDTH, SCREEN\_HEIGHT, screen, fighter\_1, round\_over)

    else:

        # Display count timer

        draw\_text(str(intro\_count), count\_font, YELLOW, 480, 230)

        # Update count timer

        if (pygame.time.get\_ticks() - last\_count\_update) >= 1000:

            intro\_count -= 1

            last\_count\_update = pygame.time.get\_ticks()

* Update the fighter’s stats and draw the proper image based on the fighter’s action

# Update fighters

fighter\_1.update()

fighter\_2.update()

# Draw fighters

fighter\_1.draw(screen)

fighter\_2.draw(screen)

* Check if a player loses, updates the score, end the round, draw the victory image, then resets all the player stats for the next round.

# Check for player defeat

    if round\_over == False:

        if fighter\_1.alive == False:

            score[1] += 1

            winner += 2

            round\_over = True

            round\_over\_time = pygame.time.get\_ticks()

        elif fighter\_2.alive == False:

            score[0] += 1

            winner += 1

            round\_over = True

            round\_over\_time = pygame.time.get\_ticks()

    else:

        # Display victory image

        draw\_text(f"PLAYER {winner} WINS", count\_font, YELLOW, 280, 200)

        if pygame.time.get\_ticks() - round\_over\_time > round\_over\_cooldown:

            round\_over = False

            winner = 0

            intro\_count = 4

            fighter\_1 = Fighter(1, 200, 350, False, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx)

            fighter\_2 = Fighter(2, 700, 350, True, SAMURAI\_DATA, samurai\_sheet, SAMURAI\_ANIMATION\_STEPS, sword\_fx)

* Check if a player reaches 3 wins or has a score of 3, if a player has 3 wins, considers the game to be finished and display the game over screen with the game\_over function and reset the fighter stats.

# Check for game over

    if score[0] >= wins\_required and round\_over == False or score[1] >= wins\_required and round\_over == False:

        # Display game over screen

        game\_over()

        # Reset scores and fighters

        score = [0, 0]

        fighter\_1 = Fighter(1, 200, 350, False, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx)

        fighter\_2 = Fighter(2, 700, 350, True, SAMURAI\_DATA, samurai\_sheet, SAMURAI\_ANIMATION\_STEPS, sword\_fx)

    # Event handler

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            run = False

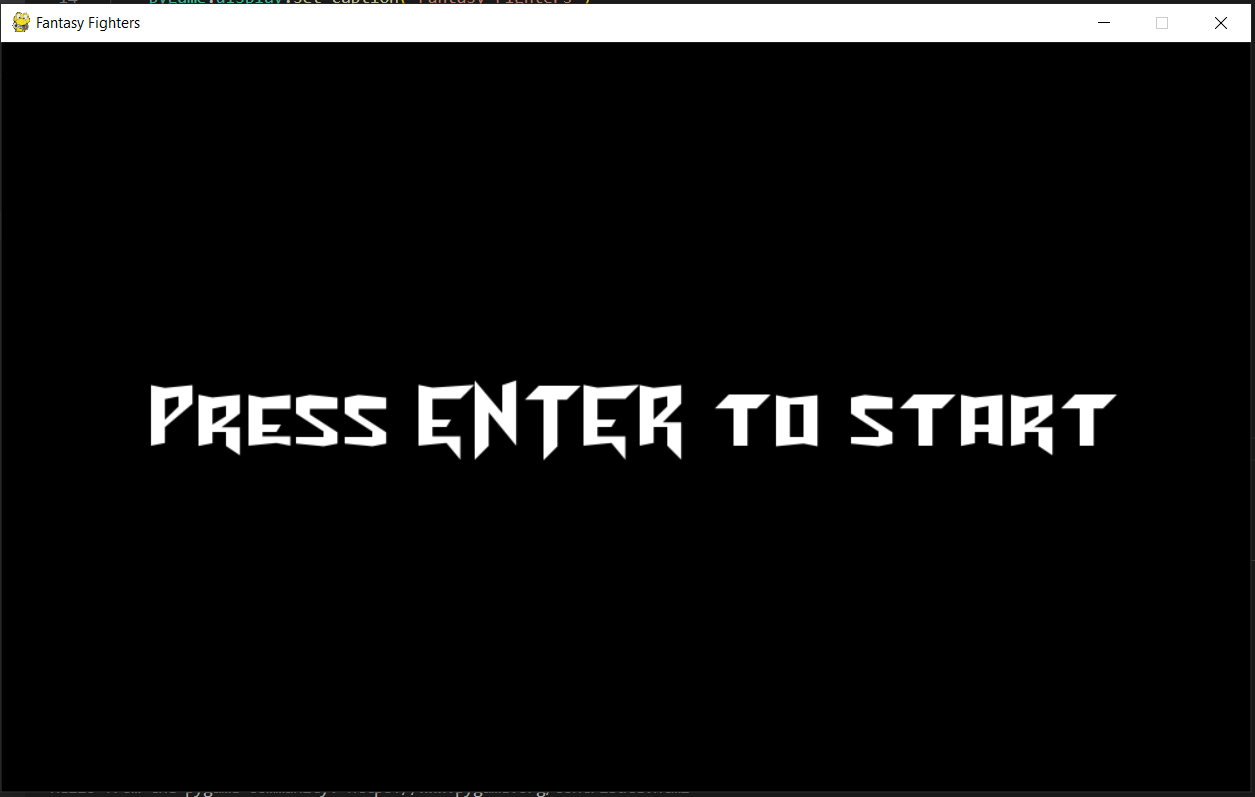
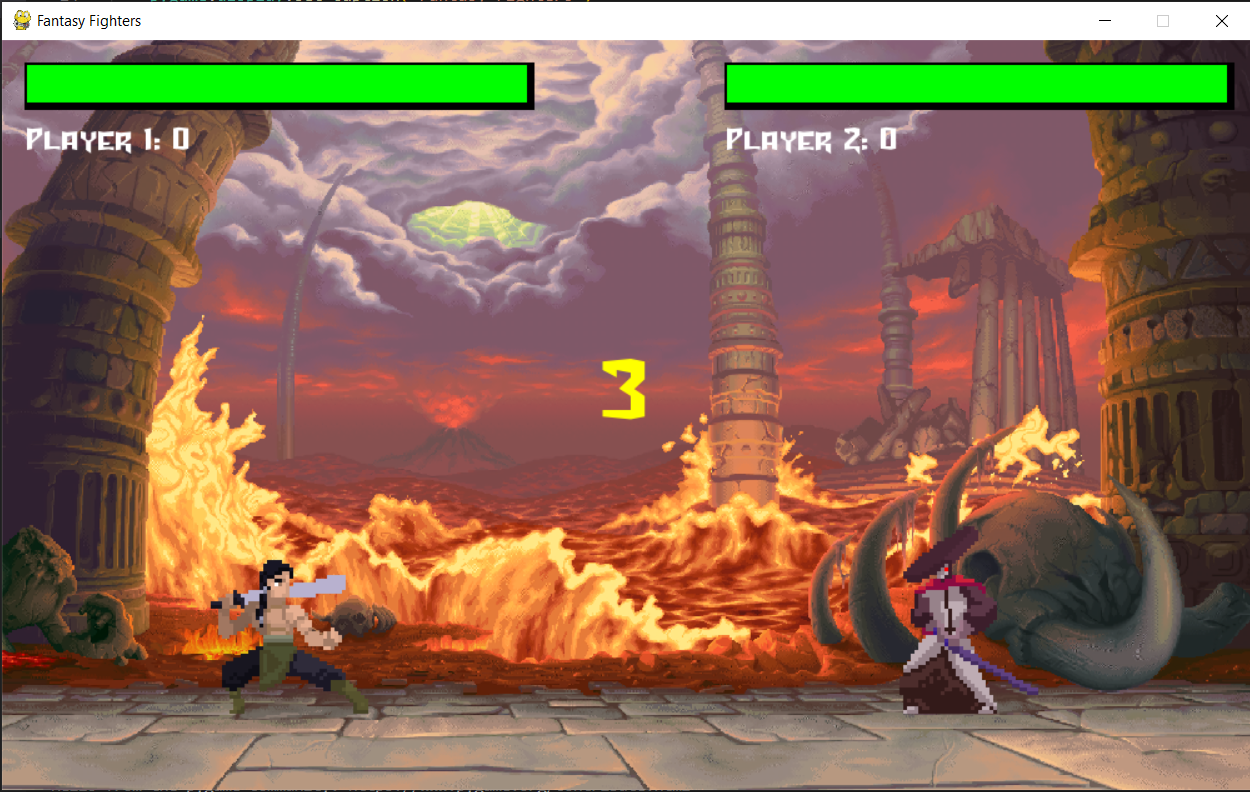
    # Update display

    pygame.display.update()

# Exit pygame

pygame.quit()

# Screenshots

* Main Menu  
  
* Countdown Intro  
  
* Player 1 Round Win  
  
* Player 2 Round Win  
  

## Game Over Screen



# Evaluation and Reflection

I had mixed emotions throughout the whole process of working on this final project and I learned many new things, specifically about game development and python as a programming language. There were times when I was stressed out when dealing with certain problems or issues within my written code that doesn’t seem to work the way I wanted it to be or when I had no idea how to implement a game feature that I had in mind into the code using the python functions that I knew already, but despite all of this, nothing can beat the feeling of success and excitement when I was finally able to find the solution to my problems.

There are a lot of things that I could have improved while working on this project, one of them would be better planning and time management. Initially, I planned on working on a top-down style RPG game with a unique map that I was going to make myself using the ‘Tiled’ program, but due to my poor time management and planning, I was never able to finish the map and progress with creating the game. So, I decided that it would be better for me to make a different type of game that wouldn’t really leave me stumped on a creative roadblock, so I made two more games, “Flappy Bird” and this game “Fantasy Fighters”.

The only thing that I wanted to improve on my game was to add a dodge or block mechanic where the fighter will be able to either reduce or completely negate the incoming damage, I had multiple attempts in trying to implement this feature, but it always ends up not working. Maybe if I started working on this project from the very beginning, I could have figured out a way to change the game’s logic to be able to adapt this feature. Other than this, I am very satisfied with how the game turned out, considering that this was my first ever “big” python project. I will strive to improve my skills in coding and have better planning so that I will be able to create more interesting games and programs.