Fighting Wario

Computer Science Project

By Zoirjon Imomaliev

Name: Zoirjon Imomaliev

Centre Number:10816

Candidate Number:

Contents

1 Analysis **6**

1.1 Problem Analysis 6

1.2 Computational Suitability 10

1.3 Research 11

1.4 Stakeholders 13

2 Interviews **14**

2.1 Summary of Interview 1 14

3 Requirements **15**

3.1 Hardware Requirements 19

3.2 Software Requirements 19

4 Design **21**

4.1 Interface 21

4.2 Attributes and Methods 22

4.3 Functionality 28

4.3.1 Defining the Player class 28

4.3.2 Defining the Tile class 30

4.3.3 Defining the Button class 31

4.3.4 Change the x and y position of objects 32

4.3.5 Get function for position 33

4.3.6 Change players health attribute 33

4.3.7 Get velocity of the player 34

4.3.8 Set velocity of the player 34

4.3.9 Check for a collision 34

4.3.10 Generate items 34

4.3.11 Import animations 35

4.3.12 Animate the player sprite 36

4.3.13 Update player gravity 38

4.3.14 Get players state of movement 38

4.3.15 Change players state when it jumps 39

4.3.16 Import all images from a folder 39

4.3.17 Draw all sprites and objects 40

4.3.18 Define a map 41

4.3.19 Main loop for the game 42

4.4 Test Plan 44

5 Implementation **45**

5.1 Iteration 1 45

5.1.1 Main Loop 45

5.1.2 Class Player 48

5.1.3 Requirements being developed 50

5.1.4 Errors 50

5.1.5 Conclusion 50

5.2 Iteration 2 50

5.2.1 Requirements being developed 50

5.2.2 Errors 50

5.2.3 Conclusion 50

5.3 Iteration 3 50

5.3.1 Requirements being developed 50

5.3.2 Errors 50

5.3.3 Conclusion 50

6 Testing **50**

7 Evaluation **50**

1 Analysis

1.1 Problem Analysis

Due to the rapid development and expansion in the technological sector in the past decade, we are presented with unlimited amount of opportunities. Operations that we were deemed 'impossible' few decades ago, such as 'Video Calling', are now a norm in our day and age. The development of the technological field has also made it impossible for a human being to live a day-to-day life without an online presence which requires a smart phone, laptop or some kind of technical device. This online presence can take place in many social media applications; Most popular being 'Facebook', 'Instagram' and 'Whats App'.

Although there are numerous positive consequences that are results of our constant online presence and dependence on the technology, like 'working from home', there are also some negative consequences. One of the

biggest negatives being the stress that social media exerts onto its users. Even though stress can be helpful, constant supply of it could be detrimental for our mental health, especially for the younger children who are now constantly on social media from a very early age.

What can we do to cope with stress? One of the ways billions of people do so, is through video games. Playing video games is a great option in order to relax and release some stress after long, mundane and arduous day of work. Due to the high demand, there are thousands of games to choose from; However most favoured ones, like 'Fortnite', 'Fifa', or 'League of Legends', are usually very hardware and software demanding. This is due to the high level of competition in the gaming sphere, as games try to become increasingly realistic.

A lot of my friends back home will love to game but they do not own a

device that will allow them to comfortably play those high-graphics game. There is a large number of low-level graphics games that are very entertaining and enjoyable. Some of those games are 'Fireboy and Watergirl', 'Pacman' and 'Mario'. I was inspired by those types of games, with simple, yet fun game-play that does not require the user to spend tens of hours on the game, just to be able to play it. Hence why, I made my game very simple, only asking the player to move the player in four basic directions, and avoid notorious enemies on their path.

The game itself is relatively simple, yet still amusing. It consists of a platform, where a player has simple options to control their character, in order to move it in the basic four directions, while avoiding getting attacked by the vicious enemies that are hunting the player or a player can also shoot those enemies in order to eliminate them. The player will get points depending on their progress into the game. During their progress, players can further col-

lect different items that help boost different attributes for a period of time, such as health or speed.

However, the main problem with platform games is that few of those games are very overused, such as 'Mario'. In order to stay creative, I carried out an extensive research in order to find older platform games ,that were seen as 'legendary' at that time, and bring them back to our modern society and try to give people a new taste of the platform games.

1.2 Computational Suitability

Many of the modern games are made by translating real life games into code and give the players ability to be able to keep playing their most loved games inside their computers. However, many other games, including my game, are originally made for virtual experience only. It is very challenging to have layers of platforms, that people can jump up and down from while eliminating their enemies and collecting items that boost their attributes and most importantly losing their lives when hit.

The execution of that concept is tailor made for a computer. It is easily playable, does not take lots of ram and does not require heavy computational power. It is really efficient, as you can compete with your friends to see who is able to achieve a higher score, and establish their superiority and most importantly the lives of players is not at risk at any point during the game. Furthermore, usage of a computer to execute this game allows developers to

add more view capturing and captivating visual effects of characters, grass and many more real life objects. Finally, the game being easily run on most machines allows millions if not billions around the world to easily download and play this game, without leaving their house, which brings joy to all communities and people around the world, without additional costs embedded.

1.3 Research

There are millions of different platform based games available, one of the most recognisable ones being 'Mario' and 'Subway Surf'. Mario allows the player to control their character in a 2D, multi-layer environment in order to jump and avoid the enemies while collecting some sort of points. However, 'Subway Surf' is a game, where player is in a 3D, dynamic environment constantly faced with moving obstacles , like trains, coming directly at the player, forcing them to react quickly, tactically and efficiently.

There is a lot that I can learn from these games. Firstly, the games can have a basic goal for the player and still manage to provide highly entertaining experience. For example, in 'Mario' it is as simple as jump over enemies and collect coins, no extra complications or high skill maneuvers required to pass levels, hence it allows more people to start playing the game without the feeling of large lack of skill, which ultimately disengages the new users from play that game again. Secondly, I can see how a large problem such as a game, can be broken down into smaller problems, hence abstracted. This will make it easier for me to build and test individual parts of the game, which will increase the my efficiency while coding and debugging.

My solution differs from these games due to the difference in the abilities of the character and the computer requirements. In my game, the user can also shoot the enemies that are on there to attempt to take the lives of the player and ultimately stop them form progressing. Furthermore, my

game will be much smaller than 'Subway Surf' and 'Mario' and will require much less computational power in order to have an optimal and enjoyable experience while playing the game.

1.4 Stakeholders

Billions of young children including myself, did not have a fortunate upbringing where our financial situation would allow us to spend hundreds of pounds on expensive computationally powerful machines in order to play all the modern games. This led to me and my friends sticking to playing simple games that were not heavy on graphics and processing, yet still provide equal or more fun.

This was a major factor that pushed towards making a simple game, in order for billions of people around to world to be able to enjoy gaming experience without spending a large amount of money. Therefore, my main

stakeholders are the children around my neighbourhood, who will greatly appreciate the game that will allow them to gain new experience with a slight competition amongst each other.

2 Interviews

2.1 Summary of Interview 1

My friends are passionate fans of 2D platform games which do not require a powerful computing unit. They also found games like 'Mario' most interesting, due the competitive environment against enemies or having a clear goal. This is what inspired me to design and plan my game to attempt to satisfy these requirements.

3 Requirements

Must Have :

1. Have a fixed screen resolution(1200x704), maintains equal size on all devices
2. Have a fixed FPS(60), ensures the game is run smoothly on all devices
3. Player can exit the game
4. Player sprite is displayed on the screen
5. Enemy sprite is displayed on the screen
6. Player has the ability to move Left, Right, Up and Down
7. Player has the ability to fire bullets
8. Player sprite cannot go outside the fixed resolution
9. Enemy sprite cannot go outside the fixed resolution
10. Player sprite can collide with bullets

11. Enemy sprite can collide with bullets

Should Have :

1. Player health is displayed on the screen
2. Enemy health is displayed on the screen
3. Enemy sprite has a complex algorithm to dodge Player bullets
4. Enemy sprite has a complex algorithm to fire bullets at the Player
5. Elementary graphics, ensures the game is run smoothly
6. Health aid is randomly spawned on each side
7. Sound effects present when the Player shoots
8. Sound effects present when the Enemy shoots
9. Sound effects present when the Player gets hit
10. Sound effects present when the Enemy gets hit
11. Player can lose lives
12. Enemy can lose lives

13. Player can gain health by collecting the health aid

14. Enemy can gain health by collecting the health aid

Could Have :

1. Have a trained model that opposes the player

2. Different levels, with ascending difficulty

3. Player can adjust the difficulty level

4. In-game background, to make the gaming experience more engaging

5. Allow the player to save the progress

6. Have a pause option

7. Advanced level graphics, to make the gaming experience more amusing

8. Player can change the appearance of the spaceship

9. Tutorial, to give a brief introduction to the game

10. Full screen mode, to fully immerse the player into the game

11. Have a multiplayer mode. to play against a real person

Won't Have :

1. Online multiplayer mode, to play against a real opponent online

2. 3D interface, to make the graphics look more realistic

3. Player movement controlled via an external device, like a controller

4. Implementation of an AI, in order to analyse player's moves, to have more accurate aim

5. Mods, to allow the player to customise and add their own features to the game

6. Settings option, to change the FPS of the game

3.1 Hardware Requirements

- Minimum CPU : i3 or above
- Minimum Clock Speed : 1GHz or more
- Minimum HDD : 50GB or more
- Minimum RAM : at least 1GB or higher
- Input : Keyboard
- Output : Screen

3.2 Software Requirements

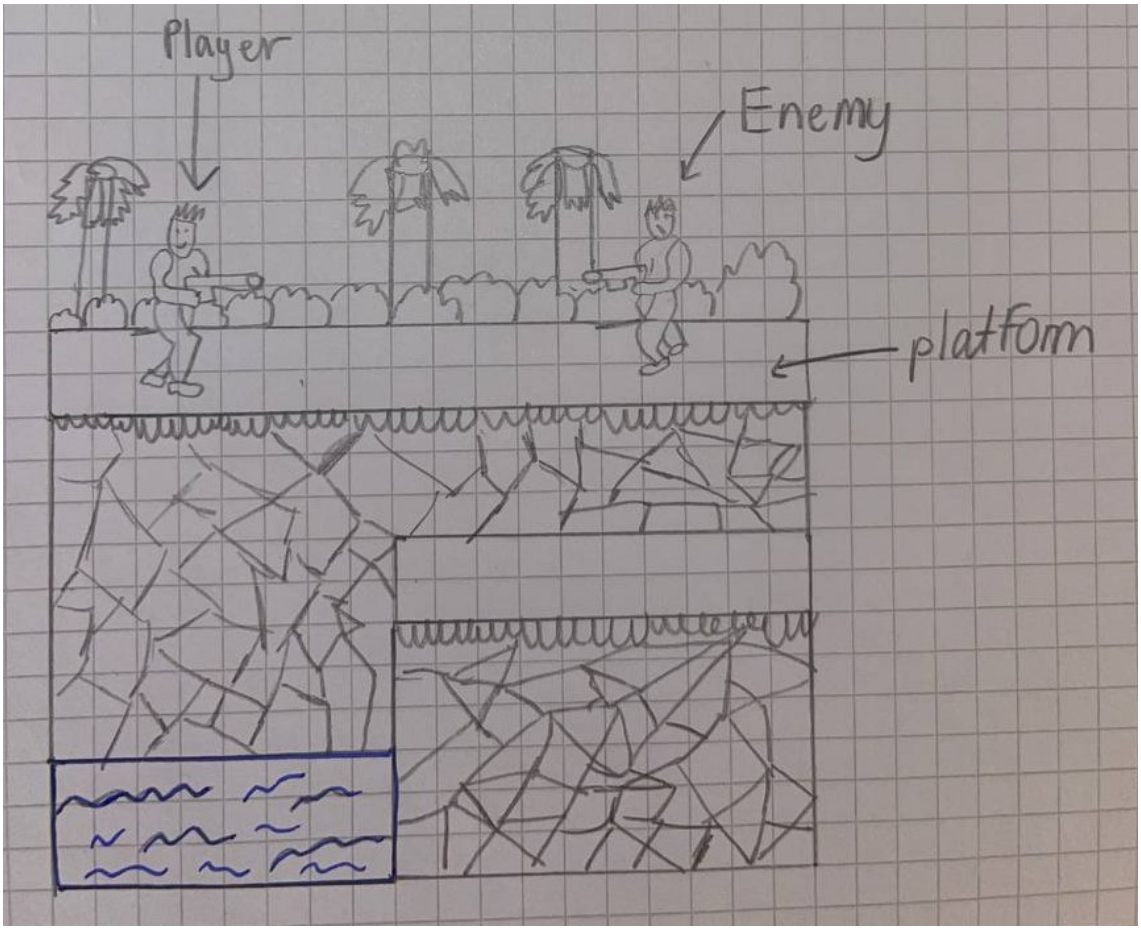
- OS : Windows, Mac-OS or Linux
- IDE : e.g Visual Studio, PyCharm and etc.
- python installed
- python libraries : pygame, os, math, json, random

- access to the internet

4 Design

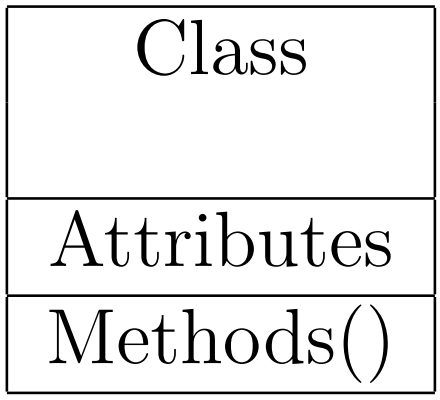
4.1 Interface

Hand drawn diagram to display the appearance of the game to the player



4.2 Attributes and Methods

I used the following structure to display my Classes, Attributes and Methods



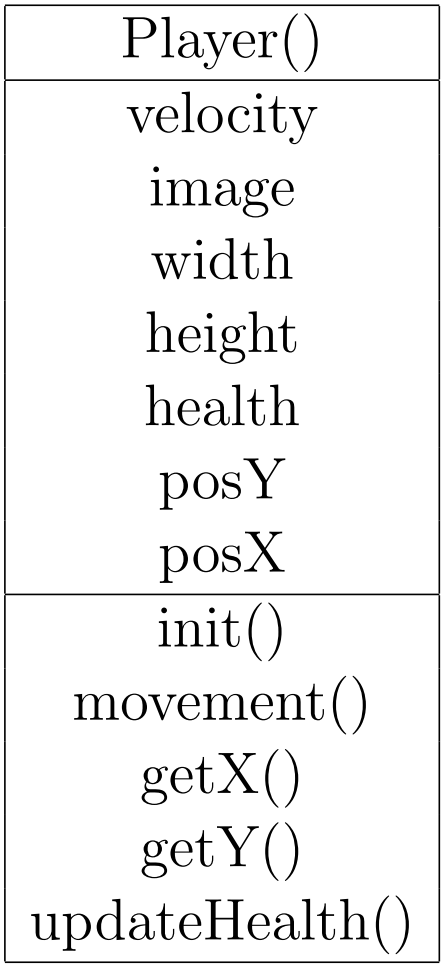
Game()
width height hasGameEnded background FPS lives bullets
main() draw()

This class is required to achieve following requirements:

Must : 3,4,6,7,8,10

Should : 1,7,9,11

This further allows me to abstract each entity into a class, using an Object Oriented Approach, in order to define all sprites from one section of code



This class is required to achieve following requirements:

Must : 5,9,11

Could : 2,3,4,8,10,12

I continue using an OOP approach through my program

Enemy()
velocity image width height health posX posY
init() movement() getX() getY() updateHealth()

This class is required to achieve following requirements:

Must : 7,10,11

Could : 7,8,9,10

This class is very useful, as it is equally used by both Enemy and Player class

Bullet()
speed height bullets
getVelocity() setVelocity() checkCollision()

This is required to achieve following requirements:

Should : 6,13,14

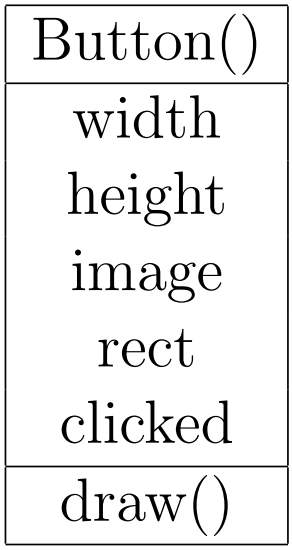
Items()
clock location value
checkCollisionItems() updateHealth() spawnItem() getX() getY()

I defined a tile, which will help me control the size and graphics of the game.

It satisfies:

Tile()
image rect
update()

Level()
display surface world shift tempx dead
setup() scroll x() horizontal collision() vertical collision() player death() run()



4.3 Functionality

4.3.1 Defining the Player class

```
class Player(pygame.sprite.Sprite):

def __init__(self,pos):

    super().__init__() #inherited from a sprite superclass



    #init functions

    self.import_animations()
```

```
#player animations

animation_indx = 0     #shows

animation_vel = 0.15   #how quickly the images change

image = self.animations['idle'][self.animation_indx]

#initial image




#player structure attributes

rect = image.get_rect(topleft = pos)

#blits the self.img to my player sprite



#mechanics

direction = pygame.math.Vector2(0,0)

speed = 8
```

```
    jump_h = -16 # how much player jumps up

    gravity = 0.8  # the gravity on the player



    #player directions

    movement = 'idle' #initial movement

    right = True   #initial direction player is facing

    floor = False

    ceiling = False

    rwall = False

    lwall = False
```

4.3.2 Defining the Tile class

```
class Tile(pygame.sprite.Sprite):

def __init__(self, pos, size):
```

```
        #initialise pygame using super as its a parent class
        super().__init__()


        #new attributes for each tile that will make up my map
        image = pygame.Surface((size, size))
        image.fill('black')
        rect = self.image.get_rect(topleft = pos)
        #pygame uses top left orientation
```

4.3.3 Defining the Button class

```
    class Button():
    def __init__(self, x, y, image, scale):
    width = image.get_width()
height = image.get_height()
image = pygame.transform.scale(
```

```
            image, (int(width*scale),int(height*scale)))

rect = image.get_rect()

rect.topleft = (x, y)

clicked = False
```

4.3.4 Change the x and y position of objects

```
     item movement(keysPressed)

        IF 'a' pressed and player is in bounds THEN

            move left[West]



        IF 'd' pressed and player is in bounds THEN

            move right[East]



        IF 'w' pressed and player is in bounds THEN

            move up[North]
```

```
    IF 's' pressed and player is in bounds THEN
        move down[South]
```

4.3.5 Get function for position

```
item getX()
    return posX


item getY()
    return posY
```

4.3.6 Change players health attribute

```
item updateHealth()
    IF Player collides with a bullet THEN
        player.health -1
    IF Enemy collides with a bullet THEN\\
        enemy.health -1
```

4.3.7 Get velocity of the player

```
item getVelocity()
    return velocity
```

4.3.8 Set velocity of the player

```
item setVelocity(x)
    set velocity to x
```

4.3.9 Check for a collision

```
item checkCollisionsItems()
    IF player collides with item THEN
        player.health +1
```

4.3.10 Generate items

```
item spawnItem()
```

```
    FOR x in range 0 to width

        x = random number


    FOR y in range 0 to height

        y = random number


    assign a new location value of (x,y)

    spawn a new item at that location
```

4.3.11 Import animations

```
import animations()

    folder path = 'path'

    animations = {'idle:[],'run':[],'jump':[],'fall':[]}


    for animation in animations.keys():

        f path = folder path + animation
```

```
        animations[animation] = lol(f path)
```

4.3.12 Animate the player sprite

```
animate()
    current pic = animations[player movement]


    #loop over images
    animation indx += animation vel
    if animation indx >= len(current pic)
        animation indx = 0


    #update the image
    temp = current pic[int(animation indx)]
    if right then
        image = temp
    else
```

```
    image = pygame.flip(temp,True,False)#item x,y



#redefine the rectange

if floor and rwall then

    rect = image.get_rect(bottomright = rect.bottomright)

elif floor and lwall then

    rect = image.get_rect(bottomleft = rect.bottomleft)

elif floor then

    rect = image.get_rect(midbottom = rect.midbottom)

elif ceiling and rwall then

    rect = image.get_rect(topright = rect.topright)

elif ceiling and lwall then

    rect = image.get_rect(topleft = rect.topleft)

elif ceiling then

    rect = image.get_rect(midtop = rect.midtop)
```

4.3.13 Update player gravity

```
    player gravity()

        #y-direction indicates the current vertical vector direction

        direction.y += gravity

        rect.y += direction.y
```

4.3.14 Get players state of movement

```
]


    get movement()

        #if player is going up

        if direction.y < 0 then

            movement = 'jump'

        elif self.direction.y > 1 then

            movement = 'fall'

        else:
```

```
        if direction.x == 0 then
            movement = 'idle'
        else:
            movement = 'run'
```

4.3.15 Change players state when it jumps

```
player jump()
    #simple jump where direction is vertically up,
    but y-value decreases as pygame
    is reversed coordinate system
    direction.y = jump_h
```

4.3.16 Import all images from a folder

```
import folder(path)
surface_list = []
#the folder must only contain images, as any other format file
```

```
    will cause an import error

    folder = os.walk(path)

    for a,b, imgs in folder:

        #loop through the imgs

        for img in imgs:

            full_path = path + '/' + img

            #to handle png images with

            transparent background

            surface = (pygame.image.load(full_path)).convert_alpha()

            surface_list.append(surface)



    return surface_list
```

4.3.17 Draw all sprites and objects

```
    #procedure to draw all sprites and objects onto the screen

    item draw()
```

```
    display a background image using .blit() function

    draw the player onto the screen

    draw the enemy onto the screen

    draw the bullets onto the screen

    draw the health counters onto the screen
```

4.3.18 Define a map

```
levOne_map = [

'                                   ',

'                                   ',

'                                   ',

' XX    XX             XX    XX     ',

' XX P      X      XX              ',

' XXXX         XX         XX       ',

' XXXX       XX      X      XX     ',

' XX    X  XXX     XX   X    XX     ',
```

```
'       X  XXXX    XX  XXX   XXX  X',

'   XXXX  XXXXXX  XX  XXX    XXX   ',

' XXXXXX   XXXXX   X  XXX    X X   ']



#parameters that are used across the game

tileDim = 64

scr_width = 1200

scr_height = len(levOne_map) * tileDim
```

4.3.19 Main loop for the game

```
item main()

    WHILE the game is running:

        fill the background with a given colour



        Check if game is paused:
```

```
    Draw menu buttons
    Check if menu_state = 'main' then
        close the menu screen
    Check if menu_state = 'options' then
        open new menu screen with different options
    Else
        run the Game and close the menu


Poll for events in pygame
    If keys are pressed THEN
        If escape button pressed THEN
            game_paused = 'True'
    If event is pygame.quit THEN
        quit pygame
        call sys exit
```

```
Update the display

Set clock tick to 60
```

4.4 Test Plan

- Need to check code and compare against the requirements
- Carry out multiple iterations, each stage improving the game
- test for crucial conditions such as collisions of the player, health of the player and the enemy

5 Implementation

5.1 Iteration 1

5.1.1 Main Loop

Whenever I tackle a problem, small or large I approach it from different perspectives. Depending on the problem or tasks complexity and functionality I carefully choose the type of programming paradigm that is most appropriate for its implementation. Here, the problem is relatively large, so like most of times, I will develop this game using object oriented techniques otherwise known as OOP. On practical basis it will allow me to break my large problem of a game into much smaller sub-problems that are easier to code, debug and test, hence help me abstract the problem.

Firstly, I needed some basic conditions to be met, to even run my program. I imported a python library named 'pygame' ,which is what I will be using to implement my game, then I implemented a basic loop, to execute my

program.

Code 1:

```
#imports
import pygame
import sys
from parameters import *


#initialise Pygame
pygame.init()
screen = pygame.display.set_mode((scr_width,scr_height))
clock = pygame.time.Clock()

#main loop
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            #forces program to close by the system as well as pygame itself
            pygame.quit()
            sys.exit()

    #basic background filled
    screen.fill('white')

    #refreshes the screen at 60 fps
    pygame.display.update()
    clock.tick(60)
```

This is my main loop to keep the program running, however whenever you have a while loop you must have a condution or in this case an event, which will stop the loop. Here I used an event in pygame, which calls the quit function which ends the pygame loop, furthermore I also added a 'sys.exit()'

to end the actual python file, which was run initially.

5.1.2 Class Player

I created a class for Player in a separate python file, to ensure modularisation and make the process of debugging easier.

```
class Player(pygame.sprite.Sprite):

    def __init__(self,pos):
        super().__init__() #inherited from a sprite superclass

        #init functions
        self.import_animations()

        #player animations
        self.animation_indx = 0     #indicates which action the player is perfoming
        self.animation_vel = 0.15   #how quickly the images change
        self.image = self.animations['idle'][self.animation_indx] #initial image


        #player structure attributes
        self.rect = self.image.get_rect(topleft = pos)  #blits the self.img to my player sprite

        #mechanics
        self.direction = pygame.math.Vector2(0,0)
        self.speed = 8
        self.jump_h = -16 # how much player jumps up
        self.gravity = 0.8  # the gravity on the player

        #player directions
        self.movement = 'idle'     #initial movement
        self.right = True          #initial direction player is facing
        self.floor = False
        self.ceiling = False
        self.rwall = False
        self.lwall = False
```

To create a pygame object, I had to call the sprite superclass

5.1.3 Requirements being developed

5.1.4 Errors

5.1.5 Conclusion

5.2 Iteration 2

5.2.1 Requirements being developed

5.2.2 Errors

5.2.3 Conclusion

5.3 Iteration 3

5.3.1 Requirements being developed

5.3.2 Errors

5.3.3 Conclusion

6 Testing

7 Evaluation