Final Project
-Algorithm and Programming-

Project Name: Man Vs Slime

Name: Justin Theofilus Yonathan

ID: 2502036382

Class: L1CC

Project Specification

For my Algorithm and Programming final project, I decided to create a platformer game using pygame, the reason why I decided to do so was due to a couple of reasons. Firstly, the reason why I chose to make a game was that I wanted to be a game developer, so I wanted to see how it was, what it takes, and just how difficult it was in general to create a game and through this project, I have gained some insight and learned some new things about game making. Secondly, I am a fan of 8bit style games so which adds to my reasoning on choosing this specific genre which is an 8bit styled platformer game, and also as a person who does not have any programming background before joining BINUS International, I found that making this type of game has the perfect difficulty for me in creating it. The game is created using the pygame module which opens up a lot of functions that can make it much easier for me to do certain things in python, which otherwise might take up literally hundreds of lines of code to write and using those functions I was able to create a game window and add in pictures inside it, by using built-in python commands we can then modify these pictures by creating conditions and actions that the system will do once a specific thing is done. In my case, I use an array as a way to store my world data, and once we move on from the main menu screen the images that correspond to the numbers inside the array will be displayed out and using pygame we can also add music, movement, and animations which adds more depth to the game, which I implemented inside my platformer's code. In my game, I made 3 levels that have different tiles to show a change of scenery and add a death animation to add a bit of aesthetic value. There are also restart and replay buttons which are buttons that restart the whole game available only when you completed the last stage and a button that resets the current level available when you die from enemies, respectively. These features add some degree of functionality and convenience for the player. To explain the mechanics of the game which in my opinion is quite straightforward, especially since it's just a platformer game, the main mechanics are just pressing the "a", "d", and space keys to get through the stage while trying not to die by colliding with an enemy.

Input

- Keypresses
- Mouse Clicks
- Mouse Hover Position

Output

- Movement
- Page Changes (Buttons)

Solution Design

- Main Menu
- Stages(1-3)
- Death Screen
- Win Screen

Main Menu

The main menu page is the page before entering the stage, I made it so that I can display the title of the game and make it so that players won't have a rushed feeling every time they start the game, the start button is there as a way to progress onto the stages and start the actual game.

Stages

The stages are the star of the game, they are what people are going to expect to see the most when they start the game. In the stages, all of them have pretty much the same assets which are tiles, enemies, and an end gate which they must go through to progress onto the next stage/phase.

Death Screen

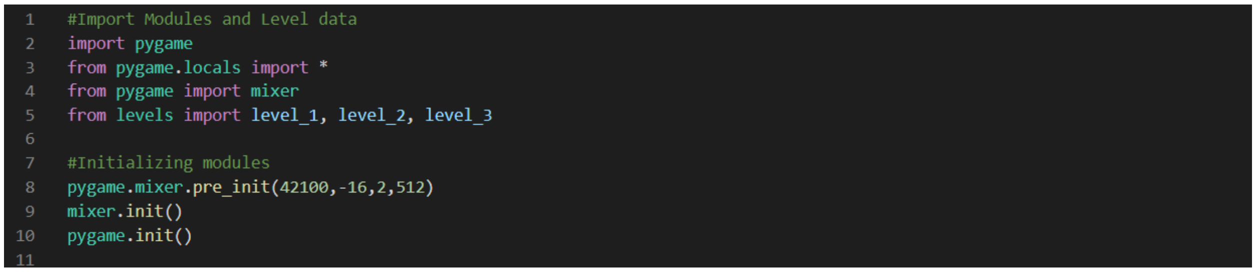
The death screen is less of a screen and more like a pop-up of an Image, but within that pop-up, I put a button that has the function to reset the stage that they are currently in.

Win Screen

Similar to the death screen, the win screen is also more of a pop-up of writing to show that they've won and the game has ended, and like the death screen too, I added a button to restart the game from stage 1 and from there the player can experience the game again.

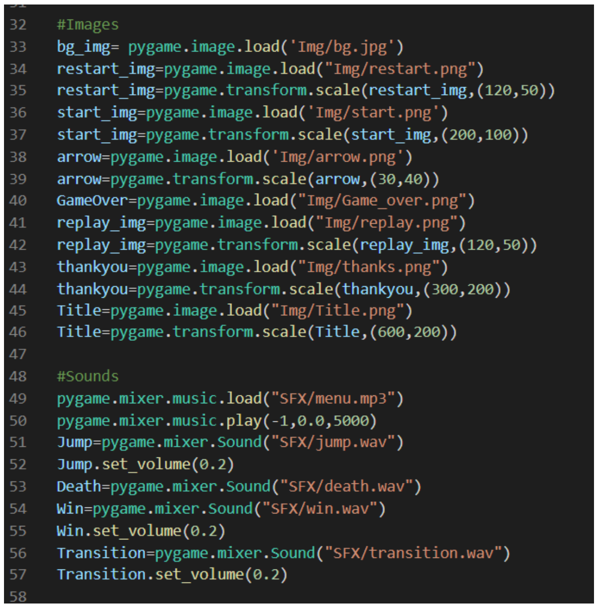
Code Explanation

In this code explanation, I will not be explaining every code that is in my game, I will explain the lines of code that are the most important or add interesting parts to the game. So to begin, the way that my code works is first when you press the run button in your editor, It initializes the modules and imports the classes and functions that will be used in the game.



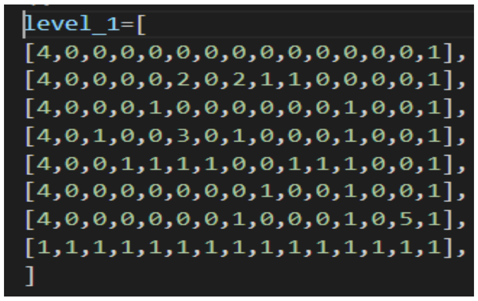
```python
#Import Modules and Level data
import pygame
from pygame.locals import *
from pygame import mixer
from levels import level_1, level_2, level_3

#Initializing modules
pygame.mixer.pre_init(42100,-16,2,512)
mixer.init()
pygame.init()

```

After that, it loads in the menu screen which is basically just the background image and the game logo with a button below it which is displayed on the screen using the blit function from pygame of the images that are loaded in using the pygame.image.load() function, and during the main menu screen there is also music playing which uses a pygame mixer function [.play()].

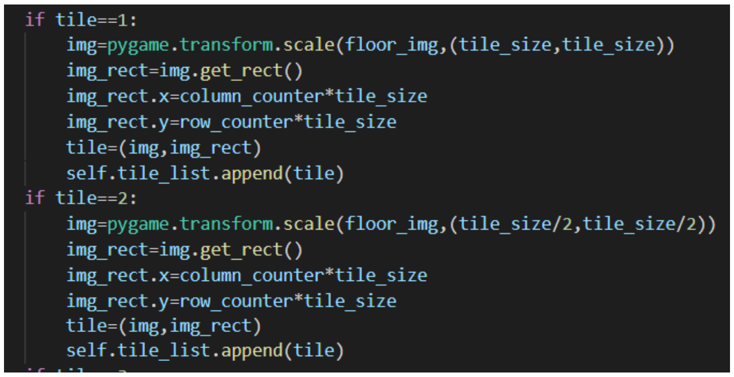


```python
#Images
bg_img= pygame.image.load('Img/bg.jpg')
restart_img=pygame.image.load("Img/restart.png")
restart_img=pygame.transform.scale(restart_img,(120,50))
start_img=pygame.image.load('Img/start.png')
start_img=pygame.transform.scale(start_img,(200,100))
arrow=pygame.image.load('Img/arrow.png')
arrow=pygame.transform.scale(arrow,(30,40))
GameOver=pygame.image.load("Img/Game_over.png")
replay_img=pygame.image.load("Img/replay.png")
replay_img=pygame.transform.scale(replay_img,(120,50))
thankyou=pygame.image.load("Img/thanks.png")
thankyou=pygame.transform.scale(thankyou,(300,200))
Title=pygame.image.load("Img/Title.png")
Title=pygame.transform.scale(Title,(600,200))

#Sounds
pygame.mixer.music.load("SFX/menu.mp3")
pygame.mixer.music.play(-1,0.0,5000)
Jump=pygame.mixer.Sound("SFX/jump.wav")
Jump.set_volume(0.2)
Death=pygame.mixer.Sound("SFX/death.wav")
Win=pygame.mixer.Sound("SFX/win.wav")
Win.set_volume(0.2)
Transition=pygame.mixer.Sound("SFX/transition.wav")
Transition.set_volume(0.2)

```

So when you press the start button it starts getting and displaying the world data, which is basically just multiple different information of the Player, Enemy, Blocks, and Gate instances which is represented as a number that was defined using a number in the level data array. So when the code runs the main loop, it reads the number on the array and assigns it to a specific image according to the number that has been assigned to it in the World() class.

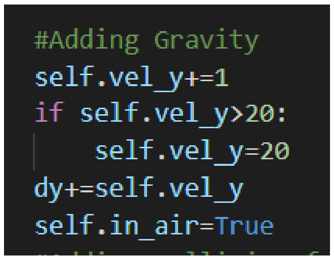


```
level_1=[
[4,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
[4,0,0,0,0,2,0,2,1,1,0,0,0,0,1],
[4,0,0,0,1,0,0,0,0,0,0,1,0,0,1],
[4,0,1,0,0,3,0,1,0,0,0,1,0,0,1],
[4,0,0,1,1,1,1,0,0,1,1,1,0,0,1],
[4,0,0,0,0,0,0,0,1,0,0,1,0,0,1],
[4,0,0,0,0,0,0,1,0,0,0,1,0,5,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
]
```

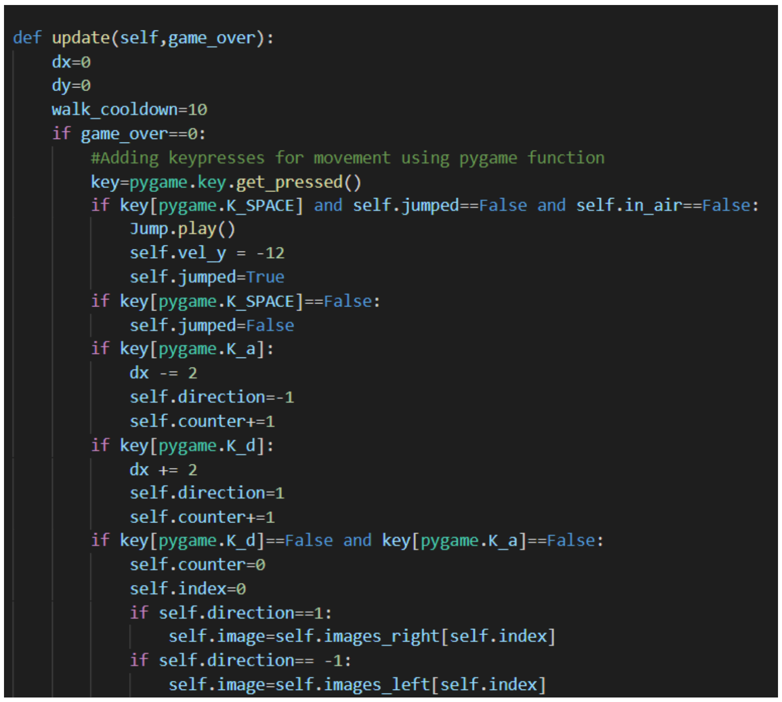


```
if tile==1:
    img=pygame.transform.scale(floor_img,(tile_size,tile_size))
    img_rect=img.get_rect()
    img_rect.x=column_counter*tile_size
    img_rect.y=row_counter*tile_size
    tile=(img,img_rect)
    self.tile_list.append(tile)
if tile==2:
    img=pygame.transform.scale(floor_img,(tile_size/2,tile_size/2))
    img_rect=img.get_rect()
    img_rect.x=column_counter*tile_size
    img_rect.y=row_counter*tile_size
    tile=(img,img_rect)
    self.tile_list.append(tile)
```

Now getting into the physics of the World specifically the player, to make the player have realistic properties such as jumping, walking, and also colliding with objects. Firstly jumping, when jumping the player basically floats up with a limiter and then gets pulled back down by an acceleration which is done by creating a y-axis velocity which increases by itself per tick which gives us an acceleration variable that can be edited by changing the velocity which gives the player gravity, this is then assigned to a key using the pygame function pygame.key.get_pressed() and by assigning that function to the key variable we can add more conditions to this function such as the one used for jumping which is the space bar, so the code will be key[pygame.K_SPACE] which basically mentions that "if the space key is being pressed then…", so by setting this as True and adding other conditions we can assign the jumping motion to the space bar key.

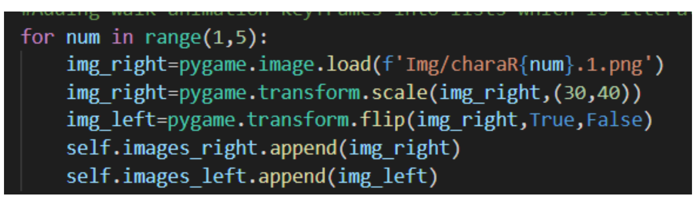


```
#Adding Gravity
self.vel_y+=1
if self.vel_y>20:
    self.vel_y=20
dy+=self.vel_y
self.in_air=True
```

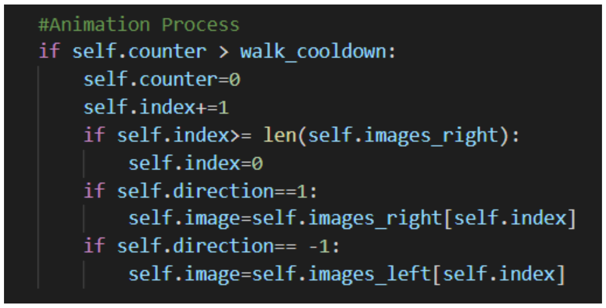


```
def update(self,game_over):
    dx=0
    dy=0
    walk_cooldown=10
    if game_over==0:
        #Adding keypresses for movement using pygame function
        key=pygame.key.get_pressed()
        if key[pygame.K_SPACE] and self.jumped==False and self.in_air==False:
            Jump.play()
            self.vel_y = -12
            self.jumped=True
        if key[pygame.K_SPACE]==False:
            self.jumped=False
        if key[pygame.K_a]:
            dx -= 2
            self.direction=-1
            self.counter+=1
        if key[pygame.K_d]:
            dx += 2
            self.direction=1
            self.counter+=1
        if key[pygame.K_d]==False and key[pygame.K_a]==False:
            self.counter=0
            self.index=0
            if self.direction==1:
                self.image=self.images_right[self.index]
            if self.direction== -1:
                self.image=self.images_left[self.index]
```

Secondly, we have walking, which basically implements the same way of thinking as the way jumping words for assigning the key, as for the movement itself it's made by creating a variable dx and dy, which signifies a change in position of the player, dy is used to for jumping and dx is used in walking, it is done by moving the player image according to the number of change in the value of dx in addition of a counter to keep track of how much movement is being done by the player, next we have the walking animation which is done by putting a series of images in a list and iterating through those images according to a walk cooldown variable which basically allows us to adjusts the movement of the player to the speed that he's going at.

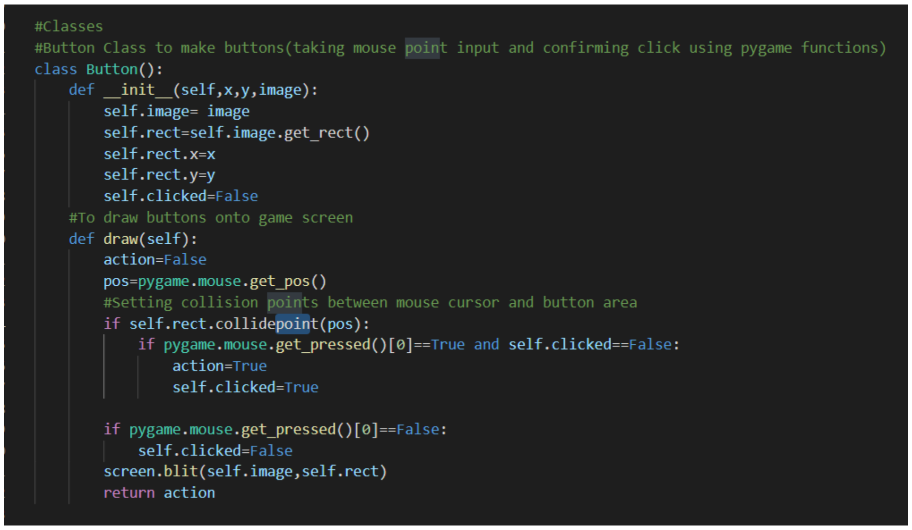


```
for num in range(1,5):
    img_right=pygame.image.load(f'Img/charaR{num}.1.png')
    img_right=pygame.transform.scale(img_right,(30,40))
    img_left=pygame.transform.flip(img_right,True,False)
    self.images_right.append(img_right)
    self.images_left.append(img_left)
```

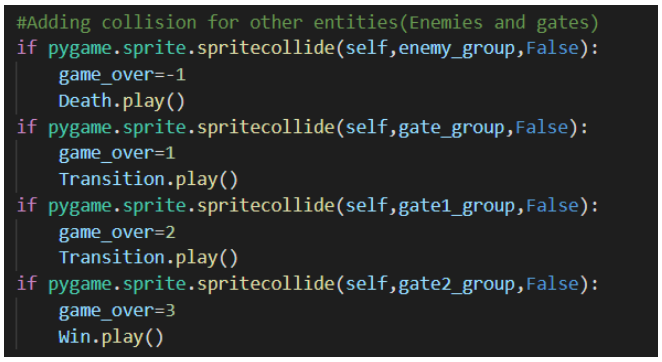


```
#Animation Process
if self.counter > walk_cooldown:
    self.counter=0
    self.index+=1
    if self.index>= len(self.images_right):
        self.index=0
    if self.direction==1:
        self.image=self.images_right[self.index]
    if self.direction== -1:
        self.image=self.images_left[self.index]
```

Lastly, we have the collision, for this part, I do it by assigning a rectangle to every image when they were loaded in first by using .image.get_rect() and then using that as a way to see what the player is touching with we can use another function colliderect(), spritecollide(), and collidepoint(), colliderect is for collision between rectangles set to an image which in my case is on the world blocks and player, and spritecollide is for contact with sprite groups such as enemies and gates as for collidepoint it's used for buttons so that when the mouse cursor floats on top of an image it receives an input.

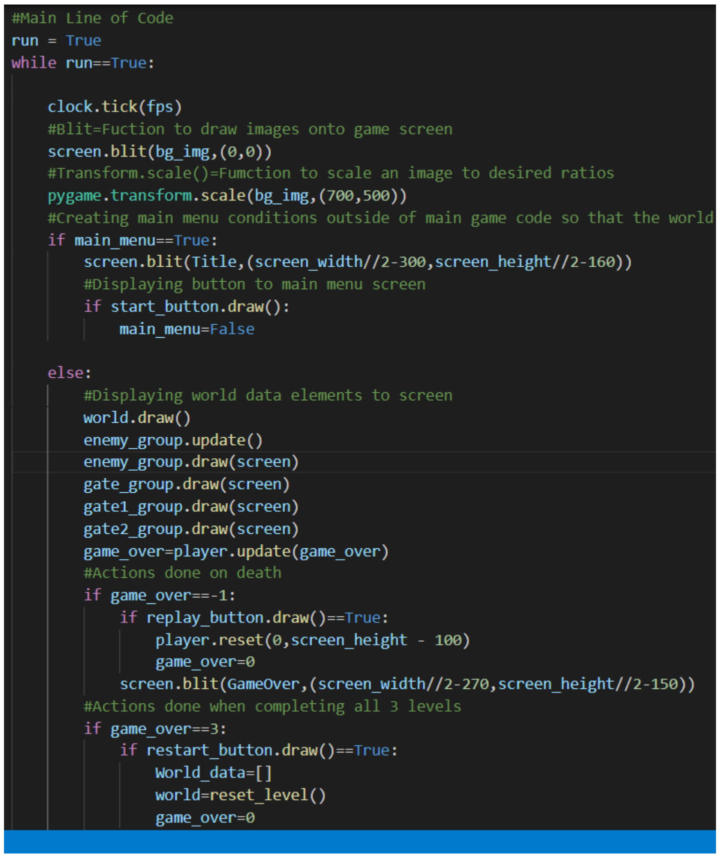


```
#Classes
#Button Class to make buttons(taking mouse point input and confirming click using pygame functions)
class Button():
    def __init__(self,x,y,image):
        self.image= image
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y
        self.clicked=False
    #To draw buttons onto game screen
    def draw(self):
        action=False
        pos=pygame.mouse.get_pos()
        #Setting collision points between mouse cursor and button area
        if self.rect.collidepoint(pos):
            if pygame.mouse.get_pressed()[0]==True and self.clicked==False:
                action=True
                self.clicked=True

        if pygame.mouse.get_pressed()[0]==False:
            self.clicked=False
        screen.blit(self.image,self.rect)
        return action
```

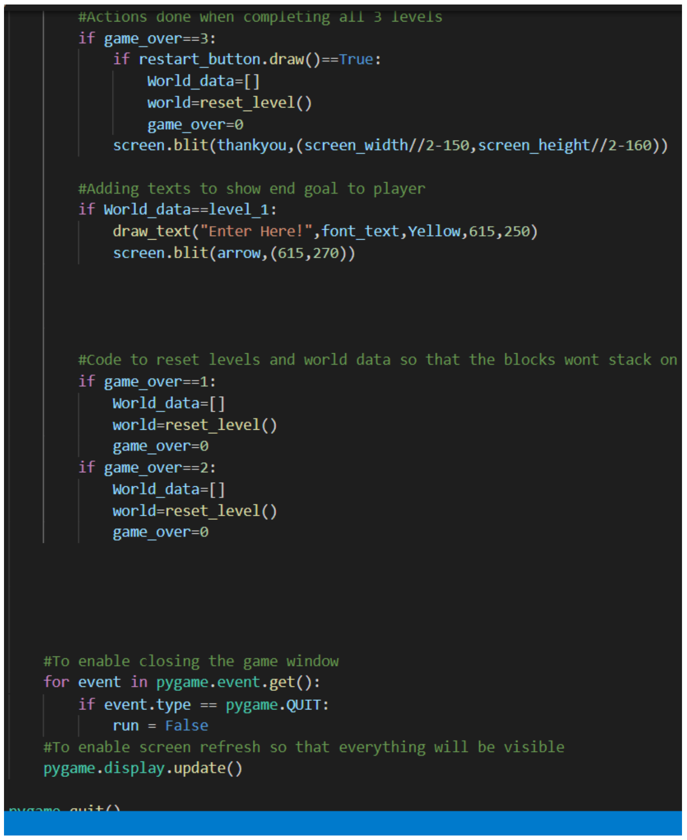


```
#Adding collision for other entities(Enemies and gates)
if pygame.sprite.spritecollide(self,enemy_group,False):
    game_over=-1
    Death.play()
if pygame.sprite.spritecollide(self,gate_group,False):
    game_over=1
    Transition.play()
if pygame.sprite.spritecollide(self,gate1_group,False):
    game_over=2
    Transition.play()
if pygame.sprite.spritecollide(self,gate2_group,False):
    game_over=3
    Win.play()
```

Next, we have the main line of code that is used to actually run the game, this line of code works by setting the run variable to True and we create a while loop to run the game based on that statement, and if we press the “X” button on the top right of the window, the run variable is then set to False and the while loop ends, hence closing the game window. There are a few other codes inside the while loop which is basically just [.blit()] functions, which is for displaying images and other assets onto the screen, also there are a few if statements that are used to set up certain conditions when running the game, which I already explained using the comments inside the editor.

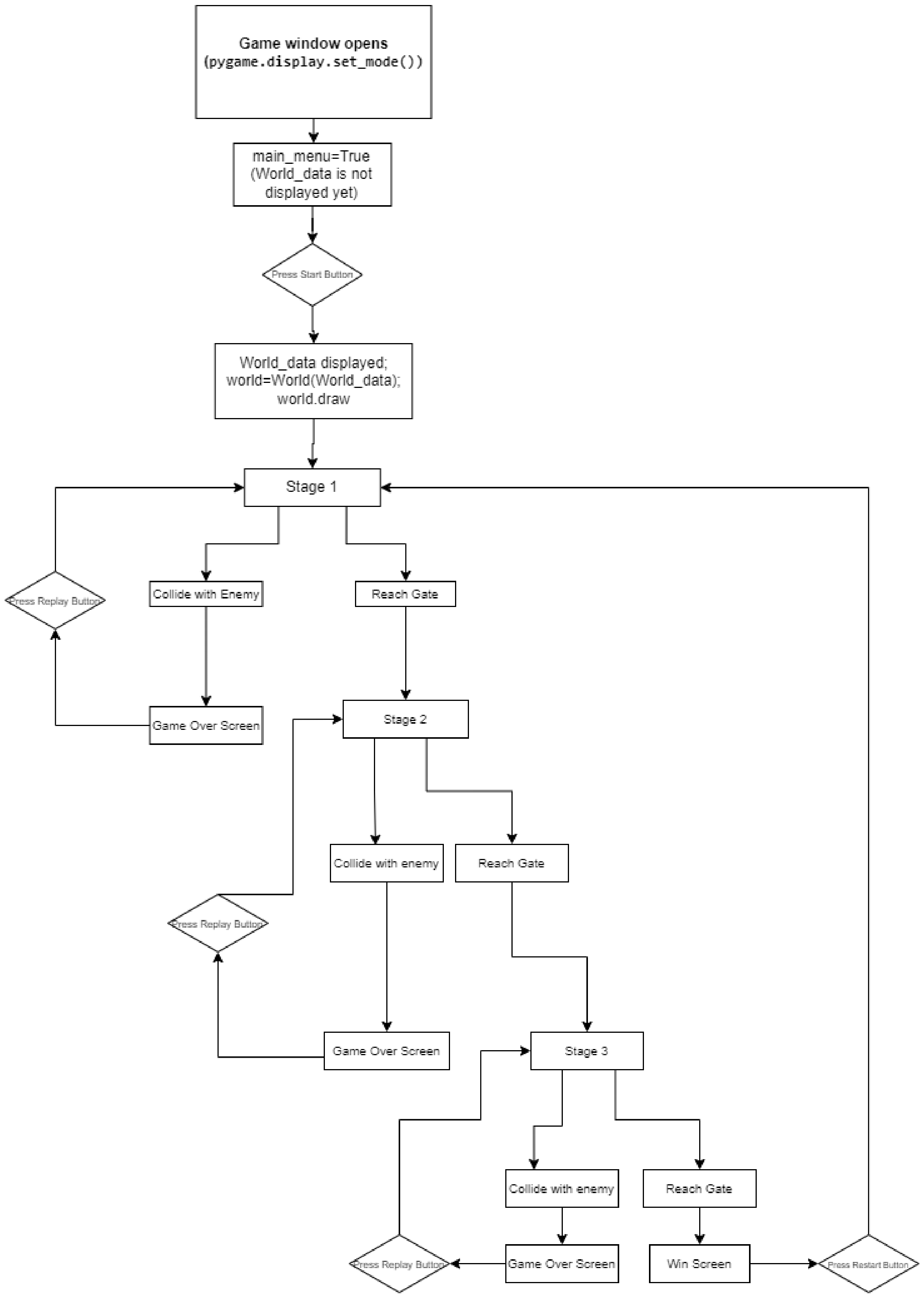


```
#Main Line of Code
run = True
while run==True:

    clock.tick(fps)
    #Blit=Fuction to draw images onto game screen
    screen.blit(bg_img,(0,0))
    #Transform.scale()=Fumction to scale an image to desired ratios
    pygame.transform.scale(bg_img,(700,500))
    #Creating main menu conditions outside of main game code so that the world
    if main_menu==True:
        screen.blit(Title,(screen_width//2-300,screen_height//2-160))
        #Displaying button to main menu screen
        if start_button.draw():
            main_menu=False

    else:
        #Displaying world data elements to screen
        world.draw()
        enemy_group.update()
        enemy_group.draw(screen)
        gate_group.draw(screen)
        gate1_group.draw(screen)
        gate2_group.draw(screen)
        game_over=player.update(game_over)
        #Actions done on death
        if game_over==-1:
            if replay_button.draw()==True:
                player.reset(0,screen_height - 100)
                game_over=0
            screen.blit(GameOver,(screen_width//2-270,screen_height//2-150))
        #Actions done when completing all 3 levels
        if game_over==3:
            if restart_button.draw()==True:
                World_data=[]
                world=reset_level()
                game_over=0
```

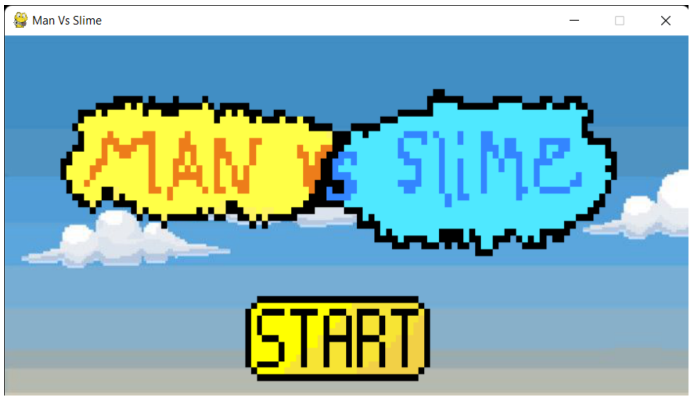


```
        #Actions done when completing all 3 levels
        if game_over==3:
            if restart_button.draw()==True:
                World_data=[]
                world=reset_level()
                game_over=0
            screen.blit(thankyou,(screen_width//2-150,screen_height//2-160))

        #Adding texts to show end goal to player
        if World_data==level_1:
            draw_text("Enter Here!",font_text,Yellow,615,250)
            screen.blit(arrow,(615,270))



        #Code to reset levels and world data so that the blocks wont stack on
        if game_over==1:
            World_data=[]
            world=reset_level()
            game_over=0
        if game_over==2:
            World_data=[]
            world=reset_level()
            game_over=0




    #To enable closing the game window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    #To enable screen refresh so that everything will be visible
    pygame.display.update()

pygame.quit()
```

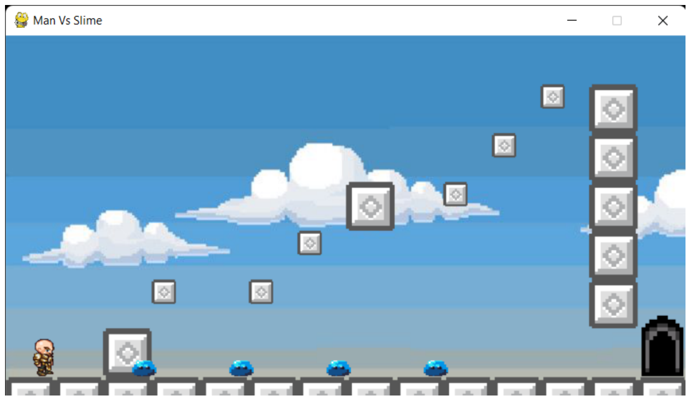
Flowchart


Game window opens
(pygame.display.set_mode())
main_menu=True (World_data is not displayed yet)
Press Start Button
World_data displayed; world=World(World_data); world.draw
Stage 1
Collide with Enemy
Reach Gate
Press Replay Button
Game Over Screen
Stage 2
Collide with enemy
Reach Gate
Press Replay Button
Game Over Screen
Stage 3
Collide with enemy
Reach Gate
Press Replay Button
Game Over Screen
Win Screen
Press Restart Button

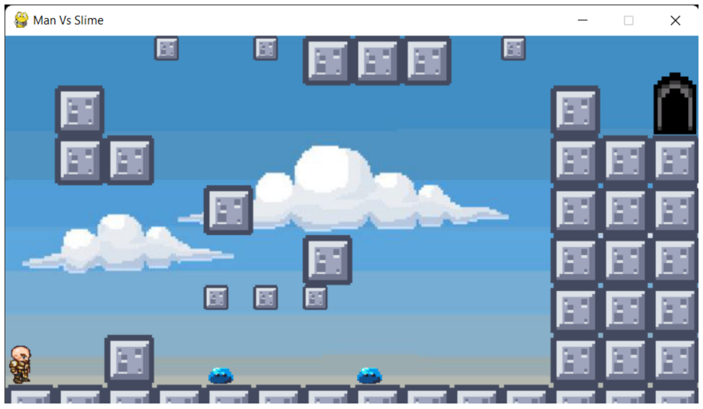
Working Program Screen Shots



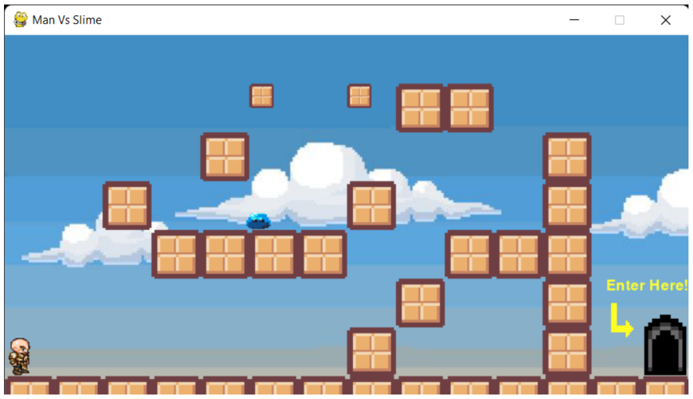
Main Menu Page



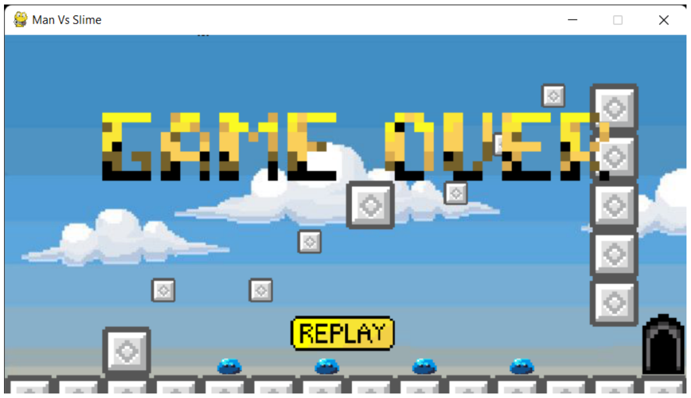
Level 3



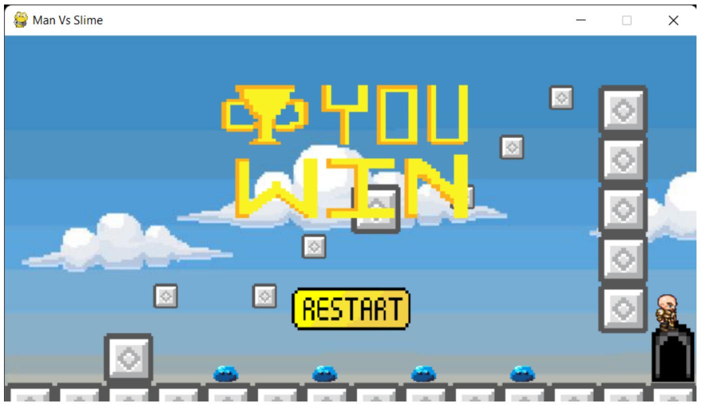
Level 2



Level 1



Death Screen



Win Screen

Reflection

To me, this project has been an eye-opener to how games are made, and I believe that because of doing this project, I am now one step closer to making my dream come true. At first, I certainly struggled quite a bit, especially since I have to learn about pygame from scratch since I have never used it before, but thankfully through watching some videos on YouTube and reading on some websites online, I managed to get a grasp of the functions that was provided by the module. Thankfully, throughout making the game I didn't encounter any fatal errors that would have forced me to restart from the beginning or anything similar to that, the only challenge that I had was a problem I had before I started my Final Project. For some reason my pip3 wasn't able to install any modules at all, at the time I wasn't even trying to install pygame, I was trying to install matpolib the Algorithm and Programming Session about making visuals in python, it was quite tedious because I couldn't find anything on the internet that worked, and I ended up having to reinstall VSCode and Python multiple times. Thankfully, after a couple of reinstalls, it finally worked and I was able to begin my Final Project. Through making this game, I have learned many things about pygame, the name of the functions, their uses, and how they can be combined to create some pretty fascinating things. but I'm sure that I'm still far away from mastering it, and that I have a lot left to learn if I want to get close at all to realizing my dream. To sum up, I am very thankful for the opportunity given to me, to be able to go through this process, and I am hoping to learn more in the future.

References:

- https://www.youtube.com/channel/UCPrRY0S-VzekrJK7I7F4-Mg
- http://pixelartmaker.com/
- https://app.diagrams.net/
- https://soundcloud.com/aaron-anderson-11/sets/rpg-maker-music-loops
- https://kenney.nl/assets/pixel-platformer-blocks
- https://mixkit.co/free-sound-effects/game/?page=2
- https://groovymaster.itch.io/groovys-ghost-pack
- https://stealthix.itch.io/animated-slimes
- https://github.com/techwithtim/Pygame-Tutorials/tree/master/Game
- https://id.pinterest.com/pin/664984701219839307/