Odd Semester (2021)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:

Surname: Silalahi **Given Name:** Joshua Alexander **Student ID Number:** 2502005244

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

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

Type of Assignments: Term Final Project

Submission Pattern

Due Date : 17 January 2022 **Submission Date :** 17 January 2022

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:

TABLE OF CONTENTS

I. Game Overview

A. Genre

The name of this game is “Spiderman: Into the 8-Bit-Verse” which of course hints towards this being an 8-bit 2D platformer game. This game of course will have a style that you would often see from a Retro-Game that you would now mostly play on Emulators.

B. Game Methods and Concepts

The player would of course play as Spiderman with a classic Spiderman costume design similar to the Spiderman cartoon in the 80s and 90s. The Spiderman that the player would be playing is able to shoot webs, jump, throw web grenades with an aim to take-down all enemies (Venom) to save the day.

C. Gameplay Visuals

The visual of the game is in 8-bit and retro-style. Similar visuals on how you would find on retro-consoles such as the Nintendo Gameboy, Sega Genesis, Playstation 1, or any other consoles from the 80s and 90s era.

D. Gameplay Summary and Objective

The story of the game is that a great part of New York is infected by the Venom Symbiote. Spiderman must defeat the Symbiotes in order to keep New York safe again. The player would be playing as Spiderman and have the ability to shoot webs, throw some web grenades, as well as jump from buildings to buildings. But the player needs to watch out for obstacles as well as their Web Ammunitions and of course their health as Spiderman is not immune to any damage.

II. INSPIRATION

As you can see, this game is of course really inspired by the Spiderman comics, moves, and cartoons of course. However, the most heavy impact on this game is of course the Spiderman game from the Nintendo GameBoy, which I often played during my childhood and found to be quite enjoyable to play, which makes this game bring about some nostalgic childhood memories.

III. GAME MECHANICS

A. Assets and Musics

All of the Assets whether it be images, Audio, or Music is stored inside the game assets file which could be called out by demand from the main program file. All of the assets are assigned to a variable to give easier access to each of them. The game assets file not only stores assets, but it also stores the game's variables like gravity, scroll speed, tile sizes, starting level, max level, screen sizes, etc.

(game assets file)

```
#define game variables
GRAVITY = 0.75
SCROLL_THRESH = 200
ROWS = 16
COLS = 150
TILE_SIZE = SCREEN_HEIGHT // ROWS
TILE_TYPES = 21
MAX_LEVELS = 2
screen_scroll = 0
bg_scroll = 0
level = 1
start_game = False
start_intro = False


#define player action variables
moving_left = False
moving_right = False
shoot = False
grenade = False
grenade_thrown = False

#-------------------------------------------------------------------------------------------------------------------------#
#ASSETS

#load music and sounds
pygame.mixer.music.load('audio/SpidermanTheme.mp3')
pygame.mixer.music.set_volume(0.4)
pygame.mixer.music.play(-1, 0.0, 5000)
jump_fx = pygame.mixer.Sound('audio/jump.wav')
jump_fx.set_volume(0.05)
shot_fx = pygame.mixer.Sound('audio/shot.wav')
shot_fx.set_volume(0.05)
grenade_fx = pygame.mixer.Sound('audio/grenade.wav')
grenade_fx.set_volume(0.05)


#IMAGES
#button images
start_img = pygame.image.load('img/start_btn.png').convert_alpha()
exit_img = pygame.image.load('img/exit_btn.png').convert_alpha()
restart_img = pygame.image.load('img/restart_btn.png').convert_alpha()
#game-background
city4_img = pygame.image.load('img/Background/city_background.jpg').convert_alpha()
city_img = pygame.image.load('img/Background/city_background.jpg').convert_alpha()
city2_img = pygame.image.load('img/Background/city_background.jpg').convert_alpha()
city3_img = pygame.image.load('img/Background/city_background.jpg').convert_alpha()
city4_img = pygame.image.load('img/Background/city_background.jpg').convert_alpha()
#menu-background
menu_web = pygame.image.load('img/Background/web wallpaper.jpg')
```

As for the music, I made it on my own using Logic Pro X on my Macbook Pro. The music is actually the soundtrack of the Spiderman movie and Amazing Spiderman cartoon, but I rearranged it and made a retro/arcade style version. I aimed it to be accurate to the 8-bit style so I made it using retro synths and retro drum loops to give a more arcade vibe to the music. The music is also set to loop all the time and 0.4 is set as the loudness of the volume.

(using pygame mixer to load musics and audio)

```
#load music and sounds
pygame.mixer.music.load('audio/SpidermanTheme.mp3')
pygame.mixer.music.set_volume(0.4)
pygame.mixer.music.play(-1, 0.0, 5000)
```

(the Logic Pro X project file for the music)

B. Main Menu

To make the Main Menu, I use the draw_menu() function to set its background. Then use the screen.fill(bg) to make it fill the entire display. Next, make a for loop with a screen.blit function to adjust and crop to fit the screen.

(draw_menu function to draw the background)

```
30  #MENU BG
31  def draw_menu():
32      screen.fill(BG)
33      width = menu_web.get_width()
34      for n in range(1):
35          screen.blit(menu_web, ((n * width) - bg_scroll * 0.5, 0))
```

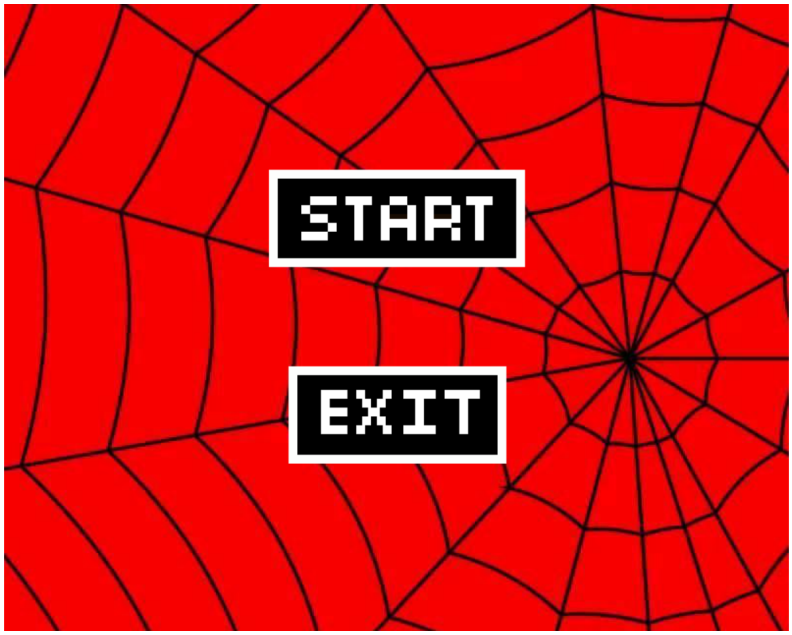
After the main menu has been made, the draw_menu function could be assigned to the main game loop by simply just calling the draw_menu function. A boolean is used to identify whether the game should be in the main menu or inside the game, so if start_game == False then it would open the main menu, if start_game == True then it would open the main game.

```
run = True
while run:

    clock.tick(FPS)

    if start_game == False:
        #MENU
        draw_menu()
        #add buttons
        if start_button.draw(screen):
            start_game = True
            start_intro = True
        if exit_button.draw(screen):
            run = False
```

(press START to begin game, press EXIT to exit game/program)



If start_game == True

C. World, Level, Environment, ItemBoxes

World and Level

First, I made a draw_bg function to make some adjustments to the background image. Setting the for loops to stack up 5 images as the grid of the screen is separated into 5 layers. And of course each of the grids has different properties, depending on its position. These 5 layers of image would then create a far and near perspective towards the background.

```
#GAME BG
def draw_bg():
    screen.fill(BG)
    width = city2_img.get_width()
    for n in range(5):
        screen.blit(city2_img, ((n * width) - bg_scroll * 0.5, 0))
        screen.blit(city_img, ((n * width) - bg_scroll * 0.6, SCREEN_HEIGHT - city_img.get_height() - 300))
        screen.blit(city4_img, ((n * width) - bg_scroll * 0.7, SCREEN_HEIGHT - city4_img.get_height() - 150))
        screen.blit(city3_img, ((n * width) - bg_scroll * 0.8, SCREEN_HEIGHT - city3_img.get_height()))
```

As for the level, the level is created using a csv file. The csv file would then place the tiles of the world accordingly by identifying the name of the file. The file is placed using numbers in which the program could access by using a for loop.

(csv data for level 1)

level1_data.csv

```
1  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
2  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
3  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
4  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
5  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
6  0,2,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
7  4,5,-1,-1,-1,-1,-1,-1,-1,13,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
8  4,5,-1,-1,-1,-1,-1,-1,6,7,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,16,-1,-1,-1,-1,6,7,
9  4,5,-1,12,-1,-1,-1,-1,-1,-1,-1,-1,11,14,14,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,12,14,-1,-1,-1,-
10 4,4,0,0,2,-1,-1,-1,-1,-1,-1,-1,6,7,7,8,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,17,14,-1,-1,6,7,7,7,8,-1,-1,-1,-1,-1,-1,-
11 4,4,4,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,-1,14,16,14,13,-1,-1,-1,-1,-1,-1,-1,14,-1,-1,1,0,2,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
12 4,4,4,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,7,7,7,7,7,8,-1,-1,-1,-1,13,-1,0,-1,-1,3,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
13 4,4,4,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,14,1,0,5,9,9,3,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
14 4,4,4,4,5,-1,-1,-1,-1,-1,-1,-1,11,1,0,2,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,0,4,4,5,10,10,3,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
15 4,4,4,4,5,14,-1,15,14,13,14,1,0,4,4,4,2,-1,-1,-1,14,16,11,14,14,-1,1,4,4,4,4,5,10,10,3,4,5,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-
16 4,4,4,4,4,0,0,0,0,0,0,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4,4,4,4,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
17
```

All of the tiles are then stored in an empty list and use pygame.image.load function to load the images, and use the pygame.image.scale to scare the image and fit them into a square grid, as the tile is placed in the game inside of a square grid with an x or y axis.

```
#store tiles in a list
img_list = []
for x in range(TILE_TYPES):
    img = pygame.image.load(f'img/Tile/{x}.png')
    img = pygame.transform.scale(img, (TILE_SIZE, TILE_SIZE))
    img_list.append(img)
```

The world class is used to identify what type of tile is placed in each part. For example tile 0-8 is the platform tile in which the player can walk above it. Tile 9-10 is water in which the player would die if they walk above them by using the Water Class which would be discussed later in this report. Tile 11-14 is just environment decorations, tile 15 is for placing the Spiderman, tile 16 is for placing Venom enemies in the game world. Tiles 17-19 are for item boxes, and tile 20 is for the exit logo to the next level. All of the tiles would then be set up with a class or function of their own which would give them purpose in the game world, according to their designated purposes.

(the World Class mentioned on the above paragraph)

```python
#PROGRAMS - WORLD and ENVIRONMENTS

class World():
    def __init__(self):
        self.obstacle_list = []

    def process_data(self, data):
        self.level_length = len(data[0])
        #iterate through each value in level data file
        for y, row in enumerate(data):
            for x, tile in enumerate(row):
                if tile >= 0:
                    img = img_list[tile]
                    img_rect = img.get_rect()
                    img_rect.x = x * TILE_SIZE
                    img_rect.y = y * TILE_SIZE
                    tile_data = (img, img_rect)
                    if tile >= 0 and tile <= 8:
                        self.obstacle_list.append(tile_data)
                    elif tile >= 9 and tile <= 10:
                        water = Water(img, x * TILE_SIZE, y * TILE_SIZE)
                        water_group.add(water)
                    elif tile >= 11 and tile <= 14:
                        decoration = Decoration(img, x * TILE_SIZE, y * TILE_SIZE)
                        decoration_group.add(decoration)
                    elif tile == 15:#add player
                        player = Spidey('player', x * TILE_SIZE, y * TILE_SIZE, 1.65, 5, 20, 5)
                        health_bar = HealthBar(10, 10, player.health, player.health)
                    elif tile == 16:#add Venom as enemy
                        venom = Spidey('enemy', x * TILE_SIZE, y * TILE_SIZE, 1.65, 2, 20, 0)
                        enemy_group.add(venom)
                    elif tile == 17:#add ammo box
                        item_box = ItemBox('Ammo', x * TILE_SIZE, y * TILE_SIZE)
                        item_box_group.add(item_box)
                    elif tile == 18:#create grenade box
                        item_box = ItemBox('Grenade', x * TILE_SIZE, y * TILE_SIZE)
                        item_box_group.add(item_box)
                    elif tile == 19:#create health box
                        item_box = ItemBox('Health', x * TILE_SIZE, y * TILE_SIZE)
                        item_box_group.add(item_box)
                    elif tile == 20:#create exit
                        exit = Exit(img, x * TILE_SIZE, y * TILE_SIZE)
                        exit_group.add(exit)

        return player, health_bar
```

(tile folder, types of tile identified using for loop)

Water

The water class is created using the pygame sprite animation which would help in placing them in tiles accordingly as well as placing them inside of the square grid so it doesn't protrude and impact other parts of the game world.

So, using this class and function, the water images would then be placed in the x and y grid as well as well as being able to move aside if the player moves away using the self.rect.x += screen_scroll.

```
363  class Water(pygame.sprite.Sprite):
364      def __init__(self, img, x, y):
365          pygame.sprite.Sprite.__init__(self)
366          self.image = img
367          self.rect = self.image.get_rect()
368          self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))
369
370      def update(self):
371          self.rect.x += screen_scroll
372
```

Of course if Spiderman hits the water we would assume that he would drown because the water is infected with the venom symbiote that is infecting the city, so a function to the Spiderman/player class was made if the player collides with water. So, if the Sprite Animation of Spiderman collides with water the health would automatically be 0 and Spiderman would die, which would of course take the player to the death screen and the player is given an option to restart level or quit to main menu.

```
166
167
168          #check for collision with water
169          if pygame.sprite.spritecollide(self, water_group, False):
170              self.health = 0
171
```

Environment

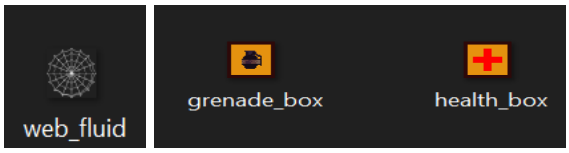
The environment does not require as much command or functionality as the other elements, as the sole purpose of the decoration is just to give tiny bits of spices so the world does not feel too empty. So the decorations class below would just perform its task to set up the decorations according to the grid as well as calling their image variable from the assets file.

```
class Decoration(pygame.sprite.Sprite):
    def __init__(self, img, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = img
        self.rect = self.image.get_rect()
        self.rect.midtop = (x + TILE_SIZE // 2, y + (TILE_SIZE - self.image.get_height()))

    def update(self):
        self.rect.x += screen_scroll
```

ItemBoxes

```
84  #pick up boxes
85  health_box_img = pygame.image.load('img/icons/health_box.png').convert_alpha()
86  ammo_box_img = pygame.image.load('img/icons/web_fluid.png').convert_alpha()
87  grenade_box_img = pygame.image.load('img/icons/grenade_box.png').convert_alpha()
88  item_boxes = {
89      'Health'    : health_box_img,
90      'Ammo'      : ammo_box_img,
91      'Grenade'   : grenade_box_img
92  }
```



The images of the item boxes are set in as variables and are stored inside of a list, which would then be easier to call upon. There are 3 types of item boxes in which Spiderman can pick-up, the first one is the web fluid, for his web ammo. The second item box is the health box, which is pretty self explanatory as it would increase Spiderman's health. The last item box is the grenade box which would help to refill Spiderman's trusty web-grenade.

```
def update(self):
    #scroll
    self.rect.x += screen_scroll
    #check if the player has picked up the box
    if pygame.sprite.collide_rect(self, player):
        #check the item in the box
        if self.item_type == 'Health':
            player.health += 25
            if player.health > player.max_health:
                player.health = player.max_health
        elif self.item_type == 'Ammo':
            player.ammo += 15
        elif self.item_type == 'Grenade':
            player.grenades += 3
        #delete the item box
        self.kill()
```

This function here would update Spiderman's health, web-ammo, or web-grenade if it's picked up. If a health box is being picked up, player health would increase by 25%, if the Web-Ammo box is being picked up, then Spiderman's ammo would increase by 15, and if grenade ammo is picked up, grenade ammo would increase by 3. All the item boxes that are picked up would directly disappear using the self.kill() function.

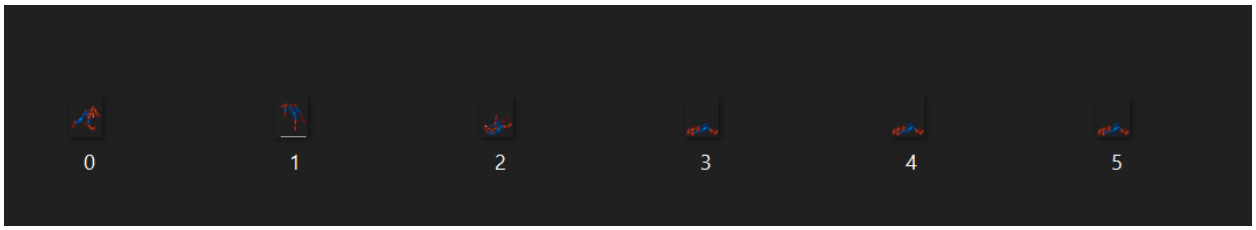
D. Spiderman and Venom (Player and Enemies)

Spiderman/Player

```
class Spidey(pygame.sprite.Sprite):
    def __init__(self, char_type, x, y, scale, speed, ammo, grenades):
        pygame.sprite.Sprite.__init__(self)
        self.alive = True
        self.char_type = char_type
        self.speed = speed
        self.ammo = ammo
        self.start_ammo = ammo
        self.shoot_cooldown = 0
        self.grenades = grenades
        self.health = 100
        self.max_health = self.health
        self.direction = 1
        self.vel_y = 0
        self.jump = False
        self.in_air = True
        self.flip = False
        self.animation_list = []
        self.frame_index = 0
        self.action = 0
        self.update_time = pygame.time.get_ticks()
```

The __init__ function in the Spidey class would set up the basic variables for the Spiderman and Venom characters. As displayed on the code above, all characters would of course be alive in the beginning, all the speed and other game physics are set up to be the same for all characters. Each character would be given Ammos, but only the Spiderman character would be able to use the Web-Grenade. The Max Health as well as health-ratio would also be assigned as a variable here to assign that all characters have health. The self.animation_list would help in giving the characters an animation whether it be idle or running, the animation would be made possible by using the Pygame Sprite animation.

```python
#Loading Images For Spiderman
animation_types = ['Idle', 'Run', 'Jump', 'Death']
for animation in animation_types:
    #reset temporary list of images
    temp_list = []
    #count number of files in the folder
    num_of_frames = len(os.listdir(f'img/{self.char_type}/{animation}'))
    for i in range(num_of_frames):
        img = pygame.image.load(f'img/{self.char_type}/{animation}/{i}.png').convert_alpha()
        img = pygame.transform.scale(img, (int(img.get_width() * scale), int(img.get_height() * scale)))
        temp_list.append(img)
    self.animation_list.append(temp_list)
```



As you can see on the above screenshots, the code would initiate an empty list which would then be appended by the moving image which is looped by the for loop. What the code does is that it recognises the file as a list ranging from 0 up until the max number of frames/images in the folder. The looped images using the for loop is similar to the concept of traditional animation or stop-motion video in which several images are run through quickly to create somewhat of a stop-motion animation effect. If the player moves left, the image would be the same but it is played with a mirror effect, so the image would look as if it is the other way around.

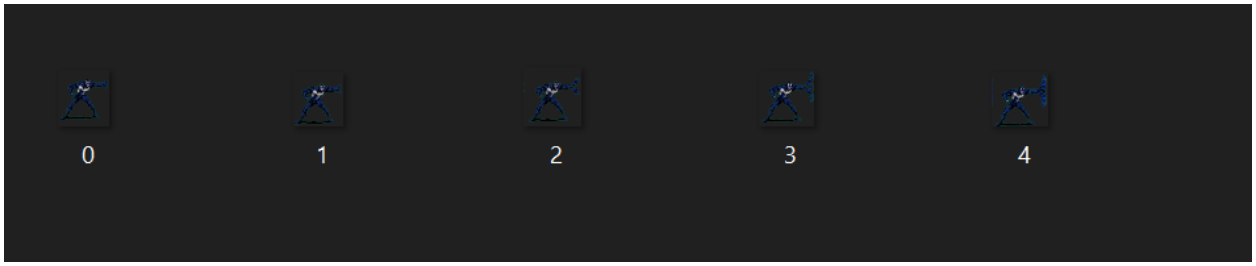
Venom/Enemies

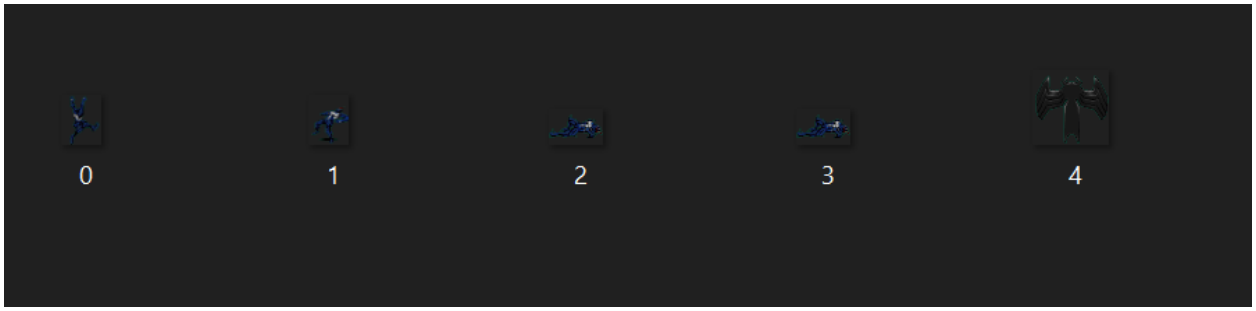
```
if self.alive and player.alive:
    if self.idling == False and random.randint(1, 200) == 1:
        self.update_action(0)#0: idle
        self.idling = True
        self.idling_counter = 50
    #check if the enemy is near the player
    if self.vision.colliderect(player.rect):
        #stop running and face the player
        self.update_action(0) # IDLE = 0
        #shoot
        self.venomshoot()
```

```
else:
    if self.idling == False:
        if self.direction == 1:
            venom_moving_right = True
        else:
            venom_moving_right = False
        ai_moving_left = not venom_moving_right
        self.move(ai_moving_left, venom_moving_right)
        self.update_action(1)#1: run
        self.move_counter += 1
        #updates the AI peripheral as Venom moves
        self.vision.center = (self.rect.centerx + 75 * self.direction, self.rect.centery)

        if self.move_counter > TILE_SIZE:
            self.direction *= -1
            self.move_counter *= -1
    else:
        self.idling_counter -= 1
        if self.idling_counter <= 0:
            self.idling = False
```

As for the main attributes of venom/the enemy, it's basically almost similar with the playable/character or Spiderman. The only difference is that Venom would be given command by the AI. For example, Venom would move across a given area using the run animation, and only performs the idle-ing stage for a very brief period, this would loop itself all the time until Venom encounters or gets close to Spider-Man or the player. As Venom encounters Spider-Man it would then go into an Idle stage and shoot Spider-Man/the player, the Shoot function would be discussed in the mechanics section. If the player goes out of sight, then Venom would of course run left and right as its movement is always updated using the self.move and self.update_action(1), which one means run.





As for the method of the animation it is pretty much the same as the one with Spider-Man/the playable character. It also uses a method of accessing the file using a for loop to create some sort of movement effect within the character's movement.

E. Mechanics - Web, Web-Bomb, Collision

Spiderman's Web

```
#PROGRAMS - SPIDERMAN's WEB and VENOM's SYMBIOTE WEB
class Web(pygame.sprite.Sprite):
    def __init__(self, x, y, direction):
        pygame.sprite.Sprite.__init__(self)
        self.speed = 10
        self.image = web_img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.direction = direction
```

As soon as the Web class is created, it is important to set the speed of the web, and to make sure that it is not to fast nor too slow, as for me, I found that setting the speed to 10 is the most ideal as you still can see the web moving but is not too fast until it is barely visible or too slow until it feels too laggy. After setting the speed just call-in the web image variable from the assets file. After that set the image position using the x and y axis, as well as getting the direction value of the character.

```
def update(self):
    #move bullet
    self.rect.x += (self.direction * self.speed) + screen_scroll
    #check if bullet has gone off screen
    if self.rect.right < 0 or self.rect.left > SCREEN_WIDTH:
        self.kill()
    #check for collision with level
    for tile in world.obstacle_list:
        if tile[1].colliderect(self.rect):
            self.kill()
    #check collision with characters
    for enemy in enemy_group:
        if pygame.sprite.spritecollide(enemy, bullet_group, False):
            if enemy.alive:
                enemy.health -= 25
                self.kill()
```

If the bullet collides with the tile in the level, it would initiate self.kill() function, meaning it would disappear. If the bullet has gone off screen, it would also initiate self.kill() function and disappear. If the bullet hits an enemy then enemy health would go down by 25%, and the bullet would also disappear using the self.kill() function.

Spiderman's Web Grenade

```
        #check for collision with level
        for tile in world.obstacle_list:
            #check collision with walls
            if tile[1].colliderect(self.rect.x + dx, self.rect.y, self.width, self.height):
                self.direction *= -1
                dx = self.direction * self.speed
            #check for collision in the y direction
            if tile[1].colliderect(self.rect.x, self.rect.y + dy, self.width, self.height):
                self.speed = 0
                #check if below the ground, i.e. thrown up
                if self.vel_y < 0:
                    self.vel_y = 0
                    dy = tile[1].bottom - self.rect.top
                #check if above the ground, i.e. falling
                elif self.vel_y >= 0:
                    self.vel_y = 0
                    dy = tile[1].top - self.rect.bottom
```

Similar to the properties of a grenade if the grenade is launched, then it would bounce into wall tiles, before it stops. As the grenade bounces the speed would also decrease. If the grenade has already hit the ground then the velocity would be 0, which means that the grenade would stop moving.

```
#countdown timer
self.timer -= 1
if self.timer <= 0:
    self.kill()
    grenade_fx.play()
    explosion = Explosion(self.rect.x, self.rect.y, 0.5)
    explosion_group.add(explosion)
    #anyone who is nearby the grenade will be damaged
    if abs(self.rect.centerx - player.rect.centerx) < TILE_SIZE * 2 and \
        abs(self.rect.centery - player.rect.centery) < TILE_SIZE * 2:
        player.health -= 50 #takes 2 grenade to die
    for enemy in enemy_group:
        if abs(self.rect.centerx - enemy.rect.centerx) < TILE_SIZE * 2 and \
            abs(self.rect.centery - enemy.rect.centery) < TILE_SIZE * 2:
            enemy.health -= 50 #takes 2 grenade to kill
```

When the timer of the grenade ends, the grenade would initiate the self.kill() function and disappear, but not only disappear it would give an explosion animation from the explosion class. Anyone who is two tiles close to the web-bomb explosion would die, including Spiderman as the playable character. Any character who is impacted by the blast would have its health reduced by 50%. It takes 2 web-bomb for Spiderman or Venom to die.

Explosion Animation

```
class Explosion(pygame.sprite.Sprite):
    def __init__(self, x, y, scale):
        pygame.sprite.Sprite.__init__(self)
        self.images = []
        for num in range(1, 6):
            img = pygame.image.load(f'img/explosion/exp{num}.png').convert_alpha()
            img = pygame.transform.scale(img, (int(img.get_width() * scale), int(img.get_height() * scale)))
            self.images.append(img)
        self.frame_index = 0
        self.image = self.images[self.frame_index]
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.counter = 0
```

The explosion animation is done by using the Pygame Sprite animation. It accesses the folder in which the explosion pictures are stored, then it runs through quickly through all the files in the folder. The images that are run through quickly would then create an animation which in here creates the explosion animation. This class would then be used in the Grenade class as the animation for when the Web-Bomb timer ends, and then the web bomb explodes.

Explosion Animation Folder:

Venom's Web

```
class VenomWeb(pygame.sprite.Sprite):
    def __init__(self, x, y, direction):
        pygame.sprite.Sprite.__init__(self)
        self.speed = 10
        self.image = venom_web_img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.direction = direction

    def update(self):
        #move bullet
        self.rect.x += (self.direction * self.speed) + screen_scroll
        #check if bullet has gone off screen
        if self.rect.right < 0 or self.rect.left > SCREEN_WIDTH:
            self.kill()
        #check for collision with level
        for tile in world.obstacle_list:
            if tile[1].colliderect(self.rect):
                self.kill()
```

For the __init__ and update method, it is similar to Spider Man's Web. The only difference in the __init__ function is the self.image, which here is the Venom web. The biggest difference could be found in the properties of the Bullet below.

```
#check collision with characters
if pygame.sprite.spritecollide(player, bullet_group, False):
    if player.alive:
        player.health -= 5
        self.kill()
```

As visible here, the collision of course has to be different with Spiderman's class. Here, if the Web collides with Spider-man the player's health would decrease by 5 and the bullet would then disappear.

Shoot Function

```
202     def shoot(self):
203         if self.shoot_cooldown == 0 and self.ammo > 0:
204             self.shoot_cooldown = 20
205             bullet = Web(self.rect.centerx + (0.75 * self.rect.size[0] * self.direction), self.rect.centery, self.direction)
206             bullet_group.add(bullet)
207             #reduce ammo
208             self.ammo -= 1
209             shot_fx.play()
210 
211 
212     def venomshoot(self):
213         if self.shoot_cooldown == 0 and self.ammo > 0:
214             self.shoot_cooldown = 20
215             bullet = VenomWeb(self.rect.centerx + (0.75 * self.rect.size[0] * self.direction), self.rect.centery, self.direction)
216             bullet_group.add(bullet)
217             #reduce ammo
218             self.ammo -= 1
219             shot_fx.play()
```

For both Venom and Spider-Man the Shoot function is pretty self-explanatory. The shoot(self) function is called upon for Spider-Man to shoot his webs and the venomshoot(self) is for Venom to shoot his Web. Each of Spider-Man and Venom has the same number of Web-Ammo which is 20. The ammo would decrease by one every time it is being shot with the self.ammo -= 1. As Venom and Spiderman shoot the shot_fx sound would always play. If the Web-Ammo is equal to 0, it would then stop firing as the ammo is finished.

F. Game Controls

```
for event in pygame.event.get():
    #quit game
    if event.type == pygame.QUIT:
        run = False
    #keyboard presses
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_a:
            moving_left = True
        if event.key == pygame.K_d:
            moving_right = True
        if event.key == pygame.K_SPACE:
            shoot = True
        if event.key == pygame.K_q:
            grenade = True
        if event.key == pygame.K_w and player.alive:
            player.jump = True
            jump_fx.play()
        if event.key == pygame.K_ESCAPE:
            run = False
```

Buttons are mapped using Pygame.event.get() function for the Python to get the keyboard commands. For events when the keyboard is pressed pygame.KEYDOWN is initiated.

CONTROL:

- **W** - JUMP
- **A** - MOVE LEFT
- **D** - MOVE RIGHT
- **SPACEBAR** - shoot webs
- **Q** - throw web-bomb
- **ESC** - exit program

```
#keyboard button released
if event.type == pygame.KEYUP:
    if event.key == pygame.K_a:
        moving_left = False
    if event.key == pygame.K_d:
        moving_right = False
    if event.key == pygame.K_SPACE:
        shoot = False
    if event.key == pygame.K_q:
        grenade = False
        grenade_thrown = False
```

As for when the control is released the command would not be initiated. For example, if A or D is released the player/Spider-Man would stop moving. If the SPACEBAR is released Spider-Man would stop shooting, and if Q is released, then a grenade would not be thrown.

G. Game-Flow

1. Initiate Main-Menu

2. Game Starts

    2. a. If Spider-Man dies restart level

    2. b. If Spider-man completes level go to level 2

    2. c. If Death Screen = Restart , Restart level 1

    2. d. If Death Screen = Exit, go to Main-Menu

3. Level 2 starts

    3. a. If Spider-Man dies restart level

    3. b. If Spider-man completes level, go to Main-Menu

    3. c. If Death Screen = Restart , Restart level 2

    3. d. If Death Screen = Exit, go to Main-Menu

References

Game Assets:

- Pinterest
- https://itch.io/game-assets/tag-8-bit
- Audio: done by myself

Modules:

- Pygame (https://www.pygame.org/download.shtm)
- OS
- CSV

Video Reference:

- https://youtu.be/YWN8GcmJ-jA
- https://www.youtube.com/watch?v=AY9MnQ4x3zk&t=83s
- https://youtu.be/1WxQt4-ic3s
- https://youtu.be/n4feYKKDncs
- https://www.youtube.com/watch?v=hDu8mcAlY4E
- https://youtu.be/UdsNBIzsmII
- https://youtu.be/M6e3_8LHc7A