Final Project Report

Algorithm and Programming Course

Vampire and Boy

BY:

Ellis Raputri

(2702298116)

Class L1AC

Computer Science Program

School Of Computing and Creative Arts

Bina Nusantara International University

Jakarta

2023

A. Background

As one of the requirements of the Algorithm and Programming class, the students were told to make something using Python as their final project. So, students can make anything for their final project, with some minimum requirements.

I was first confused about what should I make for this project. But, after several days of thinking and researching, I decided to make a game. The reason for this is that I heard that making a game is easier and takes a shorter time. There are also numerous tutorials on the internet.

Then, after deciding about making a game, I spent yet another few days contemplating what game I should make. After some time of thinking and reading references, I decided to make my own story game. In my opinion, a game with a story is more interesting and unique. Besides that, I also can incorporate my own written story in there. So, I can make my story "alive" with the game.

After that, I sought to search for assets and backgrounds, to gain inspiration for my story. And, I found some werewolf and vampire game assets. They inspired me to write a story about how a boy defends himself from a group of werewolves and a vampire. So, at last, I made a story game with some battle scenes, some cutscenes, and a finding object scene to convey my story.

B. Problem Identification

In this era, technology has been crucial in human's daily life. Humans rely on technology for various purposes, like cleaning, learning, and others. Technology can also be used for entertainment purposes. And, the game is one way of this entertainment. After a day of work, humans could relax by playing games.

Nowadays, there are various kinds of games, such as logic games, online board games, etc. But, a game without some story is like a soulless product. The game seems more static and has no meaning in it. So, I decided to make a game that convey a simple story. The moral of the story is also simple, i.e. the hero will always win over the villain. The simplicity of this story makes it easier to understand and more accessible to all ages.

C. Project Description

Vampire and Boy, a story-based game that consists of fighting games and a finding object game. This game consists of several states, specifically one main menu, six cutscenes, three game instructions, five battles, and one finding object game.

The main menu displays all the functions that the player can access when first starting the game. There is a “New Game” button to start a new game, a “Continue” button to continue to the current state of the game, and an “Exit” button to quit the game.

Then, there are the cutscenes. Basically, the cutscenes only show the text based on the story provided in each scene. But, besides that, some parts of the cutscene also have the icon of a character on it, to clarify which character that was saying the dialogue.

Next, we have the game instructions. These parts are the simplest states in this game. It only carries one command, the “Press Space” key. After pressing the key, these states will move to another state. And, the purpose of these is to provide clarity and explanation about how the game is.

We also have the battle scene. These are the main parts of this game. This state is a fighting game with a turn-based mechanism. So, on every single play, the player will start first, then followed by the enemy. To attack, the player needs to click on the enemy. The damage dealt will be a random number that is summed with the strength. Besides attacking, the player can also heal, by just clicking the potion button in the bottom panel. But the player can only heal three times. This is to ensure that the game will not last too long. Same as the player, the enemy can also attack and heal. The damage dealt to the player is also a random number that is summed with the enemy’s strength. While the healing in the enemy is quite different from the player. When the enemy hp drops down to below 50%, it will automatically choose to heal, not attack. These are the stats of each individual in the game:

Name	Max HP	Potion numbers	Strength
Boy	30	3	12
Black Wolf	15	0	3
White Wolf	15	1	4
Red Wolf	20	2	5
Man	30	3	8
Vampire	50	5	8

When the enemy dies, then the system will declare that the player won the fight and show the “next stage” button. The player can press it to proceed to the next state. However, when the player dies, it will be counted as a defeat and the player has to restart the battle.

Lastly, there is a finding object game. Like other finding object games, this game’s objective is to find the objects on the list. After finding all of them, the player can proceed to the next scene. There is also a hint button for the player. When clicking it, the player could see a red circle on one of the objects in the list.

D. Story Summary

A boy found himself waking up in a deep forest. He then searched for clues about what happened. Then, accidentally, he met a female vampire with a mysterious aura. The vampire offered the boy the information the boy wanted, but in exchange, the boy needed to find three werewolf hearts and a list of things. After some thought, the boy decided to find those items for the vampire.

He then fought some werewolves and managed to get three of said werewolf hearts. He was tired, but he still needed to find a list of things. When he was muddled about how to find the things, he spotted an unoccupied house not far from where he stood. He then entered the house without a second thought.

After entered the house, he met a prestigious man in the house. And, after explaining the things he wanted to find, the boy was attacked by the man without any particular reason. So, the boy fought back to defend himself.

After the battle, it was revealed that the vampire is the man's sister. The man hated the vampire, so he tried to fail her plans. Before becoming dust, the man warned the boy to not believe the vampire too much. The boy didn't react too much to that warning and proceeded to go upstairs to find the things on the list.

After he found the things, the boy met again with the vampire. The boy handed all the items to the vampire. The vampire laughed and told the boy that the truth was that the vampire had killed all of the villagers and the boy was just lucky not to be killed. The boy was furious after hearing this and challenged the vampire to a fight. The vampire laughed again and agreed with the boy. But before the fight started, the vampire transformed herself into a monster using the items the boy found for her.

After a tedious fight, the vampire dispersed into dust. And, the wind gradually flew all her remains away. The boy heaved a long sigh and decided to start his own new journey. He packed his things and started his journey.

E. Libraries Used

- **Pygame**

  Pygame is a module of Python that is designed to write games. Pygame has many functions to add and implement functionality to games. Pygame is highly portable and runs on every operating system. Pygame has been used by many people and downloaded millions of times. But Pygame is not a built-in library of Python. So, to use the Pygame library, it is mandatory to install the library first. To install it, we just need to type "pip install pygame" in the terminal and wait for a while. After the download process is done, then we can use the Pygame library.

- **Random**

  Random is a built-in Python library that deals with random numbers. It can generate random numbers, return a list of random numbers, return a random float number, and others. Basically, this library's function involves generating random numbers. In this project, this library is used to generate a random number of damages to the enemy.

- **Sys**

  Sys is also a built-in module in Python. This library provides functions and variables related to the Python runtime environment. It allows the operation of the interpreter and has functions and variables that interact with the interpreter. In this project, the sys module is used to stop the game (using the exit() function).

- **Math**

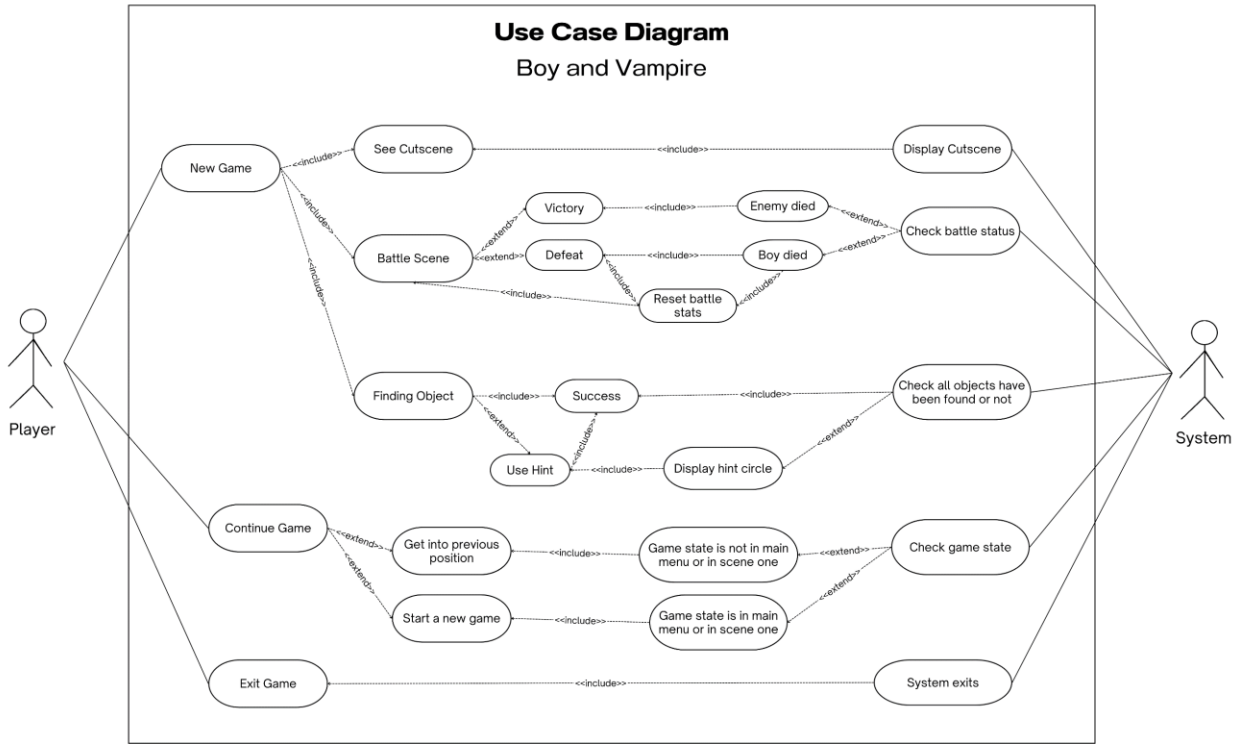
  Math is also a built-in module in Python. This library provides various functions to do mathematical operations. The math module also has some constants to be used. In this project, the math module is imported to do a ceiling division for the tiles in the main menu.

F. Important Files and Folders

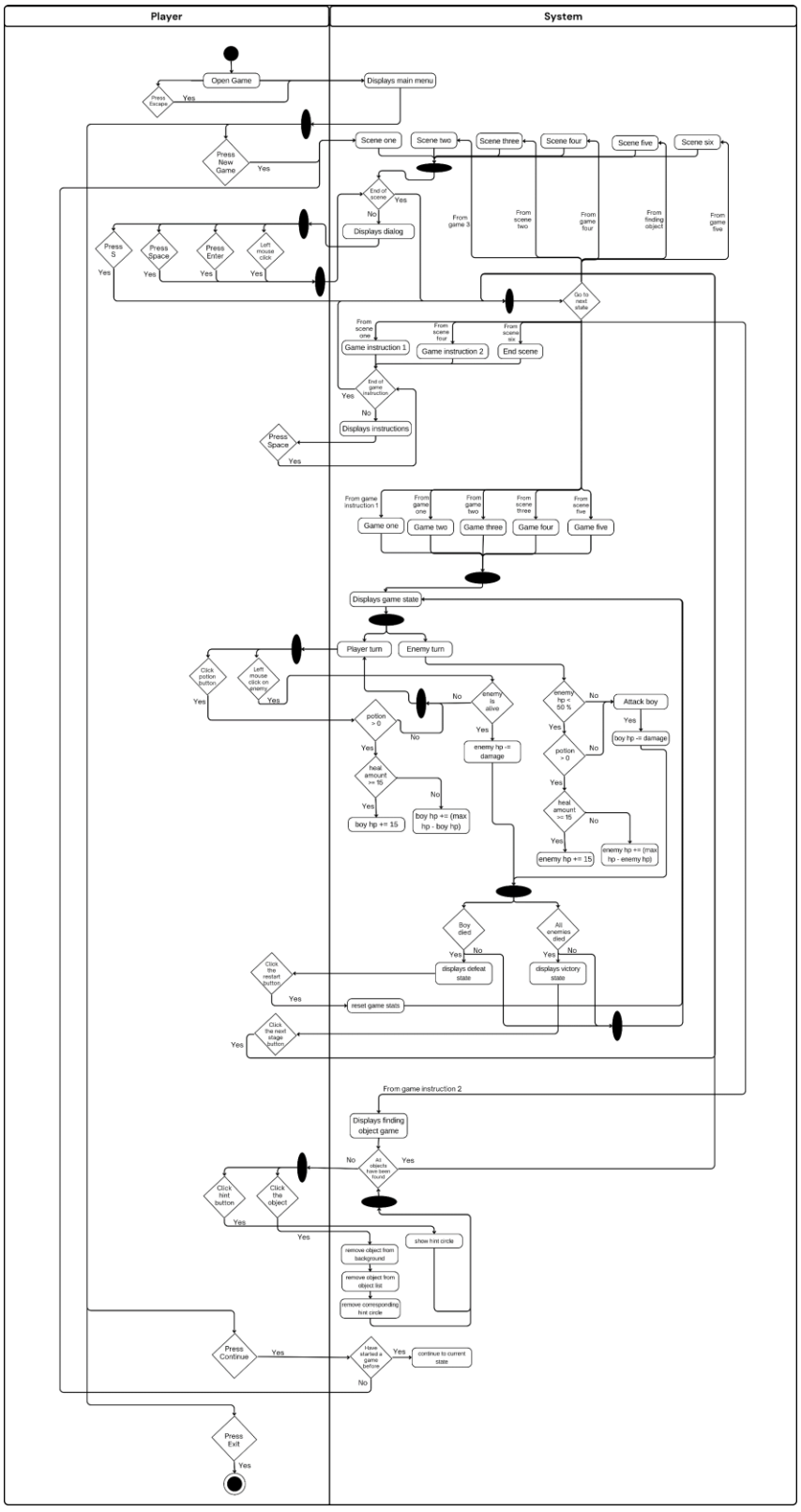
- button.py – contains the class Button to enable buttons
- continue_game.py - contains the class ContinueGame that enables the continue button
- damage_and_healthbar.py – contains class DamageText (to display the damage caused by each fighter) and class HealthBar (to display and update the health bar)
- fighter.py – contains class Fighter to specify each individual's animation and stats
- finding_object_thing.py – contains class Things (to determine and display the object in the background), class ThingsInList (to determine and display the object in the finding list), and class HintCircle (to determine and display each hint circle).
- game_function1.py – contains class GameFunctionsBattle (the algorithm and functions of the battle scene) and class GameFunctionsFind (the algorithm and functions of the finding object).
- game_function2.py – contains the function play_bgm (to play background music on each state), class MainMenu (run the main menu state), class Scene (run the cutscene state), class GameInstruction (run the game instructions), class Game (run the battle state), and class FindObject (run the finding object game).
- img_text_display.py – contains class Displaying (display images and text), class FadeTransition (to enable the transition to the next state), and draw_text_complete function (to display all the text when completing the finding object game)

- main.py – the main file running all the game functions
- running_text.py – contains the class DialogText that shows and controls the dialogue on each cutscene.
- transition.py – contains class Transition that enables and plays the transition
- 'report' folder – contains the project report and diagrams
- In the 'Assets' folder:
  - 'audio' folder – contains all the audio assets, including background music and sound effects.
  - 'font' folder – contains font used in the game
  - 'images' folder – contains all images, specifically background images, icons, things in finding objects, and all individual fighter's sprite images (boy, black wolf, white wolf, red wolf, man, and vampire).
  - 'story' folder – contains all stories in the cutscene

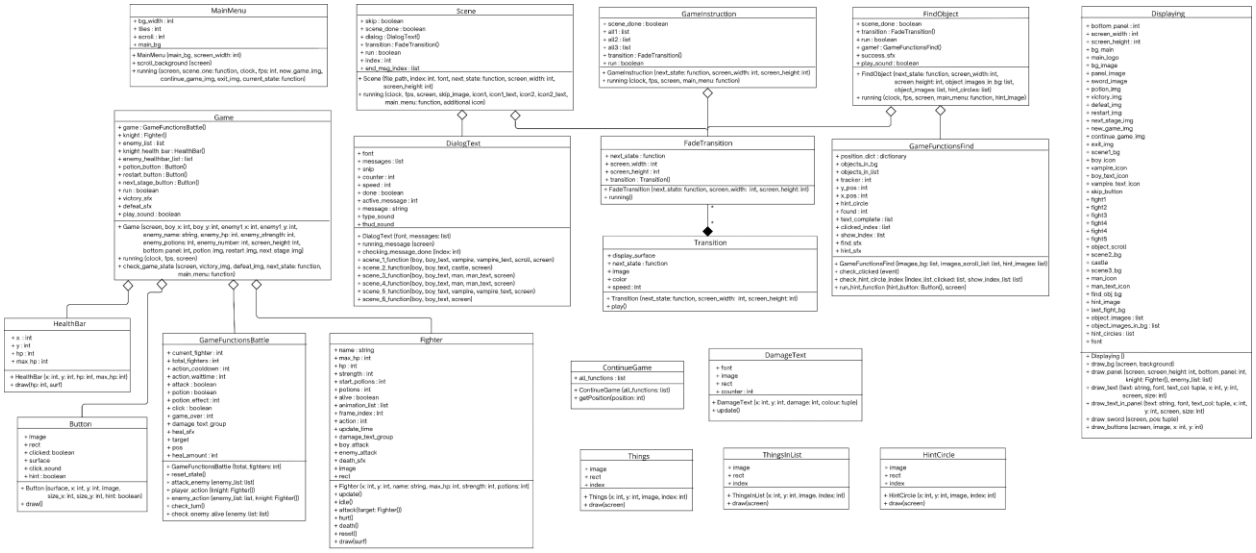
G. Use Case Diagram



H. Activity Diagram



I. Class Diagram



J. Essential Algorithms

1. button.py

- Class Button

```
import pygame

class Button():
    def __init__(self, surface,x,y,image,size_x,size_y, hint=False):
        self.image = pygame.transform.scale(image, (size_x, size_y))
        self.rect = self.image.get_rect()
        self.rect.center = (x,y)
        self.clicked = False
        self.surface = surface
        self.click_sound = pygame.mixer.Sound('Assets/audio/sfx/click.wav')
        self.hint = hint
```

The class Button initializes and manages all things regarding buttons. In the __init__() method, we declare variables, like image for the button, rect for the button's position, clicked for checking if the button has been clicked or not, surface for the location for displaying the button, click_sound for the sound effects when the button is clicked, and hint. The hint attribute is only used for the hint button, so the default value for it is False.

```
def draw(self):
    action = False
    pos = pygame.mouse.get_pos()

    #if mouse position collide with the button
    if self.rect.collidepoint(pos):
        #if mouse get press
        if pygame.mouse.get_pressed()[0] == 1 and self.clicked == False:
            action = True
            self.clicked = True
            if self.hint == False:
                self.click_sound.play()

    #if mouse doesn't get pressed
    if pygame.mouse.get_pressed()[0] == 0:
        self.clicked=False

    #draw button
    self.surface.blit(self.image, (self.rect.x, self.rect.y))

    #return whether the mouse being clicked or not
    return action
```

draw() method is used to display the button on the screen. Besides displaying, the draw() method also checks if the button is being clicked or not. When clicked, the draw() method will return a True value in the action variable, so we can use this value to determine if the button is being clicked or not. Also, there is the attached click_sound when clicking buttons. But, for the hint button, the click_sound will not play (because there is another sound effect attached for the hint function in the other file).

2. continue_game.py

- class ContinueGame

```
class ContinueGame():
    def __init__(self, all_functions):
        self.all_functions = all_functions

    def getPosition(self, position):
        return self.all_functions[position]
```

This class receives the list of all functions from the main file. Then, when the getPosition() method is called, it will return the corresponding function based on the position.

3. damage_and_healthbar.py

```
import pygame

#colors
red =(255,0,0)
green = (0,255,0)
```

These lines of code above are initialized to be used by classes below.

- class DamageText

```
class DamageText(pygame.sprite.Sprite):
    def __init__(self, x, y, damage, colour):
        pygame.sprite.Sprite.__init__(self)
        font =pygame.font.Font('Assets/font/Pixeltype.ttf',40)
        self.image = font.render(damage, True, colour)
        self.rect = self.image.get_rect()
        self.rect.center = (x,y)
        self.counter=0

    def update(self):
        #move damage text up
        self.rect.y -= 1
        #delete text after few seconds
        self.counter +=1
        if self.counter > 30:
            self.kill()
```

Class DamageText inherits from the pygame Sprite class. So, it can make a group and has several inherited functions, like add() and kill(). First, in the __init__() method, we initialize the font used for the text, image (the text will be displayed as an image), rect for the position, and counter (to determine when the damage text will disappear.

In the update() method, for every game loop, the damage text will move up a little by little. And also, for each game loop, the counter will increase. When the counter reaches 30, then the damage text will disappear. The kill() function here is needed to remove the text from the group, so it will not be displayed on the screen.

- Class HealthBar

```
class HealthBar():
    def __init__(self, x, y, hp, max_hp):
        self.x = x
        self.y = y
        self.hp = hp
        self.max_hp = max_hp

    def draw (self, hp, surf):
        #update hp as game progress
        self.hp = hp
        #calculate health ratio
        ratio = self.hp / self.max_hp

        #draw the red and green part of healthbar
        pygame.draw.rect(surf, red, (self.x, self.y, 150, 20))
        pygame.draw.rect(surf, green, (self.x, self.y, 150*ratio, 20))
```

Class HealthBar has only two functions, the __init__() method and the draw() method. __init__() method initializes the x and y for the health bar position, hp for the current hp, and max_hp for the maximum hp of an individual.

Meanwhile, the draw() method is used to update the hp as the game progresses. It accepts an hp argument to determine the current hp of each individual in each of the game loops. Then, it calculates the ratio of the hp by max_hp. Then, based on the ratio, it will draw the red and green part of the health bar. The red part always has a width of 150, while the green part changes according to the ratio (note: the green part is displayed above the red part). So, when an individual takes damage, the ratio will decrease from 1. This will make the green part have a smaller width and thus, some parts of the red part will show.

4. fighter.py

- class Fighter

```
import pygame
import random

class Fighter():
    def __init__(self, x, y, name, max_hp, strength, potions):
        self.name = name
        self.max_hp = max_hp
        self.hp = max_hp
        self.strength = strength
        self.start_potions = potions
        self.potions = potions
        self.alive = True
        self.animation_list = []
        self.frame_index = 0
        self.action = 0     # 0 is idle, 1 is attack, 2 is hurt, 3 is death
        self.update_time = pygame.time.get_ticks()
        self.damage_text_group = pygame.sprite.Group()

        #sound effects
        self.boy_attack = pygame.mixer.Sound("Assets/audio/sfx/boy_attack.wav")
        self.boy_attack.set_volume(0.8)
        self.enemy_attack = pygame.mixer.Sound("Assets/audio/sfx/enemy_attack.wav")
        self.death_sfx = pygame.mixer.Sound("Assets/audio/sfx/dead.wav")
        self.death_sfx.set_volume(0.8)
```

The __init__() method initializes some variables, like the name, max_hp, hp, strength, start_potions, potions, alive (to determine if the individual is alive or not), animation_list, frame_index (the index of the animation), action (determine which animation should be played), update_time (time update for the animation), damage_text_group, boy_attack (the sound effect for boy attacking), enemy_attack (sound effect for enemy attacking), and death_sfx.

```python
if self.name == 'Boy':
    #idle images
    temp_list = []
    for i in range(6):
        img = pygame.image.load(f"Assets/images/{self.name}/Idle/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4.5, img.get_height()*4.5))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #attack images
    temp_list = []
    for i in range(8):
        img = pygame.image.load(f"Assets/images/{self.name}/Attack/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4.5, img.get_height()*4.5))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #hurt images
    temp_list = []
    for i in range(4):
        img = pygame.image.load(f"Assets/images/{self.name}/Hurt/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4.5, img.get_height()*4.5))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #death images
    temp_list = []
    for i in range(12):
        img = pygame.image.load(f"Assets/images/{self.name}/Death/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4.5, img.get_height()*4.5))
        temp_list.append(img)
    self.animation_list.append(temp_list)
```

```python
elif self.name == 'Wolf1' or self.name =='Wolf2' or self.name =='Wolf3':
    #idle images
    temp_list = []
    for i in range(8):
        img = pygame.image.load(f"Assets/images/{self.name}/Idle/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*1.7, img.get_height()*1.7))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #attack images
    temp_list = []
    for i in range(6):
        img = pygame.image.load(f"Assets/images/{self.name}/Attack/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*1.7, img.get_height()*1.7))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #hurt images
    temp_list = []
    for i in range(2):
        img = pygame.image.load(f"Assets/images/{self.name}/Hurt/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*1.7, img.get_height()*1.7))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #death images
    temp_list = []
    for i in range(2):
        img = pygame.image.load(f"Assets/images/{self.name}/Death/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*1.7, img.get_height()*1.7))
        temp_list.append(img)
    self.animation_list.append(temp_list)
```

```
elif self.name == 'Man':
    #idle images
    temp_list = []
    for i in range(5):
        img = pygame.image.load(f"Assets/images/{self.name}/Idle/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*2, img.get_height()*2))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #attack images
    temp_list = []
    for i in range(5):
        img = pygame.image.load(f"Assets/images/{self.name}/Attack/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*2, img.get_height()*2))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #hurt images
    temp_list = []
    for i in range(2):
        img = pygame.image.load(f"Assets/images/{self.name}/Hurt/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*2, img.get_height()*2))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #death images
    temp_list = []
    for i in range(8):
        img = pygame.image.load(f"Assets/images/{self.name}/Death/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*2, img.get_height()*2))
        temp_list.append(img)
    self.animation_list.append(temp_list)
```

```
elif self.name == 'Vampire':
    #idle images
    temp_list = []
    for i in range(5):
        img = pygame.image.load(f"Assets/images/{self.name}/Idle/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4, img.get_height()*4))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #attack images
    temp_list = []
    for i in range(6):
        img = pygame.image.load(f"Assets/images/{self.name}/Attack/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4, img.get_height()*4))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #hurt images
    temp_list = []
    for i in range(2):
        img = pygame.image.load(f"Assets/images/{self.name}/Hurt/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4, img.get_height()*4))
        temp_list.append(img)
    self.animation_list.append(temp_list)

    #death images
    temp_list = []
    for i in range(8):
        img = pygame.image.load(f"Assets/images/{self.name}/Death/{i}.png").convert_alpha()
        img = pygame.transform.scale(img, (img.get_width()*4, img.get_height()*4))
        temp_list.append(img)
    self.animation_list.append(temp_list)
```

Then, the __init__() method also contains these codes. These codes are used to store the images based on the fighter's name. So, using the fighter's name, it will automatically load the corresponding images for each individual. Each individual fighter also has an animation_list that contains four lists in it. The animation_list[0] is for idle images, animation_list[1] is for attack images, animation_list[2] is for hurt images, and animation_list[3] is for death images.

```
self.image = self.animation_list[self.action][self.frame_index]
self.rect = self.image.get_rect()
self.rect.center = (x,y)
```

Then, we have these variables too. The image variable is to display the corresponding image for each game loop. If the action is 0, then it will display the idle images according to the frame_index. The frame_index value will be updated in each of the game loops (will be explained further below). Lastly, we have the rect variable to specify the position of the fighter.

```
def update(self):
    animation_cooldown = 100
    self.image = self.animation_list[self.action][self.frame_index]

    #check if enough time has passed since last update
    if (pygame.time.get_ticks() - self.update_time > animation_cooldown):
        self.update_time = pygame.time.get_ticks()
        self.frame_index += 1

    if self.frame_index >= len(self.animation_list[self.action]):
        if self.action ==3:
            self.frame_index = len(self.animation_list[self.action])-1
        else:
            self.idle()
```

The update() method is the primary method for the animation. The animation_cooldown ensures the animation doesn't show too fast, so there is some time pause between an animation state and another. If the time used for the update has already exceeded the animation_cooldown, then the time will be updated and the frame_index also increase by 1. So, when the next animation plays, then the image that is being shown is the next. After all the images in the corresponding animation_list are shown, then there are two possibilities. If the action is 3 (or, we just displayed the death animation), then the frame_index stays in the last index of the animation list, so the

fighter stays dead. However, if the action variable is not 3 (or not death animation), then the fighter goes back to the idle state.

```
#idle animation
def idle(self):
    self.action =0
    self.frame_index =0
    self.update_time = pygame.time.get_ticks()
```

These codes specify that the fighter is in idle animation. We also restart the frame_index and update_time to redo every animation state, meaning that after a idle animation being played, then other animation can be played again.

```
def attack (self, target):
    #deal damage to enemy
    rand = random.randint(-5, 5)
    damage = abs(self.strength + rand)
    target.hp -= damage
    target.hurt()

    #target die or not
    if target.hp <1:
        target.hp = 0
        target.alive = False
        target.death()

    #attack animation
    self.action =1
    self.frame_index =0
    self.update_time = pygame.time.get_ticks()

    #sfx for attack animations
    if self.name == 'Boy':
        self.boy_attack.play()
    else:
        self.enemy_attack.play()

    return damage
```

The attack() method specifies things for attacking a target. The damage is calculated by using a random number from -5 to 5, then adding it to the fighter's strength. After calculating the damage, the target hp is decreased by the damage and the target also displays hurt animation. If the target hp drops below 1, then the target is dead and displays the death animation. Then, there are also the reset animation variables and also, play sound effects based on who is attacking.

```
#hurt animation
def hurt(self):
    self.action =2
    self.frame_index =0
    self.update_time = pygame.time.get_ticks()

#death animation
def death(self):
    self.action =3
    self.frame_index =0
    self.update_time = pygame.time.get_ticks()
    self.death_sfx.play()
```

Then, we also have the hurt and death animation here.

```
#restart
def reset(self):
    self.alive = True
    self.potions = self.start_potions
    self.hp = self.max_hp
    self.frame_index =0
    self.action = 0
    self.update_time = pygame.time.get_ticks()

#animation draw
def draw(self, surf):
    surf.blit(self.image, self.rect)
```

Next, the reset() method. This method is used to reset all of the fighter stats when the player clicks on the restart button. So, all the fighters will have their stats reset and start the battle again. Lastly, there's the draw() method to draw the images on the surface. The images being drawn are updated every game loop based on the update() method earlier.

5. finding_object_thing.py

- class Things

```
import pygame

#images in the background
class Things(pygame.sprite.Sprite):
    def __init__(self, x, y, image, index):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.index = index

    def draw(self, screen):
        screen.blit(self.image, self.rect)
```

Class Things has the __init__() method that contains the object's image, the image rect for arranging the position, and the index. This class also has the draw() method that draw the image into the screen. This class is used for the things in the background.

- class ThingsInList

```
#images in the list
class ThingsInList(pygame.sprite.Sprite):
    def __init__(self, x, y, image, index):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.index = index

    def draw(self, screen):
        screen.blit(self.image, self.rect)
```

Class ThingsInList has the __init__() method that contains the object's image, the image rect for arranging the position, and the index. This class also has the draw() method that draw the image into the screen. This class is used for the things in the finding list.

- class HintCircle

```
#hint circle
class HintCircle(pygame.sprite.Sprite):
    def __init__(self, x, y, image, index):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.index = index

    def draw(self, screen):
        screen.blit(self.image, self.rect)
```

Class HintCircle has the __init__() method that contains the object's image, the image rect for arranging the position, and the index. This class also has the draw() method that draw the image into the screen. This class is used for the hint circles in each of the things in background.

6. game_function1.py

Before the classes declaration, we first need to import all the needed function and declare needed global variables.

```
import pygame
from damage_and_healthbar import DamageText
from finding_object_thing import Things, ThingsInList, HintCircle

#color
red = (255,0,0)
green = (0,255,0)
```

- class GameFunctionsBattle

```
class GameFunctionsBattle():
    def __init__(self, total_fighters):
        #game variable
        self.current_fighter = 1    #1 for boy turn, 2 for first enemy, 3 for second enemy
        self.total_fighters = total_fighters
        self.action_cooldown = 0
        self.action_waittime = 90
        self.attack = False
        self.potion = False
        self.potion_effect =15
        self.click = False
        self.game_over = 0      #0:game still run, 1:boy wins, -1:enemy win
        self.damage_text_group = pygame.sprite.Group()
        self.heal_sfx = pygame.mixer.Sound('Assets/audio/sfx/heal.wav')

    def reset_state(self):
        self.attack = False
        self.potion = False
        self.target = None
        pygame.mouse.set_visible(True)
        self.pos = pygame.mouse.get_pos()
```

The __init__() method has several variables used for the battle. The current_fighter is to determine which turn is it now. Every game always starts with the boy's turn first. Then, there are the total of the fighters, action_cooldown (wait for some time before the fighter can do another action), action_waittime (the limit time of the action cooldown), attack (check if the fighter is attacking or not), potion (check if the fighter is healing or not), potion_effect, click (check click), game_over (0: game still run, 1: boy wins, -1: enemy win), damage_text_group, and heal_sfx.

Meanwhile, the reset_state() method is to reset the game state on every action in a battle. For example, after attacking an enemy, then the attack is set to false again (so that the boy doesn't keep attacking the enemy), the potion is set to false again (so that the boy doesn't keep healing), the target is None (so the player can choose another enemy to attack), and we renew the pos variable to track the mouse position.

```
def attack_enemy(self, enemy_list):
    for count, enemy in enumerate(enemy_list):
        #attack enemy that being clicked
        if enemy.rect.collidepoint(self.pos):
            if self.click == True and enemy.alive==True:
                self.attack = True
                self.target = enemy_list[count]
```

attack_enemy() method is to track the enemy that is being clicked by the player. If the enemy is being clicked and the enemy is still alive, then the target of the attack is being set as the enemy in a particular index in the enemy list.

```
def player_action(self, knight):
    if knight.alive == True:
        #boy turn
        if self.current_fighter == 1:
            self.action_cooldown +=1
            if self.action_cooldown >= self.action_waittime:
                if self.attack ==True and self.target != None:
                    damage = knight.attack(self.target)

                    #create damage text
                    damage_text = DamageText(self.target.rect.centerx, self.target.rect.centery-40, str(damage), red)
                    self.damage_text_group.add(damage_text)

                    #go to next turn
                    self.current_fighter += 1
                    self.action_cooldown=0

                if self.potion == True:
                    #if there are still potions left
                    if knight.potions >0:
                        #heal with 15 hp
                        if knight.max_hp > knight.hp + self.potion_effect:
                            self.heal_amount = self.potion_effect

                        #if knight damage is lower than 15, then only heal till the healthbar full
                        else:
                            self.heal_amount = knight.max_hp - knight.hp

                        #add hp based on heal amount and reduce the potions
                        knight.hp += self.heal_amount
                        knight.potions -=1

                        #display the green text showing that the boy is being healed
                        damage_text = DamageText(knight.rect.centerx, knight.rect.centery-40, str(self.heal_amount), green)
                        self.damage_text_group.add(damage_text)

                        #go to next turn and play sfx
                        self.current_fighter += 1
                        self.action_cooldown=0
                        self.heal_sfx.play()
    else:
        self.game_over = -1
```

The player_action() method specifies things for every player action. First, we need to check if the player is still alive or not. If not, then it will automatically be game over and the enemy wins. If the player is still alive and the current turn is the boy's turn, then the action_cooldown will increase until a specific amount of time in action_waittime. After exceeding the action_waittime, check if the player is attacking or healing. If the

player is attacking, then calculate the damage text and create the red damage text based on the damage text. After that, go to the next turn. If the player chooses to heal, then check if there are still potions left, and then the boy heals based on his hp. If his damage is lower than 15, then only heal until max hp. Otherwise, add 15 to the boy's hp. After healing, then show the green text and go to the next turn.

```
def enemy_action(self, enemy_list, knight):
    for count,enemy in enumerate(enemy_list):
        #enemy 1 = current fighter 2, enemy 2 = current fighter 3
        if self.current_fighter == 2 + count:
            if enemy.alive ==True:
                self.action_cooldown +=1
                if self.action_cooldown >= self.action_waittime:
                    #check if enemy need heal
                    if(enemy.hp / enemy.max_hp < 0.5 and enemy.potions>0):
                        if enemy.max_hp > enemy.hp + self.potion_effect:
                            self.heal_amount = self.potion_effect
                        else:
                            self.heal_amount = enemy.max_hp - enemy.hp

                        #heal and reduce potions
                        enemy.hp += self.heal_amount
                        enemy.potions -=1
                        #show green text that indicates healing
                        damage_text = DamageText(enemy.rect.centerx, enemy.rect.centery -40, str(self.heal_amount), green)
                        self.damage_text_group.add(damage_text)

                        #go to next turn and play sfx
                        self.current_fighter += 1
                        self.action_cooldown=0
                        self.heal_sfx.play()

                    #attack
                    else:
                        damage = enemy.attack(knight)

                        #create damage text
                        damage_text = DamageText(knight.rect.centerx, knight.rect.centery -40, str(damage), red)
                        self.damage_text_group.add(damage_text)
                        #go to next turn
                        self.current_fighter += 1
                        self.action_cooldown = 0
            else:
                self.current_fighter += 1
```

enemy_action() method specifies things for the enemy's action. For every enemy in the enemy list, if the enemy is alive, then it can attack or heal, like the player. The healing and attack mechanism is the same. The only thing that is different is that the enemy heals based on their hp ratio. When their hp ratio drops below 0.5, then it will automatically heal. Also, for each game loop, this function will go through the enemy list, so when there is a dead enemy and it is the dead enemy's turn, it will automatically skip its turn.

```
def check_turn(self):
    #if all fighter have had turn, then reset
    if self.current_fighter > self.total_fighters:
        self.current_fighter = 1

def check_enemy_alive(self, enemy_list):
    self.alive_enemies =0
    for enemy in enemy_list:
        if enemy.alive == True:
            self.alive_enemies+=1
    if self.alive_enemies == 0:
        self.game_over = 1
```

check_turn() method is to check if all fighters have had their turn. If yes, then we reset the turn to 1, which means it is the boy's turn again. Next, the check_enemy_alive() method is used to check if all enemies are still alive or not. If there is no enemy that is still alive, then game_over becomes 1, which means it's the player's win.

- class GameFunctionsFind

```
class GameFunctionsFind():
    def __init__(self, images_bg, images_scroll_list, hint_images):
        #dealing with objects in the background
        self.position_dict = {1: (187,100), 2:(660, 180), 3:(60,40), 4:(430,400), 5:(370,380),
                              6:(600,540), 7:(20,610), 8:(430, 80), 9:(360,600)}
        self.objects_in_bg = pygame.sprite.Group()
        for count,object_image in enumerate(images_bg):
            image = Things(self.position_dict[count+1][0], self.position_dict[count+1][1], object_image, count)
            self.objects_in_bg.add(image)

        #dealing with objects in the scroll list
        self.objects_in_list = pygame.sprite.Group()
        self.tracker = 0     #to track the index of the images
        self.y_pos = 190
        for i in range(3):
            self.x_pos = 750
            #for every three image, enter to a new line (have a new y-position)
            for j in range(3):
                image = ThingsInList(self.x_pos, self.y_pos, images_scroll_list[self.tracker], self.tracker)
                self.objects_in_list.add(image)
                self.x_pos += 100
                self.tracker += 1
            self.y_pos += 120
```

The __init__() method is used to initialize things for the finding object game. First of all, the position_dict is used to store the position of each item in the background. Then, we add all the background items in the group of objects_in_bg. Next, we have the object_in_list group. It is the same with the previous background items. The only difference between them is the position of it. The position of the objects in the list is set by the x_pos and y_pos. There is also the tracker attribute to track the index image.

```
#hint circle
self.hint_circle = pygame.sprite.Group()
for count,object_image in enumerate(hint_images):
    image = HintCircle(self.position_dict[count+1][0], self.position_dict[count+1][1], object_image, count)
    self.hint_circle.add(image)

#keep track of number of object being found
self.found = 0

#if all things have been found
self.text_complete = ["NICE!", " ", "Press space", "to continue"]

#hint function variables
self.clicked_index = []
self.show_index = [0,1,2,3,4,5,6,7,8]

#sfx
self.find_sfx = pygame.mixer.Sound("Assets/audio/sfx/findobj.wav")
self.find_sfx.set_volume(0.7)
self.hint_sfx = pygame.mixer.Sound("Assets/audio/sfx/hint.wav")
self.hint_sfx.set_volume(0.8)
```

There is also the group of hint circles, which consists of each hint circle with the same location of each background item. The found attribute is to track how many objects are already being found. There is also the text_complete (will be shown when the game is complete), the clicked_index (store image index that has been clicked), show_index (determine which hint circle should be shown), find_sfx (sound effect when an object is found), and hint_sfx (sound effect when using hint).

```
def check_clicked(self, event):
    #check if the object in the background is being clicked or not
    for obj_in_bg in self.objects_in_bg:
        if obj_in_bg.rect.collidepoint(event.pos):
            obj_in_bg.kill()

            #if object in the background is clicked, then remove the same object from the scroll list
            for obj_in_list in self.objects_in_list:
                if obj_in_bg.index == obj_in_list.index:
                    obj_in_list.kill()

            #if object in background is clicked, then remove the hint circle attached to that object
            for circle in self.hint_circle:
                if obj_in_bg.index == circle.index:
                    circle.kill()

            #increase number of object being found
            self.found += 1

            #play sound effects
            self.find_sfx.play()

            self.clicked_index.append(obj_in_bg.index)
```

check_clicked() method is the main method for the finding object game. This method is quite simple. So, when the object in the background has been clicked, then remove the corresponding object from the finding list and also, remove the hint circle

attached to that object. Then, increase the number of found attributes and play the sound effects. After that, we also have to append the index of that object to the list of clicked_index, so we know that the index has been clicked before.

```
#to check if the shown index has been clicked or not.
#if it has been clicked, then we dont need to show the hint for it
def check_hint_circle_index(self, index_list_clicked, show_index_list):
    for clicked in index_list_clicked:
        for show in show_index_list:
            #if the index being shown has been clicked, then remove the index
            if show == clicked:
                show_index_list.remove(show)
    return show_index_list
```

check_hint_circle_index() method is used to control the show_index list. In this method, we compare the clicked_index and the show_index. If there is an index in the show_index that has been clicked, then remove that index.

```
#show the circle that the item hasn't been clicked
def run_hint_function(self, hint_button, screen):
    self.show_index = self.check_hint_circle_index(self.clicked_index, self.show_index)
    if hint_button.draw():
        #draw the circle to the screen based on the show index
        for circle in self.hint_circle:
            if self.show_index != []:
                if circle.index == self.show_index[0]:
                    circle.draw(screen)

                    #play sfx
                    self.hint_sfx.play()
```

run_hint_function() method is the core of the hint button. In this method, first, we check the clicked_index and show_index, and then we check if the hint button is being clicked or not. If the hint button is being clicked, then show the hint circle based on the first show index and play the hint sound effect.

7. game_function2.py

First, we import all of the needed modules

```
import pygame
from button import Button
import sys
import math
import img_text_display
from running_text import DialogText
from img_text_display import FadeTransition
from game_function1 import GameFunctionsBattle, GameFunctionsFind
from damage_and_healthbar import HealthBar
from fighter import Fighter
```

- play_bgm()

```
def play_bgm(music_now):
    #unload previous music
    if pygame.mixer.music.get_busy():
        pygame.mixer.music.fadeout(500)
        pygame.mixer.music.unload()

    #load the new corresponding music
    pygame.mixer.music.load(music_now)
    pygame.mixer.music.set_volume(0.4)
    pygame.mixer.music.play(loops = -1)
```

This function is used to play background music based on what state is showing now. This function unloads the previous music and play the new current corresponding music.

- class MainMenu

```
class MainMenu():
    def __init__(self, main_bg, screen_width):
        self.bg_width = main_bg.get_width()
        self.tiles = math.ceil(screen_width/self.bg_width) + 1
        self.scroll = 0
        self.main_bg = main_bg

    def scroll_background(self, screen):
        #draw the background image
        for i in range (0, self.tiles):
            screen.blit(self.main_bg, (i * self.bg_width + self.scroll, 0))

        #scroll background
        self.scroll -= 2
        if abs(self.scroll) > self.bg_width:
            self.scroll = 0
```

The __init__() method has some attributes, like bg_width (the background width), tiles (how many tiles that showing on the screen now), scroll, and main_bg (the background image). The scroll_background() method enables the endless scrolling of

the background in the main menu. The position of the background image shifted a bit to the left every game loop. And, if all of the background is already scrolled to the left, then redraw the background on the right side of the screen.

```
def running(self, screen, scene_one, clock, fps, new_game_img, continue_game_img, exit_img, current_state):
    pygame.mouse.set_visible(True)

    #initializing each button
    new_game = Button(screen, 520, 320, new_game_img, 300, 100)
    continue_game = Button(screen, 520, 440, continue_game_img, 300, 100)
    exit_game = Button(screen, 520,560, exit_img, 300, 100)

    #each button click
    if new_game.draw():
        scene_one()

    elif continue_game.draw():
        current_state()

    elif exit_game.draw():
        pygame.quit()
        sys.exit()

    #event checker
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    pygame.display.update()
    clock.tick(fps)
```

running() method enables the main menu to run. running() method creates each button in the main menu screen and determines each button click. If the new game button that being clicked, then it will start from the first cutscene. If the continue game button is clicked, it will run the current state of the game. If the exit game button is clicked, it will quit the game. There is also the event checker to check if the user wants to quit the game or not and a clock tick to set up the frame rate.

- class Scene

```
class Scene():
    def __init__(self, file_path, index, font, next_state, screen_width, screen_height):
        self.skip = False
        self.scene_done = False
        myfile=open(file_path,'rt')
        all=myfile.readlines()

        self.dialog = DialogText(font, all)
        self.transition = FadeTransition(next_state, screen_width, screen_height)

        self.run = True
        self.index = index
        self.end_msg_index = [18, 12, 16, 14, 23, 13]
```

The __init__() method of this class contains several attributes, such as skip (to check if the player skips or not), scene_done (to check if the cutscene is done or not), dialogue (the dialogue for the cutscene), transition, run (for the game loop), index (the index of the cutscene), and end_msg_index (the last index of each cutscene).

```
def running(self, clock, fps, screen, skip_image, icon1, icon1_text, icon2, icon2_text, main_menu, additional_icon):
    #frame rate
    clock.tick(fps)

    #run the message
    self.dialog.running_message(screen)

    #show character in corresponding part of story
    if self.index == 1:
        self.dialog.scene_1_function(icon1, icon1_text, icon2, icon2_text, additional_icon, screen)
    elif self.index == 2:
        self.dialog.scene_2_function(icon1, icon1_text, additional_icon, screen)
    elif self.index == 3:
        self.dialog.scene_3_function(icon1, icon1_text, icon2, icon2_text, screen)
    elif self.index ==4:
        self.dialog.scene_4_function(icon1, icon1_text, icon2, icon2_text, screen)
    elif self.index ==5:
        self.dialog.scene_5_function(icon1, icon1_text, icon2, icon2_text, screen)
    elif self.index == 6:
        self.dialog.scene_6_function(icon1, icon1_text, screen)

    #draw skip button
    skip_button = Button(screen, 820, 50, skip_image, 330, 50)
    skip_button.draw()

    #when user skips or user have done the scene, then run transition
    if self.skip or self.scene_done:
        self.transition.running()

    #event checker
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.run = False
            sys.exit()

        if event.type == pygame.KEYDOWN:
            #press enter or space to move on to next dialog
            if event.key == pygame.K_RETURN or event.key == pygame.K_SPACE:
                self.dialog.checking_message_done(self.index)

                #checking what index number each dialog is done
                if(self.dialog.active_message == self.end_msg_index[self.index-1]):
                    self.scene_done = True
                    self.dialog.type_sound.stop()      #stop the typing sfx

            #press escape to go to main menu
            if event.key == pygame.K_ESCAPE:
                main_menu()

            #press s to skip
            if event.key == pygame.K_s:
                self.skip=True
                self.dialog.type_sound.stop()

        #click to move on to next dialog
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                self.dialog.checking_message_done(self.index)
                if(self.dialog.active_message == self.end_msg_index[self.index-1]):
                    self.scene_done = True
                    self.dialog.type_sound.stop()

    pygame.display.update()
```

The running() method is the method to run the cutscene. It runs the dialogue message by calling using the running_message() from the class DialogText. It also passes the corresponding object for each cutscene to the class DialogText. This method

also draws the skip button. And, there is several event checkers in this method. First of all, if the player quits, then it will exit the game. Then, if the player presses ESC, the player will go back to the main menu. Then, when the player presses enter or presses space or left mouse click, the dialogue will be played. If the dialogue already reaches the last index, then it will set scene_done as True. Also, the player can skip the scene by pressing S. By pressing S, the skip attribute will be set as True. And, when the scene_done is True or skip is True, then the transition to the next stage will be played.

- class GameInstruction

```
class GameInstruction():
    def __init__(self, next_state, screen_width, screen_height):
        self.scene_done = False
        self.all1=["Game Instructions:", "1. Click on the enemies to attack them.", "2. Click on the potion button to heal.",
            "3. Choose between attack or heal in each turn.", "4. Be careful and good luck!", " ", "Press space to continue"]
        self.all2 = ["Game Instructions:", "Find the object based on the scroll.","  ", "Press space to continue"]
        self.all3 = ["Thank you for playing!", "", "Press space to return to the main menu"]
        self.transition = FadeTransition(next_state, screen_width, screen_height)
        self.run = True
```

The __init__() method in this class has several attributes, such as scene_done (to see if the scene is done or not), all1 (the messages for the game instructions 1), all2 (the messages for the game instructions 2), all3 (the messages for the end scene), transition, and run (for the game loop).

```
def running(self, clock, fps, main_menu):
    #frame rate
    clock.tick(fps)

    #if scene done, run transition to next state
    if self.scene_done:
        self.transition.running()

    #event checker
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.run = False
            sys.exit()

        if event.type == pygame.KEYDOWN:
            #press space to move on
            if event.key == pygame.K_SPACE:
                self.scene_done = True

            #press escape to main menu
            if event.key == pygame.K_ESCAPE:
                main_menu()

    pygame.display.update()
```

The running() method is used to run the game instructions state. There is only a simple event checker in here, that is press quit to exit, press ESC to the main menu, and

press space to end this state. When the player presses space, then the transition to the next state will be immediately played.

- class Game

```
class Game():
    def __init__(self, screen, boy_x, boy_y, enemy1_x, enemy1_y, enemy_name, enemy_hp, enemy_strength, enemy_potions, enemy_number,
                 screen_height, bottom_panel, potion_img, restart_img, next_stage_img):
        #game function
        self.game = GameFunctionsBattle(enemy_number + 1)

        #create boy
        self.knight = Fighter(boy_x, boy_y, 'Boy', 30, 12, 3)

        #create enemy
        self.enemy_list = []
        for i in range (enemy_number):
            enemy = Fighter(enemy1_x, enemy1_y, enemy_name, enemy_hp, enemy_strength, enemy_potions)
            self.enemy_list.append(enemy)
            enemy1_x += 150

        #create health bar for boy
        self.knight_health_bar = HealthBar(70, screen_height - bottom_panel + 40, self.knight.hp, self.knight.max_hp)

        #create health bar for enemy
        self.enemy_healthbar_list = []
        for j in range (enemy_number):
            enemy_healthbar = HealthBar(550, (screen_height - bottom_panel + 40 + 60*j), self.enemy_list[j].hp, self.enemy_list[j].max_hp)
            self.enemy_healthbar_list.append(enemy_healthbar)

        #create button
        self.potion_button = Button(screen, 100, screen_height-bottom_panel+105, potion_img, 64,64)
        self.restart_button = Button(screen, 500, 110, restart_img, 120, 30)
        self.next_stage_button = Button(screen, 500, 122, next_stage_img, 160, 40)

        #game loop
        self.run = True

        #sound effects
        self.victory_sfx = pygame.mixer.Sound("Assets/audio/sfx/victory.wav")
        self.victory_sfx.set_volume(0.7)
        self.defeat_sfx = pygame.mixer.Sound("Assets/audio/sfx/defeat.wav")
        self.defeat_sfx.set_volume(0.7)
        self.play_sound = False     #to ensure the sound only play once
```

The __init__() method here initializes the game functions for the battle scene. Then, it also creates the fighters (the boy and the enemies) and the fighters' health bars. It also creates the potion button, restart button, and next stage button. It also initializes the run attribute for the game loop and sound effects. To ensure victory and defeat sound effects only play once, there is also the play_sound attribute.

```
def running(self, clock, fps, screen):
    #frame rate
    clock.tick(fps)

    #draw healthbar for boy
    self.knight_health_bar.draw(self.knight.hp, screen)

    #draw healthbar for enemy
    index = 0
    for enemy_healthbar in self.enemy_healthbar_list:
        enemy_healthbar.draw(self.enemy_list[index].hp, screen)
        index += 1

    #draw knight
    self.knight.draw(screen)
    self.knight.update()

    #draw enemy
    for enemy in self.enemy_list:
        enemy.draw(screen)
        enemy.update()

    #draw damage text
    self.game.damage_text_group.update()
    self.game.damage_text_group.draw(screen)
```

The running() method is used to run some part of the game. It draws the health bar and fighters to the screen. It also manages the damage text.

```
def check_game_state(self,screen, victory_img, defeat_img, next_state, main_menu):
    #displaying potion button
    #button click
    if self.potion_button.draw():
        self.game.potion = True

    #each player action
    if self.game.game_over == 0:
        #player action
        self.game.player_action(self.knight)

        #enemy action
        self.game.enemy_action(self.enemy_list, self.knight)

        #check all action
        self.game.check_turn()

    #check if all enemies are death
    self.game.check_enemy_alive(self.enemy_list)

    #check if game is over
    if self.game.game_over != 0:
        #if player wins
        if self.game.game_over == 1:
            #draw victory image and show next stage button
            victory_rect = victory_img.get_rect(center=(500,60))
            screen.blit(victory_img, victory_rect)
            if self.next_stage_button.draw():
                next_state()

            #play victory music
            if self.play_sound == False:
                self.victory_sfx.play()
                self.play_sound = True

        #if enemy wins
        elif self.game.game_over == -1:
            #play defeat music
            if self.play_sound == False:
                self.defeat_sfx.play()
                self.play_sound = True

            #draw defeat image, show restart button, and reset all the stats
            defeat_rect = defeat_img.get_rect(center=(500,60))
            screen.blit(defeat_img, defeat_rect)
            if self.restart_button.draw():
                self.knight.reset()
                for enemy in self.enemy_list:
                    enemy.reset()
                self.game.current_fighter =1
                self.game.action_cooldown = 0
                self.game.game_over = 0
                self.play_sound = False

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.run = False
            sys.exit()

        #track left mouse click
        if event.type == pygame.MOUSEBUTTONDOWN:
            self.game.click = True
        else:
            self.game.click = False

        if event.type == pygame.KEYDOWN:
            #press escape to go to main menu
            if event.key == pygame.K_ESCAPE:
                main_menu()

    pygame.display.update()
```

The check_game_state() method is used to check the game state after every game

loop. It checks and displays the potion button. It also calls the player_action(), enemy_action(), check_turn(), and check_enemy_alive() to run the game. Then, it will check if the game is over or not. If the player wins, then it will play the victory sound and display the next stage button. If the enemy wins, it will play the defeated sound and display the restart button. When the restart button is clicked, then all the game stats are reset, so the game will start over again. As for the event checker, there is the press quit to exit, press ESC to the main menu, and the checker for left mouse click. If the mouse click is true, then it will set the click attribute in the class GameFunctionsBattle as True.

- class FindObject

```
class FindObject():
    def __init__(self, next_state, screen_width, screen_height, object_images_in_bg, object_images, hint_circles):
        self.scene_done = False
        self.transition = FadeTransition(next_state, screen_width, screen_height)
        self.run = True
        self.gamef = GameFunctionsFind(object_images_in_bg, object_images, hint_circles)

        #sound effects
        self.success_sfx = pygame.mixer.Sound("Assets/audio/sfx/findobj_success.wav")
        self.success_sfx.set_volume(0.4)
        self.play_sound = False     #ensure the sfx only play once
```

The __init__() method in this class initializes the scene_done (to know if the scene is done or not), transition, run (for the game loop), gamef (the game function), success_sfx (sound effects for completing the game), and play_sound (to ensure the sound effects only play once).

```
def running(self, clock, fps, screen, main_menu, hint_image):
    #frame rate
    clock.tick(fps)

    #draw object in background and object in scroll list
    self.gamef.objects_in_bg.draw(screen)
    self.gamef.objects_in_list.draw(screen)

    #when all of the objects have been found
    text_y = 240
    if self.gamef.found == 9:
        #display congrats text
        for text in self.gamef.text_complete:
            img_text_display.draw_text_complete(text, 'black', 850, text_y, screen, 60)
            text_y += 40

            #play success sound
            if self.play_sound == False:
                self.success_sfx.play()
                self.play_sound = True

    #enabling hint function
    hint_button= Button(screen, 920, 520, hint_image, 140, 60, True)
    self.gamef.run_hint_function(hint_button, screen)

    #run transition when scene done
    if self.scene_done:
        self.transition.running()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.run = False
            sys.exit()
        if event.type == pygame.KEYDOWN:
            #press space to move on
            if self.gamef.found == 9:
                if event.key == pygame.K_SPACE:
                    self.scene_done = True

            #press escape to go to main menu
            if event.key == pygame.K_ESCAPE:
                main_menu()

        #check each click
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:
                self.gamef.check_clicked(event)

    pygame.display.update()
```

The running() method is used to run the finding object game. It draws the object in the screen and also, determines if the objects have been found or not. When all objects have been found, then it will display the complete text and play the sound effects. It also calls on the run_hint_function() to enable the hint function. And, for the event checker for this state, there are press quit to exit, press space to set scene_done as True, and the mouse click checker. If the scene_done is True, then it will run the transition to the next state. The mouse click checker is used to see whether the objects in the background have been clicked or not.

8. img_text_display.py

First, we import needed modules and declare needed global variables.

```
import pygame
from transition import Transition

#color
red= (255,0,0)
```

- class Displaying

```
class Displaying():
    def __init__(self):
        #screen width and height
        self.bottom_panel = 150
        self.screen_width = 1000
        self.screen_height = 500 + self.bottom_panel

        #images
        self.bg_main = pygame.image.load("Assets/images/background/main_menu.png").convert_alpha()
        self.bg_main = pygame.transform.scale(self.bg_main, (self.bg_main.get_width()*0.5, self.bg_main.get_height()*1.25))

        self.main_logo = pygame.image.load("Assets/images/icon/main_menu_logo.png").convert_alpha()
        self.main_logo = pygame.transform.scale(self.main_logo, (self.main_logo.get_width()*0.5, self.main_logo.get_height()*0.5))

        self.bg_image = pygame.image.load("Assets/images/background/background.png").convert_alpha()
        self.bg_image = pygame.transform.scale(self.bg_image, (self.bg_image.get_width()*1.25, self.bg_image.get_height()*1.25))

        self.panel_image = pygame.image.load("Assets/images/icon/panel.png").convert_alpha()
        self.panel_image = pygame.transform.scale(self.panel_image, (self.panel_image.get_width()*1.25, self.panel_image.get_height()))

        self.sword_image = pygame.image.load("Assets/images/icon/sword.png").convert_alpha()
        self.potion_img = pygame.image.load("Assets/images/icon/potion.png").convert_alpha()
        self.victory_img = pygame.image.load("Assets/images/icon/victory.png").convert_alpha()
        self.defeat_img = pygame.image.load("Assets/images/icon/defeat.png").convert_alpha()
        self.restart_img = pygame.image.load("Assets/images/icon/restart.png").convert_alpha()
        self.next_stage_img = pygame.image.load("Assets/images/icon/next_stage.png").convert_alpha()

        self.new_game_img = pygame.image.load("Assets/images/icon/button_main_1.png").convert_alpha()
        self.new_game_img = pygame.transform.scale(self.new_game_img, (self.new_game_img.get_width()*0.45, self.new_game_img.get_height()*0.45))

        self.continue_game_img = pygame.image.load("Assets/images/icon/button_main_2.png").convert_alpha()
        self.continue_game_img = pygame.transform.scale(self.continue_game_img, (self.continue_game_img.get_width()*0.45, self.continue_game_img.get_height()*0.45))

        self.exit_img = pygame.image.load("Assets/images/icon/button_main_3.png").convert_alpha()
        self.exit_img = pygame.transform.scale(self.exit_img, (self.exit_img.get_width()*0.45, self.exit_img.get_height()*0.45))

    self.scene1_bg = pygame.image.load("Assets/images/background/scene_1.png").convert_alpha()
    self.scene1_bg = pygame.transform.scale(self.scene1_bg, (self.scene1_bg.get_width()*1.5, self.scene1_bg.get_height()*1.5))

    self.boy_icon = pygame.image.load("Assets/images/icon/boy_icon.png").convert_alpha()
    self.boy_icon = pygame.transform.scale(self.boy_icon, (self.boy_icon.get_width()*5.5, self.boy_icon.get_height()*5.5))

    self.vampire_icon = pygame.image.load("Assets/images/icon/vampire_icon.png").convert_alpha()
    self.vampire_icon = pygame.transform.scale(self.vampire_icon, (self.vampire_icon.get_width()*3.5, self.vampire_icon.get_height()*3.5))

    self.boy_text_icon = pygame.image.load("Assets/images/icon/boy_text_icon.png").convert_alpha()
    self.boy_text_icon = pygame.transform.scale(self.boy_text_icon, (self.boy_text_icon.get_width()*0.35, self.boy_text_icon.get_height()*0.35))

    self.vampire_text_icon = pygame.image.load("Assets/images/icon/vampire_text_icon.png").convert_alpha()
    self.vampire_text_icon = pygame.transform.scale(self.vampire_text_icon, (self.vampire_text_icon.get_width()*0.35, self.vampire_text_icon.get_height()*0.35))

    self.skip_button = pygame.image.load("Assets/images/icon/skip_button.png").convert_alpha()

    self.fight1 = pygame.image.load("Assets/images/icon/fight1.png").convert_alpha()
    self.fight1 = pygame.transform.scale(self.fight1, (self.fight1.get_width()*0.35, self.fight1.get_height()*0.35))

    self.fight2 = pygame.image.load("Assets/images/icon/fight2.png").convert_alpha()
    self.fight2 = pygame.transform.scale(self.fight2, (self.fight2.get_width()*0.3, self.fight2.get_height()*0.3))

    self.fight3 = pygame.image.load("Assets/images/icon/fight3.png").convert_alpha()
    self.fight3 = pygame.transform.scale(self.fight3, (self.fight3.get_width()*0.3, self.fight3.get_height()*0.3))

    self.fight4 = pygame.image.load("Assets/images/icon/fight4.png").convert_alpha()
    self.fight4 = pygame.transform.scale(self.fight4, (self.fight4.get_width()*0.3, self.fight4.get_height()*0.3))

    self.fight5 = pygame.image.load("Assets/images/icon/fight5.png").convert_alpha()
    self.fight5 = pygame.transform.scale(self.fight5, (self.fight5.get_width()*0.3, self.fight5.get_height()*0.3))
```

```
        self.object_scroll = pygame.image.load("Assets/images/icon/object_scroll.png").convert_alpha()

        self.scene2_bg = pygame.image.load("Assets/images/background/scene_2.png").convert_alpha()
        self.scene2_bg = pygame.transform.scale(self.scene2_bg, (self.scene2_bg.get_width()*0.7, self.scene2_bg.get_height()*0.46))

        self.castle = pygame.image.load("Assets/images/icon/castle.png").convert_alpha()

        self.scene3_bg = pygame.image.load("Assets/images/background/scene_3.png").convert_alpha()
        self.scene3_bg = pygame.transform.scale(self.scene3_bg, (self.scene3_bg.get_width(), self.scene3_bg.get_height()*0.8))

        self.man_icon = pygame.image.load("Assets/images/icon/man_icon.png").convert_alpha()
        self.man_icon = pygame.transform.scale(self.man_icon, (self.man_icon.get_width()*3, self.man_icon.get_height()*3))

        self.man_text_icon = pygame.image.load("Assets/images/icon/man_text_icon.png").convert_alpha()
        self.man_text_icon = pygame.transform.scale(self.man_text_icon, (self.man_text_icon.get_width()*0.35, self.man_text_icon.get_height()*0.35))

        self.find_obj_bg = pygame.image.load("Assets/images/background/find_obj_bg.jpg").convert_alpha()
        self.find_obj_bg = pygame.transform.scale(self.find_obj_bg, (self.find_obj_bg.get_width()*1.1, self.find_obj_bg.get_height()*1.25))

        self.hint_image = pygame.image.load("Assets/images/icon/hint.png").convert_alpha()

        #object images in the finding object list
        self.object_images=[]
        for i in range(1,10):
            image = pygame.image.load(f"Assets/images/Objects/{i}.png").convert_alpha()
            self.object_images.append(image)

        #object images in the finding object background
        self.object_images_in_bg = []
        for j in range(1,10):
            bg_object = pygame.image.load(f"Assets/images/Objects/dark{j}.png").convert_alpha()
            self.object_images_in_bg.append(bg_object)
        #circle hint in finding object
        self.hint_circles = []
        for k in range(1,10):
            circle = pygame.image.load(f"Assets/images/Objects/circle.png").convert_alpha()
            circle = pygame.transform.scale(circle, (circle.get_width()*0.2, circle.get_height()*0.2))
            self.hint_circles.append(circle)

        self.last_fight_bg = pygame.image.load("Assets/images/background/last_fight.png").convert_alpha()
        self.last_fight_bg = pygame.transform.scale(self.last_fight_bg, (self.last_fight_bg.get_width()*0.4, self.last_fight_bg.get_height()*0.33))


        #text
        self.font =pygame.font.Font('Assets/font/Pixeltype.ttf',40)
```

The __init__() method contains the attributes that load most of the image assets needed in this game and load the default font in this game.

```
#draw functions
def draw_bg(self,screen,background):
    screen.blit(background,(0,0))

def draw_panel(self,screen, screen_height, bottom_panel, knight, enemy_list):
    #draw panel rectangle
    screen.blit(self.panel_image, (0, screen_height - bottom_panel))
    #show knight stats
    self.draw_text_in_panel(f'{knight.name} HP: {knight.hp}', self.font, red, 70, screen_height-bottom_panel+25, screen, 40)

    #show enemy stats
    for count, i in enumerate(enemy_list):
        if i.name == "Wolf1":
            self.draw_text_in_panel(f'Black Wolf {count+1} HP: {i.hp}', self.font, red, 550, screen_height-bottom_panel+25 + count*60, screen, 40)
        elif i.name == "Wolf2":
            self.draw_text_in_panel(f'White Wolf {count+1} HP: {i.hp}', self.font, red, 550, screen_height-bottom_panel+25 + count*60, screen, 40)
        elif i.name == "Wolf3":
            self.draw_text_in_panel(f'Red Wolf {count+1} HP: {i.hp}', self.font, red, 550, screen_height-bottom_panel+25 + count*60, screen, 40)
        elif i.name == "Man":
            self.draw_text_in_panel(f'Man HP: {i.hp}', self.font, red, 550, screen_height-bottom_panel+25 + count*60, screen, 40)
        elif i.name == "Vampire":
            self.draw_text_in_panel(f'Vampire HP: {i.hp}', self.font, red, 550, screen_height-bottom_panel+25 + count*60, screen, 40)
```

```
def draw_text (self, text, font, text_col, x, y, screen, size):
    font =pygame.font.Font('Assets/font/Pixeltype.ttf',size)
    img = font.render(text, True, text_col)
    img_rect= img.get_rect(center = (x,y))
    screen.blit(img, img_rect)

def draw_text_in_panel (self, text, font, text_col, x, y, screen, size):
    font =pygame.font.Font('Assets/font/Pixeltype.ttf',size)
    img = font.render(text, True, text_col)
    img_rect= img.get_rect(midleft = (x,y))
    screen.blit(img, img_rect)

def draw_sword (self, screen, pos):
    screen.blit(self.sword_image, pos)

def draw_buttons(self, screen, image, x, y):
    image_rect = image.get_rect(center=(x,y))
    screen.blit(image, image_rect)
```

Then, we have the draw_bg() method to draw the background to the screen, the draw_panel() method to draw the bottom panel in the battle scene, the draw_text() method to draw text on the screen, the draw_text_in_panel() method to draw text in the bottom panel, draw_sword to draw sword icon, and draw_buttons() to draw buttons or icons to the screen. In the draw_panel() method, it draws the panel rectangle, boy stats, and every enemy stats. Every kind of enemy has a different name to be drawn in the panel.

- class FadeTransition

```
class FadeTransition():
    def __init__(self, next_state, screen_width, screen_height):
        self.next_state = next_state
        self.screen_width = screen_width
        self.screen_height = screen_height
        self.transition = Transition(self.next_state, self.screen_width, self.screen_height)

    def running(self):
        self.transition.play()
```

The __init__() method of this class has the next_state attribute to determine the next state, screen_width, screen_height, and transition. This class also has the running() method to call in the play() method in the Transition class.

- draw_text_complete()

```
def draw_text_complete (text, text_col, x, y, screen, size):
    font =pygame.font.Font('Assets/font/Pixeltype.ttf',size)
    img = font.render(text, True, text_col)
    img_rect= img.get_rect(center = (x,y))
    screen.blit(img, img_rect)
```

This function is used to draw text when completing the finding object game. It has the same code as the draw_text() method before. But this function resides outside of the class, so we don't need to declare an object before using this particular function.

9. main.py

```
import pygame
import game_function2
from game_function2 import Scene, GameInstruction, Game, FindObject, MainMenu
from img_text_display import Displaying
from continue_game import ContinueGame

pygame.init()

#color
red = (255,0,0)
white = (255,255,255)

#setting the frame rate
clock = pygame.time.Clock()
fps = 60

#display
screen = pygame.display.set_mode((1000, 650))
displays = Displaying()

#the game title
pygame.display.set_caption('Vampire and Boy')

#to track current state
current_state_list = []
```

Before all of the functions, we import all of the modules needed and declare some global variables. Then, we set up the screen display and the game title. After that, we also need the current_state_list to track the current state of the game.

- main_menu()

```
def main_menu():
    #if player press continue before starting a new game, automatically starts into new game
    current_state = continue_game.getPosition(0)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/main_menu.wav')

    main_menu_obj = MainMenu(displays.bg_main, displays.screen_width)

    #game loop
    while True:
        #draw background
        main_menu_obj.scroll_background(screen)

        #draw logo
        displays.draw_buttons(screen, displays.main_logo, 520, 150)

        #if player press continue after starting a new game
        if len(current_state_list) > 1:
            main_menu_obj.running(screen, scene_one, clock, fps, displays.new_game_img, displays.continue_game_img, displays.exit_img, current_state_list[-2])

        #if player press continue before starting a new game
        else:
            main_menu_obj.running(screen, scene_one, clock, fps, displays.new_game_img, displays.continue_game_img, displays.exit_img, current_state_list[-1])
```

The main_menu() function consists of several things. First, it tracks its location by sending out the index of 0 to the ContinueGame class. Then, it also appends the current state of the function to the current_state_list. It also plays the main menu music. Then, it draws the background and logo. Also, it manages the "continue" function. If the length of current_state_list is less than or equal to 1, then it will send the variable of current_state_list[-1], because there is only one state in the list. But if the length of the current_state_list is more than 1, then it will send the variable of current_state_list[-2]. This is because every time the user presses ESC to go back to the main menu, then the current_state_list will have the index 0 appended as the last item in the list. So, if we use the current_state_list[-1] here, then the continue button will not work (because it will automatically continue to the index 0 state).

- scene_one()

```
def scene_one():
    #track position
    current_state = continue_game.getPosition(1)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/scene.wav')

    scene_1 = Scene("Assets/story/dialog1_text.txt", 1, displays.font, game_instruction, displays.screen_width, displays.screen_height)

    #first typing sound
    scene_1.dialog.type_sound.play()

    while scene_1.run:
        #draw background and black panel
        displays.draw_bg(screen, displays.scene1_bg)
        pygame.draw.rect(screen, 'black', [0, 450, 1000, 300])

        #dialog functions and event key
        scene_1.running(clock, fps, screen, displays.skip_button, displays.boy_icon, displays.boy_text_icon, displays.vampire_icon, displays.vampire_text_icon, main_menu, displays.object_scroll)
```

In this function, the current_state is obtained by using the getPosition() method and then, we append the current_state to the list. Then, we also play the scene music and

typing sound. We also draw the background and black panel. Lastly, we also run the scene by passing the required parameters for the first scene.

- game_instruction()

```
def game_instruction():
    #track position
    current_state = continue_game.getPosition(2)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/main_menu.wav')

    game_instruction_1 = GameInstruction(game_one, displays.screen_width, displays.screen_height)

    while game_instruction_1.run:
        #draw background
        pygame.draw.rect(screen, 'black', [0, 0, 1000, 650])

        #draw text
        height = 120
        for i in game_instruction_1.all1:
            displays.draw_text(i, displays.font, white, 500, height, screen, 60)
            height+=60

        #event checker and functions
        game_instruction_1.running(clock, fps, main_menu)
```

The game_instruction() function tracks the position of the current state and then, appends it to the list. Then, it also plays the background music and draws the background image and text. And, last but not least, it runs the functions for game instruction.

- game_one()

```
def game_one():
    #track position
    current_state = continue_game.getPosition(3)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/battle.wav')

    game_1 = Game(screen, 270, 300, 650, 310, 'Wolf1', 15, 3, 0, 2, displays.screen_height, displays.bottom_panel, displays.potion_img, displays.restart_img, displays.next_stage_img)

    while game_1.run:
        #draw background and panel
        displays.draw_bg(screen, displays.bg_image)
        displays.draw_panel(screen, displays.screen_height, displays.bottom_panel, game_1.knight, game_1.enemy_list)
        displays.draw_buttons(screen, displays.fight1, 870, 50)

        game_1.running(clock, fps, screen)

        #control player action
        #reset action variable
        game_1.game.reset_state()

        #attacking enemy
        game_1.game.attack_enemy(game_1.enemy_list)
        for enemy in game_1.enemy_list:
            if enemy.rect.collidepoint(game_1.game.pos):
                #hide mouse
                pygame.mouse.set_visible(False)
                #show sword
                displays.draw_sword(screen, game_1.game.pos)

        #show number potion remaining
        displays.draw_text(str(game_1.knight.potions), displays.font, red, 145, displays.screen_height -displays.bottom_panel+85, screen, 40)

        #check if game over or not and check click
        game_1.check_game_state(screen, displays.victory_img, displays.defeat_img, game_two, main_menu)
```

The game_one() function first tracks the position of the current state and appends it to the current_state_list. Then, it plays the background music, draws the background

and panel, and draws the remaining potion numbers. Then, it controls the game flow. It also hides the mouse and displays a sword when the mouse collides with the enemy. Lastly, it checks if the game is over or not.

- game_two()

```
def game_two():
    #track position
    current_state = continue_game.getPosition(4)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/battle.wav')

    game_2 = Game(screen, 270, 300, 650, 310, 'Wolf2', 15, 4, 1, 2, displays.screen_height, displays.bottom_panel, displays.potion_img, displays.restart_img, displays.next_stage_img)

    while game_2.run:
        #draw background and panel
        displays.draw_bg(screen, displays.bg_image)
        displays.draw_panel(screen, displays.screen_height, displays.bottom_panel, game_2.knight, game_2.enemy_list)
        displays.draw_buttons(screen, displays.fight2, 870, 50)

        game_2.running(clock, fps, screen)

        #control player action
        #reset action variable
        game_2.game.reset_state()

        #attacking enemy
        game_2.game.attack_enemy(game_2.enemy_list)
        for enemy in game_2.enemy_list:
            if enemy.rect.collidepoint(game_2.game.pos):
                #hide mouse
                pygame.mouse.set_visible(False)
                #show sword
                displays.draw_sword(screen, game_2.game.pos)

        #show number potion remaining
        displays.draw_text(str(game_2.knight.potions), displays.font, red, 145, displays.screen_height -displays.bottom_panel+85, screen, 40)

        #check if game over or not and check click
        game_2.check_game_state(screen, displays.victory_img, displays.defeat_img, game_three, main_menu)
```

The game_two() function has the similar code with the game_one(). The difference between them is that the index position of the current_state variable.

- game_three()

```
def game_three():
    #track position
    current_state = continue_game.getPosition(5)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/battle.wav')

    game_3 = Game(screen, 270, 300, 650, 310, 'Wolf3', 20, 5, 2, 2, displays.screen_height, displays.bottom_panel, displays.potion_img, displays.restart_img, displays.next_stage_img)

    while game_3.run:
        #draw background and panel
        displays.draw_bg(screen, displays.bg_image)
        displays.draw_panel(screen, displays.screen_height, displays.bottom_panel, game_3.knight, game_3.enemy_list)
        displays.draw_buttons(screen, displays.fight3, 870, 50)

        game_3.running(clock, fps, screen)

        #control player action
        #reset action variable
        game_3.game.reset_state()

        #attacking enemy
        game_3.game.attack_enemy(game_3.enemy_list)
        for enemy in game_3.enemy_list:
            if enemy.rect.collidepoint(game_3.game.pos):
                #hide mouse
                pygame.mouse.set_visible(False)
                #show sword
                displays.draw_sword(screen, game_3.game.pos)

        #show number potion remaining
        displays.draw_text(str(game_3.knight.potions), displays.font, red, 145, displays.screen_height -displays.bottom_panel+85, screen, 40)

        #check if game over or not, check click
        game_3.check_game_state(screen, displays.victory_img, displays.defeat_img, scene_two, main_menu)
```

The game_three() function has the similar code with the game_one(). The difference between them is that the index position of the current_state variable.

- scene_two()

```
def scene_two():
    #track position
    current_state = continue_game.getPosition(6)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/scene.wav')

    scene_2 = Scene("Assets/story/dialog2_text.txt", 2, displays.font, scene_three, displays.screen_width, displays.screen_height)

    #first typing sound
    scene_2.dialog.type_sound.play()

    while scene_2.run:
        #draw background and black panel
        displays.draw_bg(screen, displays.scene2_bg)
        pygame.draw.rect(screen, 'black', [0, 450, 1000, 300])

        #dialog functions and event key
        scene_2.running(clock, fps, screen, displays.skip_button, displays.boy_icon, displays.boy_text_icon, "", "",main_menu, displays.castle)
```

The scene_two() function has a similar code to the scene_one(). The difference between them is that the index position of the current_state variable. Scene two also doesn't need the icon2 and icon2_text. So, it passes "“ for those parameters.

- scene_three()

```
def scene_three():
    #track position
    current_state = continue_game.getPosition(7)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/scene.wav')

    scene_3 = Scene("Assets/story/dialog3_text.txt", 3, displays.font, game_four, displays.screen_width, displays.screen_height)

    #first typing sound
    scene_3.dialog.type_sound.play()

    while scene_3.run:
        #draw background and black panel
        displays.draw_bg(screen, displays.scene3_bg)
        pygame.draw.rect(screen, 'black', [0, 450, 1000, 300])

        #dialog functions and event key
        scene_3.running(clock, fps, screen, displays.skip_button, displays.boy_icon, displays.boy_text_icon, displays.man_icon, displays.man_text_icon,main_menu, "")
```

The scene_three() function has a similar code to the scene_one(). The difference between them is that the index position of the current_state variable. Also, unlike scene one and scene two, scene three doesn't need any additional icons, so it passes the additional icon parameter as “”.

- game_four()

```
def game_four():
    #track position
    current_state = continue_game.getPosition(8)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/battle.wav')

    game_4 = Game(screen, 240, 320, 700, 320, 'Man', 30, 8, 3, 1, displays.screen_height, displays.bottom_panel, displays.potion_img, displays.restart_img, displays.next_stage_img)

    while game_4.run:
        #draw background and panel
        displays.draw_bg(screen, displays.scene3_bg)
        displays.draw_panel(screen, displays.screen_height, displays.bottom_panel, game_4.knight, game_4.enemy_list)
        displays.draw_buttons(screen, displays.fight4, 870, 50)

        game_4.running(clock, fps, screen)

        #control player action
        #reset action variable
        game_4.game.reset_state()

        #attacking enemy
        game_4.game.attack_enemy(game_4.enemy_list)
        for enemy in game_4.enemy_list:
            if enemy.rect.collidepoint(game_4.game.pos):
                #hide mouse
                pygame.mouse.set_visible(False)
                #show sword
                displays.draw_sword(screen, game_4.game.pos)

        #show number potion remaining
        displays.draw_text(str(game_4.knight.potions), displays.font, red, 145, displays.screen_height -displays.bottom_panel+85, screen, 40)

        #check if game over or not, check click
        game_4.check_game_state(screen, displays.victory_img, displays.defeat_img, scene_four, main_menu)
```

The game_four() function has the similar code with the game_one(). The difference between them is that the index position of the current_state variable.

- scene_four()

```
def scene_four():
    #track position
    current_state = continue_game.getPosition(9)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/scene.wav')

    scene_4 = Scene("Assets/story/dialog4_text.txt", 4, displays.font, game_instruction2, displays.screen_width, displays.screen_height)

    #first typing sound
    scene_4.dialog.type_sound.play()

    while scene_4.run:
        #draw background and black panel
        displays.draw_bg(screen, displays.scene3_bg)
        pygame.draw.rect(screen, 'black', [0, 450, 1000, 300])

        #dialog functions and event key
        scene_4.running(clock, fps, screen, displays.skip_button, displays.boy_icon, displays.boy_text_icon, displays.man_icon, displays.man_text_icon,main_menu, "")
```

The scene_four() function has a similar code to the scene_one(). The difference between them is that the index position of the current_state variable. Also, unlike scene one and scene two, scene four doesn't need any additional icons, so it passes the additional icon parameter as "".

- game_instruction2()

```
def game_instruction2():
    #track position
    current_state = continue_game.getPosition(10)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/main_menu.wav')

    game_instruction_2 = GameInstruction(find_obj, displays.screen_width, displays.screen_height)

    while game_instruction_2.run:
        #draw background
        pygame.draw.rect(screen, 'black', [0, 0, 1000, 650])

        #draw text
        height = 220
        for i in game_instruction_2.all2:
            displays.draw_text(i, displays.font, white, 500, height, screen, 60)
            height+=60

        #event checker and functions
        game_instruction_2.running(clock, fps, main_menu)
```

The game_instruction2() function has the similar code with the game_instruction(). The difference between them is that the index position of the current_state variable.

- find_obj()

```
def find_obj():
    #track position
    current_state = continue_game.getPosition(11)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/finding_object.wav')

    find_object = FindObject(scene_five, displays.screen_width, displays.screen_height, displays.object_images_in_bg, displays.object_images, displays.hint_circles)

    while find_object.run:
        #draw background and text
        screen.fill((159,118,61))
        displays.draw_buttons(screen, displays.find_obj_bg, 325, 310)
        displays.draw_text(f"Object: {find_object.gamef.found}/9", displays.font, 'black', 850, 110, screen, 50)

        #run game functions
        find_object.running(clock, fps, screen, main_menu, displays.hint_image)
```

The find_obj() function is used to run the finding object game. It first tracks the current state position and appends it to the list (to enable the continue button). Then, it also plays the music, draws the background, and draws text. Lastly, it also runs the game functions.

- scene_five()

```
def scene_five():
    #track position
    current_state = continue_game.getPosition(12)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/scene.wav')

    scene_5 = Scene("Assets/story/dialog5_text.txt", 5, displays.font, game_five, displays.screen_width, displays.screen_height)

    #first typing sound
    scene_5.dialog.type_sound.play()

    while scene_5.run:
        #draw background and black panel
        displays.draw_bg(screen, displays.scene2_bg)
        pygame.draw.rect(screen, 'black', [0, 450, 1000, 300])

        #dialog functions and event key
        scene_5.running(clock, fps, screen, displays.skip_button, displays.boy_icon, displays.boy_text_icon, displays.vampire_icon, displays.vampire_text_icon,main_menu, "")
```

The scene_five() function has a similar code to the scene_one(). The difference between them is that the index position of the current_state variable. Also, unlike scene one and scene two, scene five doesn't need any additional icons, so it passes the additional icon parameter as "".

- game_five()

```
def game_five():
    #track position
    current_state = continue_game.getPosition(13)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/battle.wav')

    game_5 = Game(screen, 280, 240, 790, 180, 'Vampire', 50, 8, 5, 1, displays.screen_height, displays.bottom_panel, displays.potion_img, displays.restart_img, displays.next_stage_img)

    while game_5.run:
        #draw background and panel
        displays.draw_bg(screen, displays.last_fight_bg)
        displays.draw_panel(screen, displays.screen_height, displays.bottom_panel, game_5.knight, game_5.enemy_list)
        displays.draw_buttons(screen, displays.fight5, 870, 50)

        game_5.running(clock, fps, screen)

        #control player action
        #reset action variable
        game_5.game.reset_state()

        #attacking enemy
        game_5.game.attack_enemy(game_5.enemy_list)
        for enemy in game_5.enemy_list:
            if enemy.rect.collidepoint(game_5.game.pos):
                #hide mouse
                pygame.mouse.set_visible(False)
                #show sword
                displays.draw_sword(screen, game_5.game.pos)

        #show number potion remaining
        displays.draw_text(str(game_5.knight.potions), displays.font, red, 145, displays.screen_height -displays.bottom_panel+85, screen, 40)

        #check if game over or not, check click
        game_5.check_game_state(screen, displays.victory_img, displays.defeat_img, scene_six, main_menu)
```

The game_five() function has the similar code with the game_one(). The difference between them is that the index position of the current_state variable.

- scene_six()

```
def scene_six():
    #track position
    current_state = continue_game.getPosition(14)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/ending.wav')

    scene_6 = Scene("Assets/story/dialog6_text.txt", 6, displays.font, end_scene, displays.screen_width, displays.screen_height)

    #first typing sound
    scene_6.dialog.type_sound.play()

    while scene_6.run:
        #draw background and black panel
        displays.draw_bg(screen, displays.scene2_bg)
        pygame.draw.rect(screen, 'black', [0, 450, 1000, 300])

        #dialog functions and event key
        scene_6.running(clock, fps, screen, displays.skip_button, displays.boy_icon, displays.boy_text_icon, "", "",main_menu, "")
```

The scene_six() function has a similar code to the scene_one(). The difference between them is that the index position of the current_state variable. Also, unlike scene one and scene two, scene six doesn't need any additional icons, so it passes the additional icon parameter as "". Scene six also only needs the boy icon and boy text icon, so it also passes "" for icon2 and icon2_text.

- end_scene()

```
def end_scene():
    #track position
    current_state = continue_game.getPosition(15)
    current_state_list.append(current_state)

    #play music
    game_function2.play_bgm('Assets/audio/music/ending.wav')

    game_end = GameInstruction(main_menu, displays.screen_width, displays.screen_height)

    while game_end.run:
        #draw background
        pygame.draw.rect(screen, 'black', [0, 0, 1000, 650])

        #draw text
        height = 250
        for i in game_end.all3:
            displays.draw_text(i, displays.font, white, 500, height, screen, 60)
            height+=60

        #event checker and functions
        game_end.running(clock, fps, main_menu)
```

The end_scene() function has the similar code with the game_instruction(). The difference between them is that the index position of the current_state variable.

Lastly, after the functions, we also have these lines of codes:

```
#list of all scene and game
all_function = [scene_one, scene_one, game_instruction, game_one, game_two,
                game_three, scene_two, scene_three, game_four, scene_four,
                game_instruction2, find_obj, scene_five, game_five, scene_six, end_scene]
#enable continue function
continue_game = ContinueGame(all_function)

#starts game in the main menu
main_menu()

pygame.quit()
```

First of all, the all_function variable specifies all of the functions listed above. Then, it creates an object of the ContinueGame class. Then, it calls the main_menu() function to start the game in the main menu. And, lastly, when the game loop is over, it will quit the game.

10. running_text.py

- class DialogText

```
import pygame
class DialogText:
    def __init__(self, font, messages):
        self.font = font
        self.messages = messages
        self.snip = font.render('', True, 'white')
        self.counter = 0
        self.speed = 3
        self.done = False
        self.active_message = 0
        self.message = messages[self.active_message]
        self.type_sound = pygame.mixer.Sound('Assets/audio/sfx/typing.wav')
        self.thud_sound = pygame.mixer.Sound('Assets/audio/sfx/thud.wav')
```

The __init__() method here initializes some attributes, like the font, messages, snip (to snip the messages), counter (to restrict how much text is displayed each game loop), speed (the speed of displaying the text), done, active_message(the index of the message displayed), type_sound, and thud_sound.

```
def running_message(self, screen):
    #to restrict how many text can be blit at a time
    if self.counter < self.speed * len(self.message):
        self.counter += 1
    elif self.counter >= self.speed*len(self.message):
        self.done = True

    #for a frame, only display some of the message snip.
    #So, when running it, it looks like animations of a running text
    self.snip = self.font.render(self.message[0:self.counter//self.speed], True, 'white')
    screen.blit(self.snip, (30,525))
```

The running_message() method is used to run the message and display it to the player. It has a ‘counter’ variable that increases each game loop. Then, it will display the snip message on the screen. The snip message is obtained by displaying the message from the index 0 to the counter. So, when the game loop runs, it will look like the text is been typed out. But actually, the thing that happened is that in every game loop, this function updated the text image to be displayed on the screen.

```
def checking_message_done(self, index):
    #play typing sound
    self.type_sound.play()

    #if done, run next message in the list
    if self.done==True and self.active_message < len(self.messages)-1:
        self.active_message +=1
        self.done = False
        self.message = self.messages[self.active_message]
        self.counter = 0

    #several restrictions for the first cutscene
    if index == 1:
        if self.active_message ==12:
            self.type_sound.stop()
        if self.active_message == 3:
            self.type_sound.stop()
            self.thud_sound.play()

    #restriction for second cutscene
    elif index == 2:
        if self.active_message ==9:
            self.type_sound.stop()
```

The checking_message_done() will check if the message has been displayed or not. If it has been done, then it will increase the active_message index, so the next message in the list can be displayed. It is also responsible for playing the typing sound and restricting some sound effects in some parts of the cutscene.

```
#show the character based on their dialog
def scene_1_function(self, boy, boy_text, vampire, vampire_text, scroll, screen):
    boy_rect= boy.get_rect(midleft = (30, 300))
    boy_text_rect = boy_text.get_rect(midleft =(30, 450))
    vampire_rect = vampire.get_rect(midright = (980,280))
    vampire_text_rect = vampire_text.get_rect(midright=(970,450))
    scroll_rect = scroll.get_rect(center=(500,325))

    if self.active_message == 1 or self.active_message == 7 or self.active_message == 10 :
        screen.blit(boy, boy_rect)
        screen.blit(boy_text, boy_text_rect)

    elif (self.active_message == 4 or self.active_message == 6 or self.active_message == 8
        or self.active_message == 9 or self.active_message == 11 or self.active_message==13):
        screen.blit(vampire, vampire_rect)
        screen.blit(vampire_text, vampire_text_rect)

    elif self.active_message == 12:
        screen.blit(scroll, scroll_rect)

def scene_2_function(self, boy, boy_text, castle, screen):
    boy_rect= boy.get_rect(midleft = (30, 320))
    boy_text_rect = boy_text.get_rect(midleft =(30, 450))
    castle_rect = castle.get_rect(center=(500, 275))

    if (self.active_message == 0 or self.active_message == 1 or
        self.active_message == 3 or self.active_message == 4 or
        self.active_message == 6 or self.active_message == 7 or self.active_message == 10):
        screen.blit(boy, boy_rect)
        screen.blit(boy_text, boy_text_rect)

    elif self.active_message == 9:
        screen.blit(castle,castle_rect)

def scene_3_function(self, boy, boy_text, man, man_text, screen):
    boy_rect= boy.get_rect(midleft = (30, 350))
    boy_text_rect = boy_text.get_rect(midleft =(30, 450))
    man_rect = man.get_rect(midright = (980,320))
    man_text_rect = man_text.get_rect(midright=(950,450))

    if (self.active_message == 0 or self.active_message == 3 or self.active_message == 4
        or self.active_message == 8 or self.active_message == 11 or self.active_message == 13):
        screen.blit(boy, boy_rect)
        screen.blit(boy_text, boy_text_rect)

    elif self.active_message == 1 or self.active_message == 5 or self.active_message == 7 or self.active_message == 9 :
        screen.blit(man, man_rect)
        screen.blit(man_text, man_text_rect)

def scene_4_function(self, boy, boy_text, man, man_text, screen):
    boy_rect= boy.get_rect(midleft = (30, 350))
    boy_text_rect = boy_text.get_rect(midleft =(30, 450))
    man_rect = man.get_rect(midright = (980,320))
    man_text_rect = man_text.get_rect(midright=(950,450))

    if (self.active_message == 0 or self.active_message == 2 or self.active_message == 6
        or self.active_message == 8 or self.active_message == 10 or self.active_message == 12):
        screen.blit(boy, boy_rect)
        screen.blit(boy_text, boy_text_rect)

    elif (self.active_message == 1 or self.active_message == 3 or self.active_message == 4
        or self.active_message == 5 or self.active_message == 7 or self.active_message == 9):
        screen.blit(man, man_rect)
        screen.blit(man_text, man_text_rect)
```

```
def scene_5_function(self, boy, boy_text, vampire, vampire_text, screen):
    boy_rect= boy.get_rect(midleft = (30, 300))
    boy_text_rect = boy_text.get_rect(midleft =(30, 450))
    vampire_rect = vampire.get_rect(midright = (980,280))
    vampire_text_rect = vampire_text.get_rect(midright=(970,450))

    #because there is too much dialog, for this scene, the dialog index will be stored in dictionary
    dialog_index = {"boy":[0,1,3,6,7,16], "vampire":[2,4,9,10,11,12,13,17,18,19,22]}

    for key,dialog_list in dialog_index.items():
        if key == "boy":
            for i in dialog_list:
                if self.active_message == i:
                    screen.blit(boy, boy_rect)
                    screen.blit(boy_text, boy_text_rect)

        elif key == "vampire":
            for i in dialog_list:
                if self.active_message == i:
                    screen.blit(vampire, vampire_rect)
                    screen.blit(vampire_text, vampire_text_rect)
```

```
def scene_6_function(self, boy, boy_text, screen):
    boy_rect= boy.get_rect(midleft = (30, 300))
    boy_text_rect = boy_text.get_rect(midleft =(30, 450))

    if (self.active_message == 2 or self.active_message == 4 or self.active_message == 5
        or self.active_message == 7 or self.active_message == 8 or self.active_message == 10):
        screen.blit(boy, boy_rect)
        screen.blit(boy_text, boy_text_rect)
```

These functions above are used to display the icon based on the active message that is being shown. For example, when the message plays the boy dialogue, then these functions will show the boy icon in the cutscene. Each of the scenes has its scene function to determine when the icons will show up.

11. transition.py

- class Transition

```
import pygame

class Transition:
    def __init__(self, next_state, screen_width, screen_height):
        self.display_surface = pygame.display.get_surface()
        self.next_state = next_state

        #overlay image
        self.image = pygame.Surface((screen_width,screen_height))
        self.color = 255
        self.speed = -2
```

The __init__() method initializes attributes for the transition, such as display_surface (to get the surface on which the transition will be played), next_state (to determine the next state to go), image, colour, and speed.

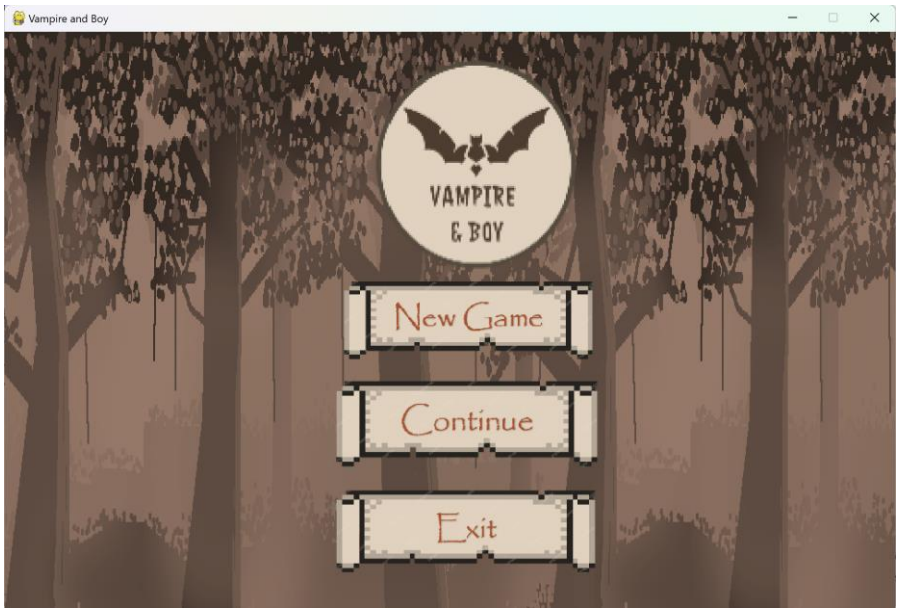
```
def play(self):
    #playing color transition
    self.color += self.speed
    if self.color <= 0:
        self.speed *= -1
        self.color = 0
        self.next_state()
    if self.color > 255:
        self.color = 255
        self.speed = -2

    self.image.fill((self.color,self.color,self.color))
    self.display_surface.blit(self.image, (0,0), special_flags = pygame.BLEND_RGBA_MULT)
```

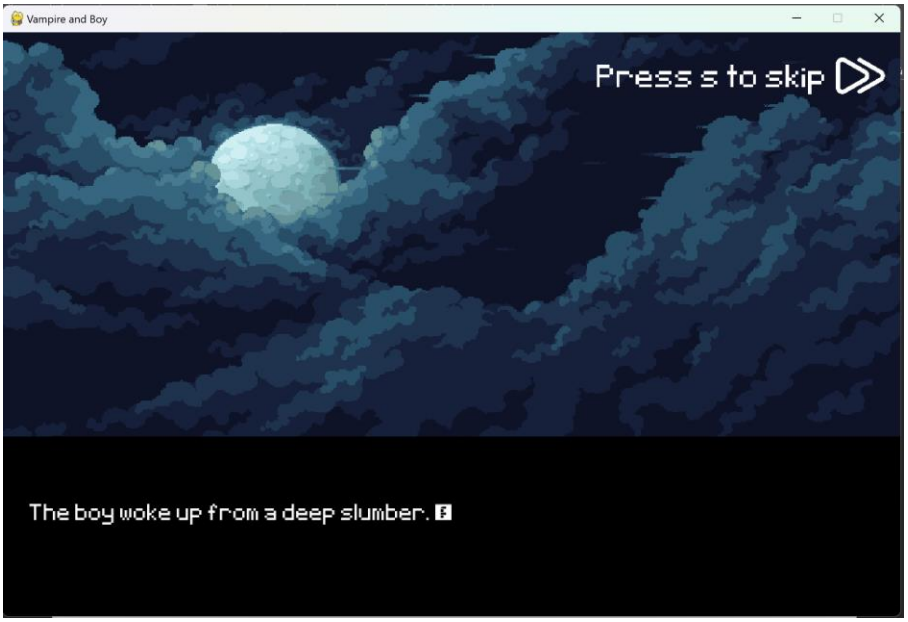
The play() method will play the colour transition based on the speed. First, it displays the colour from white (transparent) to black. Then, If the colour number becomes negative, it will make the speed becomes a positive number, so that the colour will change from black to white (transparent) again. It also moves to the next state. And, when the colour exceeds 255, then it restarts the colour attribute to 255 and the speed attribute to -2. Lastly, it fills the screen with the colour and sets it with special flags of BLEND_RGBA_MULT (so that it has a transparent mode).

K. Evidence of Working Program

1. Main Menu



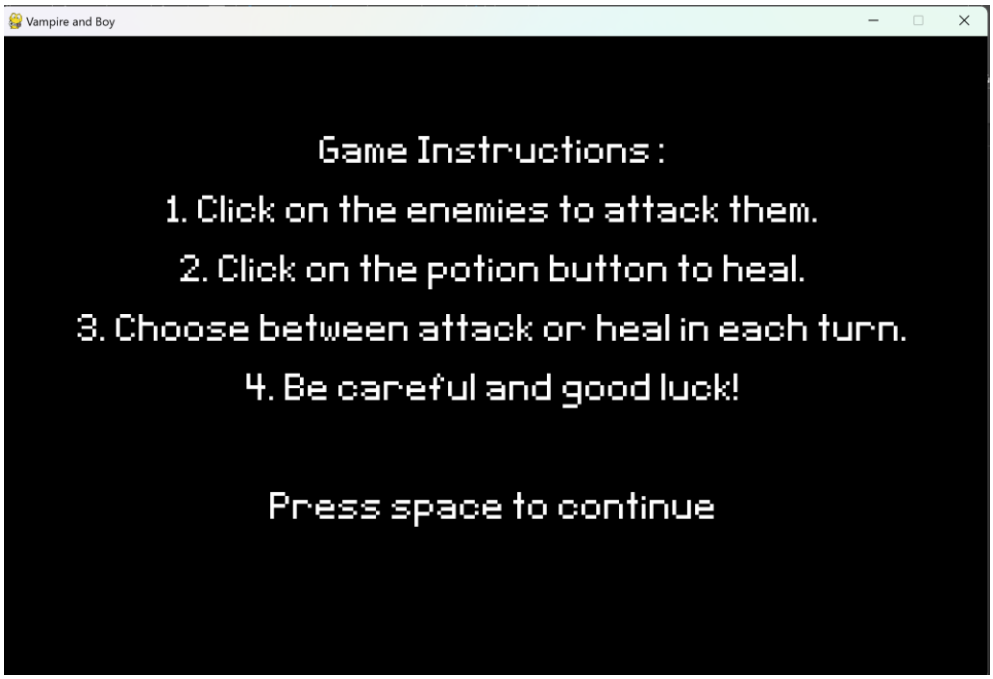
2. Cutscene



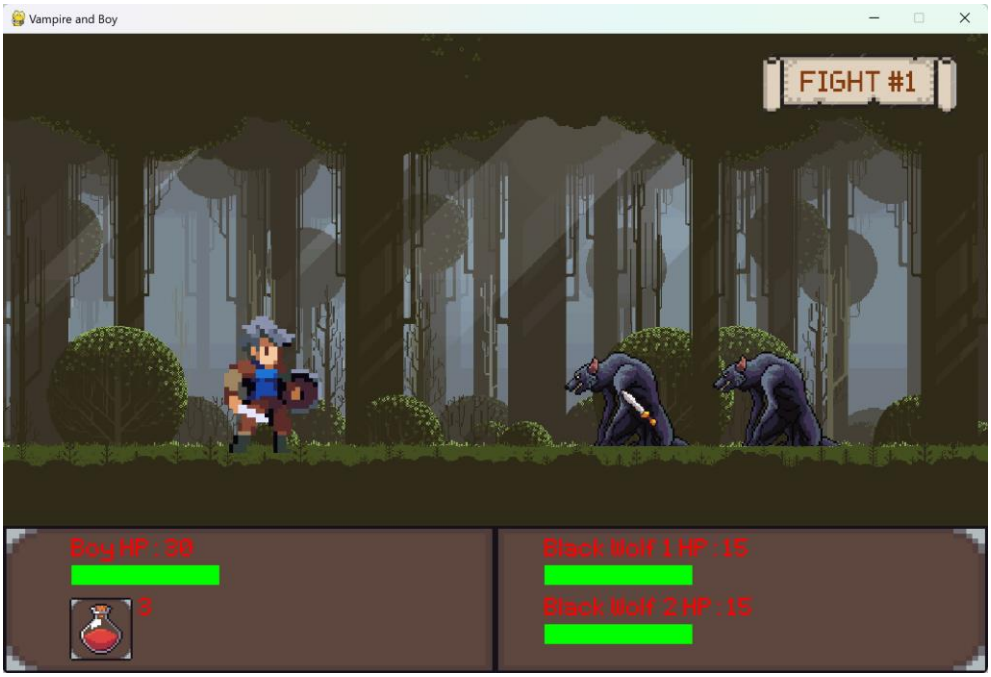

Vampire and Boy
Press s to skip
Boy
"Huft... I am so tired."

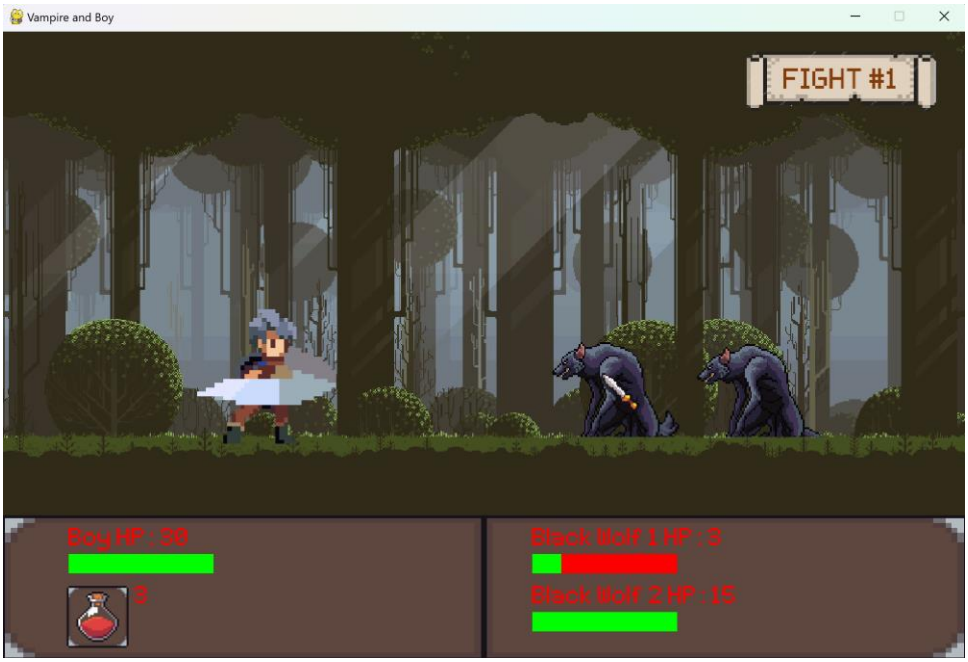

Vampire and Boy
Press s to skip
Man
"Hey, who are you and why are you in my house?!"

3. Game Instructions

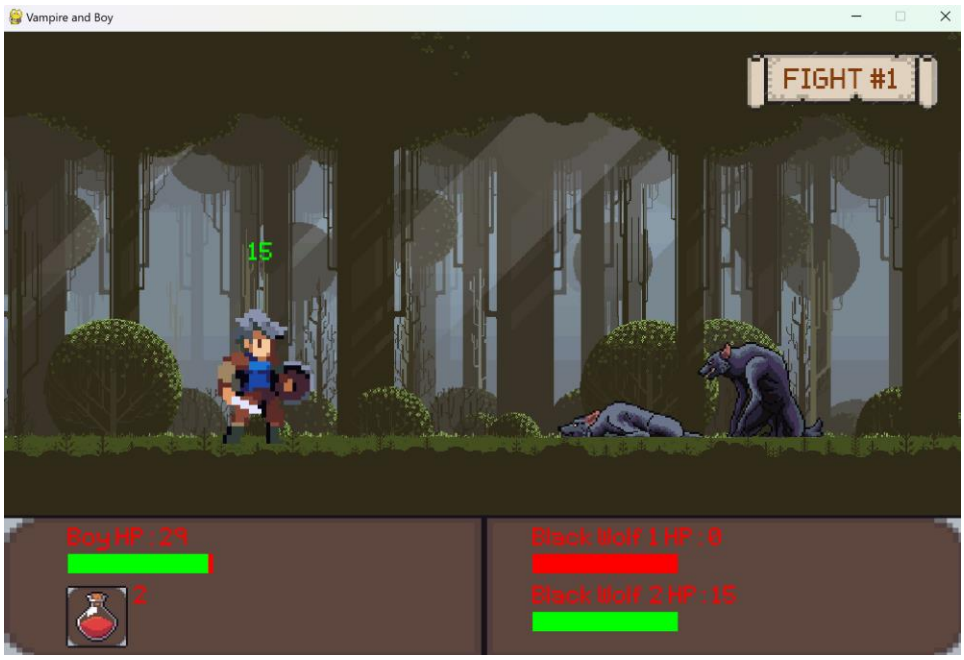


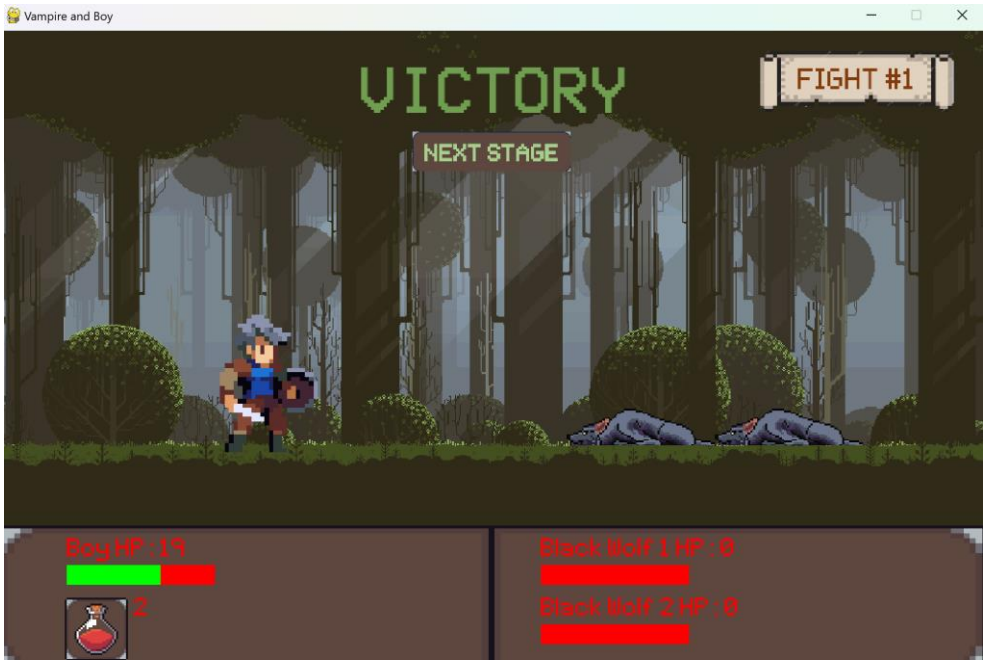
4. Battle Scene

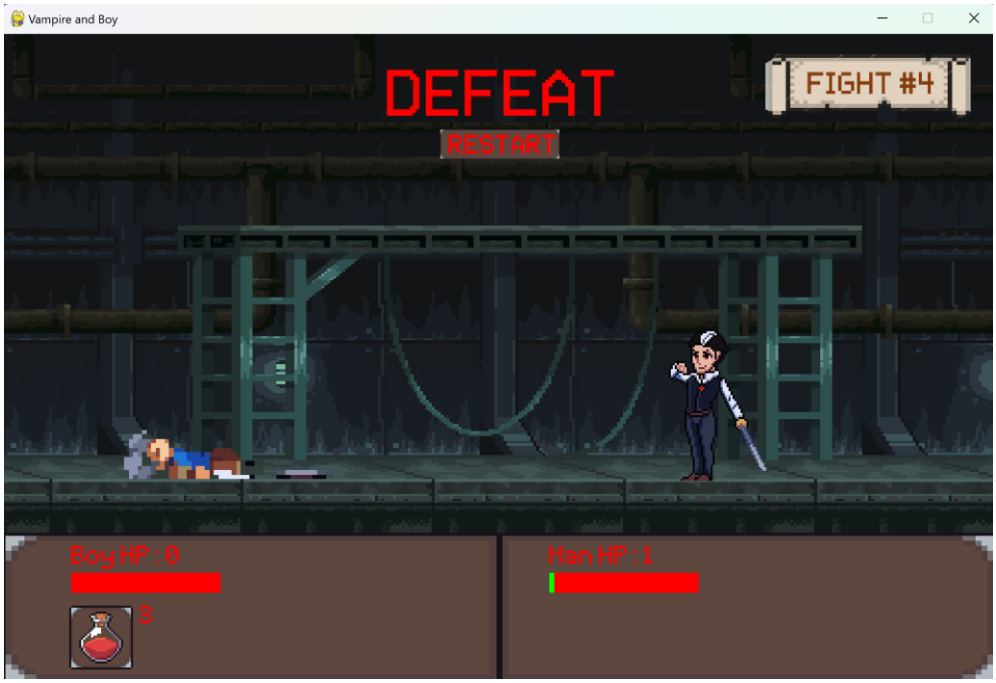


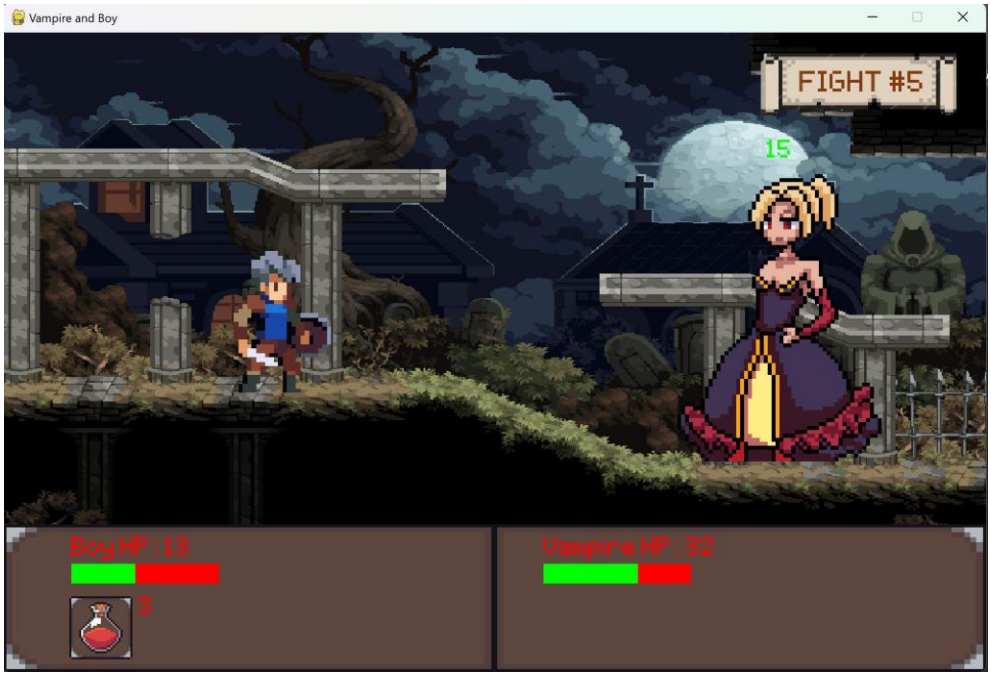

Vampire and Boy
FIGHT #1
Boy HP : 30
3
Black Wolf 1 HP : 3
Black Wolf 2 HP : 15


Vampire and Boy
FIGHT #1
Boy HP : 14
3
Black Wolf 1 HP : 0
Black Wolf 2 HP : 15

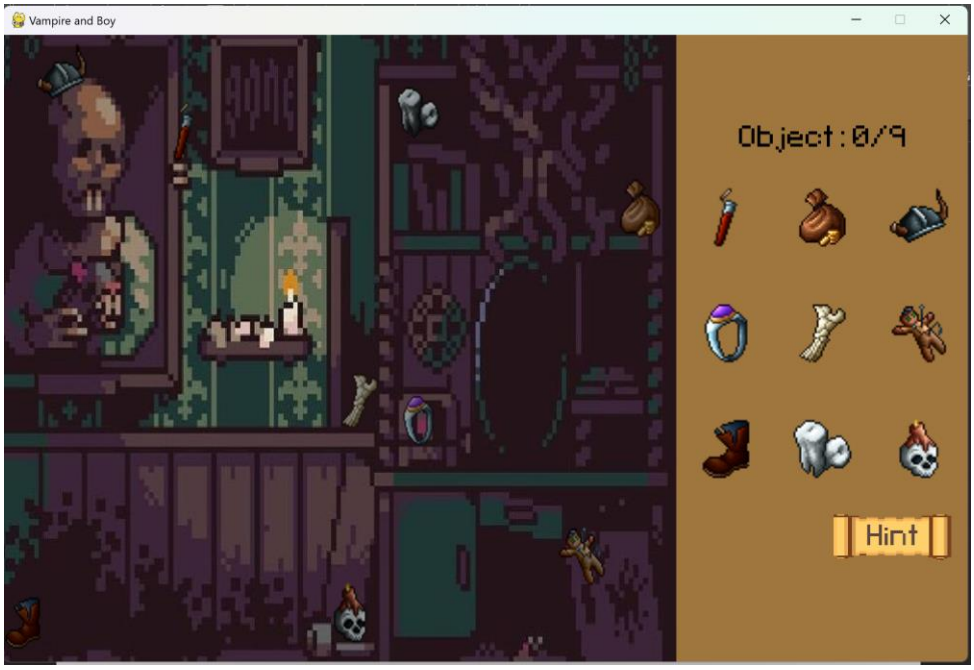

Vampire and Boy
FIGHT #1
15
Boy HP : 29
2
Black Wolf 1 HP : 0
Black Wolf 2 HP : 15

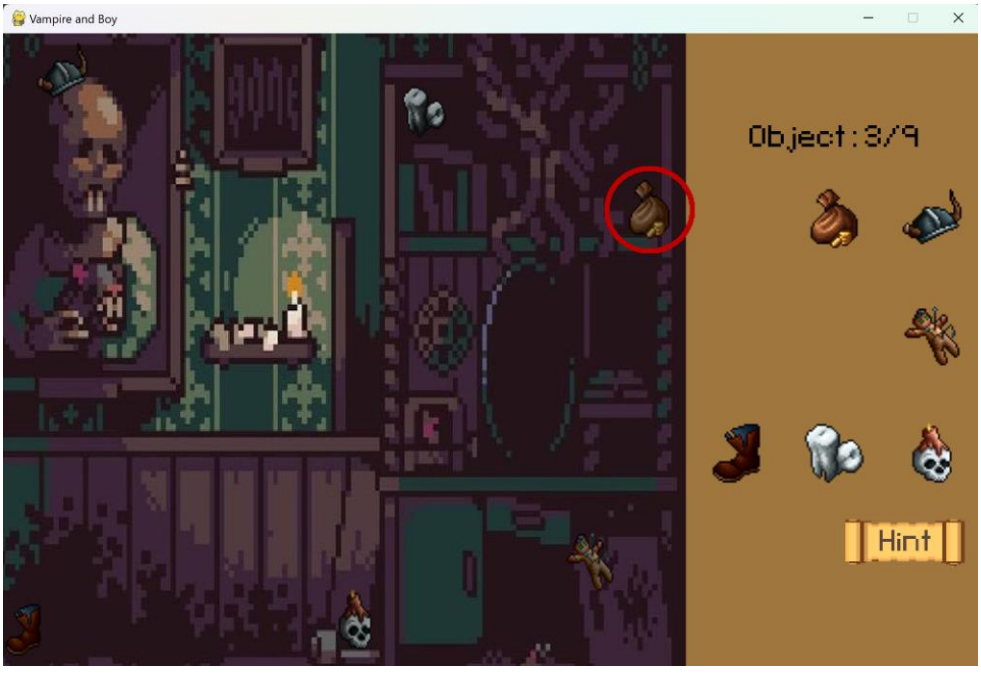

Vampire and Boy
VICTORY
FIGHT #1
NEXT STAGE
Boy HP : 19
2
Black Wolf 1 HP : 0
Black Wolf 2 HP : 0

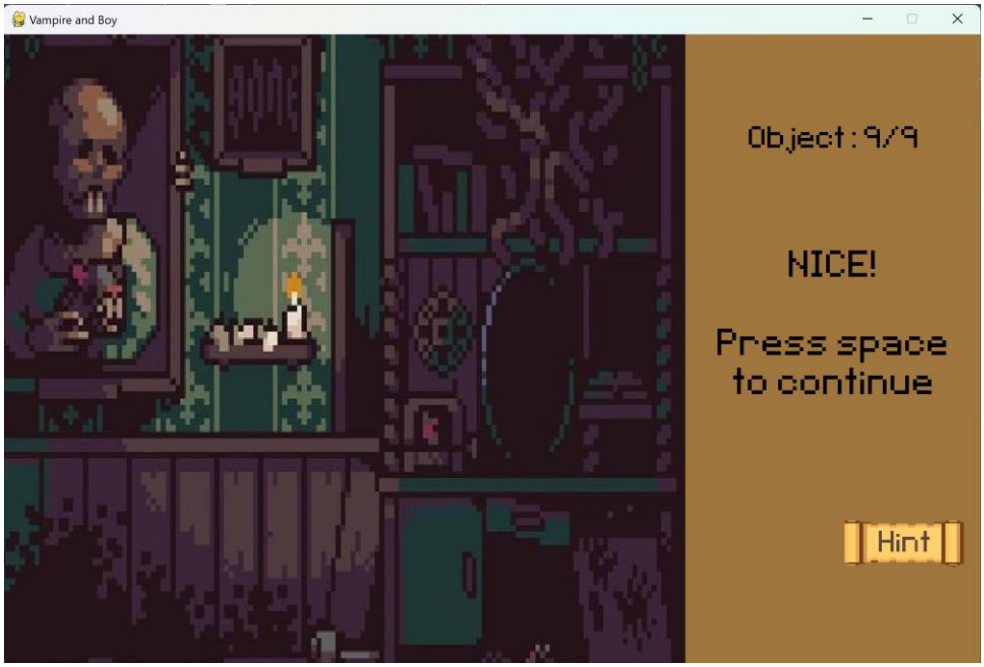

Vampire and Boy
DEFEAT
RESTART
FIGHT #4
Boy HP : 0
3
Man HP : 1


Vampire and Boy
FIGHT #5
15
Boy HP : 13
3
Vampire HP : 32

5. Finding Object






Vampire and Boy
Object : 9/9
NICE!
Press space
to continue
Hint

L. Reflection

I have learned a lot of things when working on this project. I decided to make this game because I love playing games with stories in them. In my opinion, a game with a story is more interesting and fun to play. So, I researched a lot about how to make a visual novel game with mini-games in it. But, as I researched further, I found out that Pygame can't be used to make visual novel games. It is more suitable to use Renpy instead. But I don't want to use Renpy, because if I use it, it will be hard to implement OOP in my Python files, which then doesn't meet the final project requirement. So, after a lot of days thinking, I decided to make a simple fighting game without a story.

After completing a fighting game, I realized I still had many times to add new things to it. So, I add more battle scenes for the game. But, after adding three battle scenes, I thought again about adding a story to this game. So, I spent a day thinking about the story and character. After researching the character sprites in the game assets store, I got the inspiration to make this story. Then, I tried to make the cutscene. Luckily, there is a tutorial video on YouTube about how to make the typing text cutscene that I wanted and thankfully, I could successfully implement it in my code.

After making cutscenes and battles, I found out that I could do more again. So, I added the finding object game. At first, I thought about adding three stages of it. But the game will last too long and become boring. So, I decided to add only one stage of it. However, the challenge came. I can't find a tutorial for it. So, I can only rely on my fresh knowledge about Pygame to make this finding object game. And, thankfully, the finding object game is done too.

In this process, I encountered a lot of bugs. I spent more time on the debugging process than the coding itself. Because I added many new functions to it, so I need to debug it myself too. The stack overflow community and the internet have helped me a lot in this process. There are many references and documentation that helped during this time.

I also learned that when making games, the assets are as important as the code. We need to look for suitable assets to make the game look good. And, it needs a lot of time and patience to make or search for a good game asset. Although I'm quite happy with the result I have now, I know that some of the game assets still don't look too good, since my design skill is lacking. So, I learned that a designer is a very important figure when making a game.

And, another issue that I had is my code readability. I am aware that my code is very messy and has a complicated structure, especially the classes and objects. And also, a few parts of my code have a similar function that is repeated. I tried to increase the code readability before but I can only go as far as this final result. I tried to make it simpler before, but it shows an error (mainly because of the Displaying class). The Displaying class loads the images, so the object can only be declared in the main file. So, every method in the Displaying class can be only called in the main file. So, my main file becomes quite long and repeated. But I'm already satisfied that I can go as far as this to increase my code readability.

Overall, this project is a fun experience for me. It was also a bit stressful, but thankfully, all things went well in the end. I originally wanted to increase the mini-games and write a more complicated story, but I was already running out of time (because I still needed to leave some time to write the report). So, I can only make this simple game for this final project. But I am quite satisfied with the result and I hope I can do better next time.

M. References

1. Tutorial Video

- https://www.youtube.com/watch?v=0RryiSjpJn0
- https://youtube.com/playlist?list=PLjcN1EyupaQnvpv61iriF8Ax9dKra-MhZ&si=aTzLqczT_tG4VgEt
- https://www.youtube.com/watch?v=ARt6DLP38-Y&t=628s
- https://www.youtube.com/watch?v=DhK5P2bWznA

2. Report Information

- https://www.pygame.org/wiki/about
- https://www.w3schools.com/python/module_random.asp
- https://www.geeksforgeeks.org/python-sys-module/
- https://www.w3schools.com/python/module_math.asp

3. **Image Assets**

- **Background Image**

  Battle 1, Battle 2, Battle 3:

https://edermunizz.itch.io/free-pixel-art-forest

Main Menu

https://saurabhkgp.itch.io/pixel-art-forest-background-simple-seamless-parallax-ready-for-2d-platformer-s

Scene 1, Scene 2, Scene 5, Scene 6, and Battle 5

https://anokolisa.itch.io/moon-graveyard?download

Scene 3, Scene 4, and Battle 4

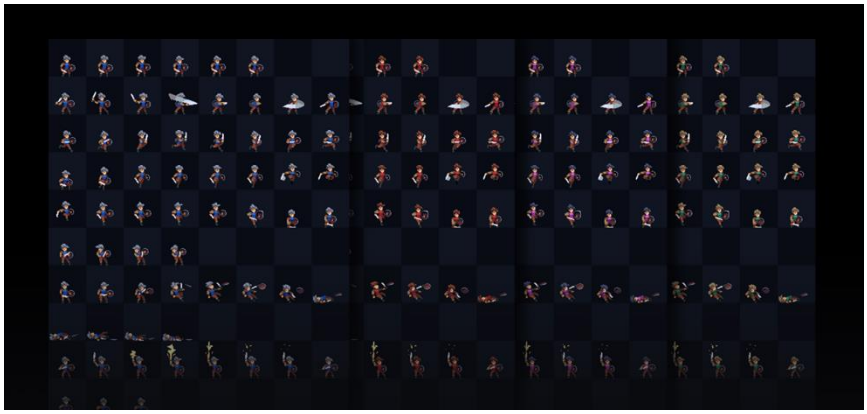
https://ansimuz.itch.io/bulkhead-walls-environment

Finding Object

https://www.pinterest.com/pin/oc-spooky-room-pixelart--468796642469406395/

- **Characters**

Boy



https://brullov.itch.io/generic-char-asset

Werewolf

https://free-game-assets.itch.io/free-werewolf-sp rite-sheets-pixel-art

Vampire and Man

https://free-game-assets.itch.io/free-vampire-pixel-art-sprite-sheets

- **Others**

  Hidden Objects

https://free-game-assets.itch.io/free-undead-loot-pixel-art-icons

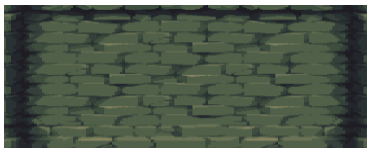
Logo

https://www.brandcrowd.com/maker/logo/halloween-bat-wings-65267?text=VAMPIRE&colorPalette=orange&isVariation=True

Scroll

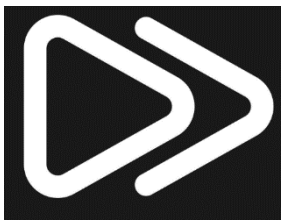
https://www.freepik.com/premium-vector/pixel-scroll-ribbon-ancient-manuscript-parchment-banner_47929261.htm

Text Panel



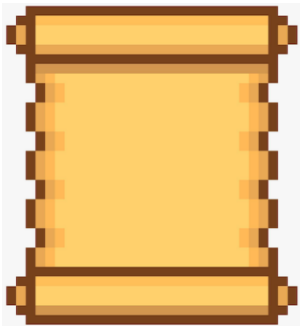
https://twitter.com/JoeCreates/status/1593164892219248643

Skip Button



https://www.vecteezy.com/vector-art/22737175-fast-reverse-button-pixel-perfect-white-linear-ui-icon-for-dark-theme-music-player-play-video-vector-line-pictogram-isolated-user-interface-symbol-for-night-mode-editable-stroke

Scroll (Finding Object List and Hint Button)



https://www.freepik.com/premium-vector/rolled-old-paper-pixel-art-style_22094015.htm

4. **Background Music**
    - Main Menu: Ludum Dare 38 Track 2 (https://tallbeard.itch.io/music-loop-bundle)
    - Cutscene: 90s Kid Revenge (https://clement-panchout.itch.io/yet-another-free-music-pack)
    - Battle Scene: Ludum Dare 38 Track 4 (https://tallbeard.itch.io/music-loop-bundle)
    - Finding Object: Ludum Dare 38 Track 6 (https://tallbeard.itch.io/music-loop-bundle)
    - Ending Scene: Ludum Dare 30 Track 1 (https://tallbeard.itch.io/music-loop-bundle)

5. **Sound Effects**
    - typing sound: https://pixabay.com/sound-effects/keyboard-typing-5997/
    - thud sound: https://pixabay.com/sound-effects/traditional-stamp-44189/
    - click sound: https://pixabay.com/sound-effects/mouse-click-153941/
    - enemy attack: https://pixabay.com/sound-effects/swinging-staff-whoosh-strong-08-44658/
    - boy attack: https://pixabay.com/sound-effects/knife-slice-41231/
    - heal sound: https://pixabay.com/sound-effects/084373-heal-36672/
    - victory sound: https://pixabay.com/sound-effects/goodresult-82807/
    - defeat sound: https://pixabay.com/sound-effects/failure-1-89170/
    - find sound: https://pixabay.com/sound-effects/correct-choice-43861/
    - success in finding object sound: https://pixabay.com/sound-effects/success-1-6297/
    - hint sound: https://pixabay.com/sound-effects/snd-fragment-retrievewav-14728/