BINUS UNIVERSITY

BINUS INTERNATIONAL

Algorithm and Programming

Final Project Report

Student Information:

Name: Justin Hadinata **Student ID**: 2702298236

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

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

Type of Assignment: Final Project report

A. Introduction

1. Background

Python is the first programming language I've ever learned; I remember started learning it back in 2021. I've had tons fun with it and back then I also was fond with Video Games, so one of my goals was making a game with Python. But I remember feel pressured with the coding and never actually learned the library needed to do so that is Pygame. This decision gave me an Imposter Syndrome.

So, after some consideration, in order to challenge myself and prove to myself that I can indeed do the stuff I've always wanted, I decided to make a game in Pygame (Also to remove my Imposter Syndrome Feeling). The Game I've decided is a short Visual Novel with some 2D Gameplay to it. At first, I actually wanted to use RenPy Game engine for it, but turns out RenPy uses its own programming language instead of Python (Even though the languages are very similar), So it wouldn't be fit.

2. Identified Problems

There are two problems I wishes to solve by doing this Project. The first one is related to the limitations of mobility and freedom in the Visual Novel games, this limitation heavily impacted the gameplay and results in a repetitive and tedious gameplay that sadly diminish the potential of any Immersive storytelling. I'm aware that most of people that played Visual Novel already know about this fact, and actually played this genre of game for that exact reason.

But there are some groups that actually liked the idea of the genre of Visual Novel and wishes to enjoy it, but get easily appalled by the repetitive gameplay. So, by creating this Visual Novel game that gave the player more mobility and freedom, it aims to captivate, including myself, or groups of people who find the Traditional Visual Novel gameplay boring, with few aspects added to the Traditional genre. The second problem is related to the background.

The realm of programming is vast and, as a result it can be complex and difficult for certain individuals to grasp. This complexity often results in some Individuals experiencing an Imposter Syndrome. This issue also exacerbated by social medias, especially with how many talented programmers there are in the Internet. Therefore, by embarking on this project, it would inspire fellow programmers that we can do anything we aside our mind too, and in turn also relieves our senses of Imposter Syndrome.

B. Project Specification

1. Project Description

The Game is called Behind Me. It's an extremely simple and short Visual Novel horror game home invasion themed with some 2D game elements to it. The game is about a little boy named Timmy who lives with his parents, and the game takes place at a cabin during a snowstorm. During the night, a stranger invades their house, and Timmy wakes up due to the sound, now you as a player has to play as Timmy and control his action. When running the code, the game starts with a simple menu, that has start button and quit button in it. Clicking the quit button would terminate the program immediately but upon clicking the start button, the game will begin. The game starts by first showing a cinematic scene of a cabin during a winter, it was on a snow storm, the scenes last roughly 5 seconds.

After the cinematic scene is over, a textbox will appear, narrating the story and situation of the game, the player can scheme through the story by pressing the Space bar just like in a visual novel. After some narration and storytelling, the player will be presented by choices and decisions making, and these choices and decision would lead into different path and stories just like in a Traditional Visual Novel. Certain decision would lead the player to a simple 2D side-scrolling mode, where they can explore Timmy's room and can go into different story path depending on an action in the room. Certain decision would lead into a game over scene where the player would be given an option to go back to the main menu, and certain decision would lead into a winning screen with the same go back option. Later in the game there also will be

an enemy whom you have to dodge. The goal of the game is to received the good ending or survive the whole ordeal.

2. **Libraries / Modules**
   - **Pygame**

     Pygame is a set of Python modules that is designed for making Video Games. For this particular project, they're used for displaying the images, music, tracking player's input, responsible for the animation, etc.
   - **Time**

     Time is a python module that provides various time-related functions. In my case, I use the modules to add some timing in the game for functionality reason and dramatic reasons.

   - **Sys**

     Sys is a module that provides various functions and variables that are used to manipulate different parts of the python runtime environment. In my case, I uses them to terminate the game safely, and efficiently without causing an error or bugs.
3. **Game Mechanics overview**
   - Player can press space to navigate through the story
   - If chooses to investigate, player can move left and right using A, D. Tiny jump using Y, and jump by pressing space.
4. **Third Party Software and Source**
   - **Aseprite**

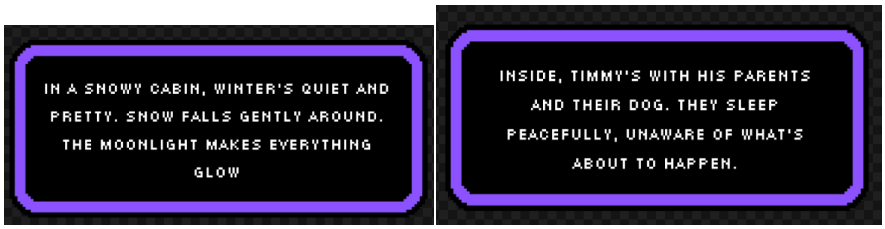
     Aseprite is a program to create pixel arts and animate them. Aseprite is used a lot in the project for the making of the pixel arts and the animation, like the Player and Killers sprite.
   - **Canva**

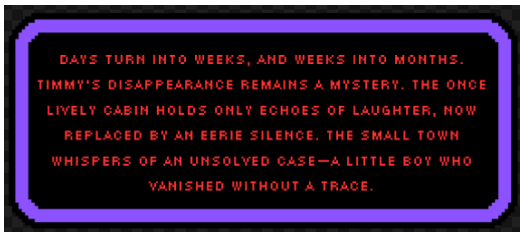
     Canva is a free-to-use online graphic design tool, which is heavily used in the project for the Textboxes and decision box.
   - **Youtube**

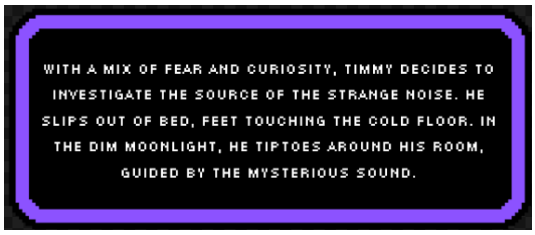
Youtube are a video platform Some of the sound effects are taken from Youtube.com as its source.

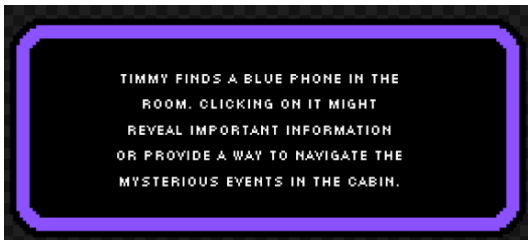
5. Game Assets

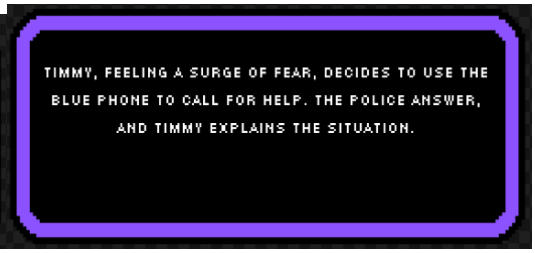
Textboxes (Source: Made by myself):

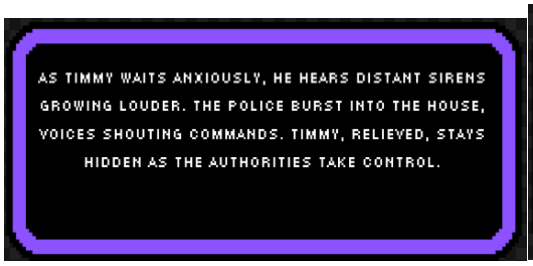


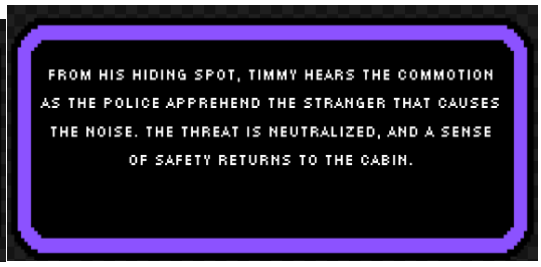

DAYS TURN INTO WEEKS, AND WEEKS INTO MONTHS. TIMMY'S DISAPPEARANCE REMAINS A MYSTERY. THE ONCE LIVELY CABIN HOLDS ONLY ECHOES OF LAUGHTER, NOW REPLACED BY AN EERIE SILENCE. THE SMALL TOWN WHISPERS OF AN UNSOLVED CASE—A LITTLE BOY WHO VANISHED WITHOUT A TRACE.

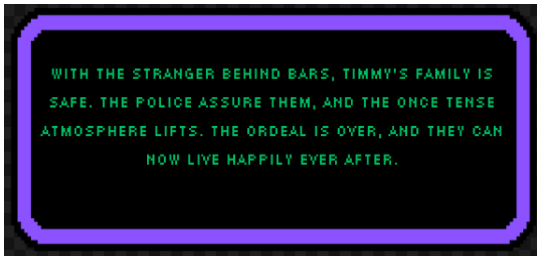

WITH A MIX OF FEAR AND CURIOSITY, TIMMY DECIDES TO INVESTIGATE THE SOURCE OF THE STRANGE NOISE. HE SLIPS OUT OF BED, FEET TOUCHING THE COLD FLOOR. IN THE DIM MOONLIGHT, HE TIPTOES AROUND HIS ROOM, GUIDED BY THE MYSTERIOUS SOUND.


TIMMY FINDS A BLUE PHONE IN THE ROOM. CLICKING ON IT MIGHT REVEAL IMPORTANT INFORMATION OR PROVIDE A WAY TO NAVIGATE THE MYSTERIOUS EVENTS IN THE CABIN.

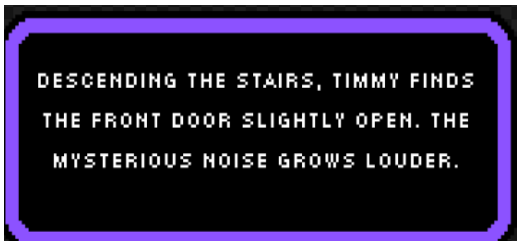

TIMMY, FEELING A SURGE OF FEAR, DECIDES TO USE THE BLUE PHONE TO CALL FOR HELP. THE POLICE ANSWER, AND TIMMY EXPLAINS THE SITUATION.


AS TIMMY WAITS ANXIOUSLY, HE HEARS DISTANT SIRENS GROWING LOUDER. THE POLICE BURST INTO THE HOUSE, VOICES SHOUTING COMMANDS. TIMMY, RELIEVED, STAYS HIDDEN AS THE AUTHORITIES TAKE CONTROL.


FROM HIS HIDING SPOT, TIMMY HEARS THE COMMOTION AS THE POLICE APPREHEND THE STRANGER THAT CAUSES THE NOISE. THE THREAT IS NEUTRALIZED, AND A SENSE OF SAFETY RETURNS TO THE CABIN.


WITH THE STRANGER BEHIND BARS, TIMMY'S FAMILY IS SAFE. THE POLICE ASSURE THEM, AND THE ONCE TENSE ATMOSPHERE LIFTS. THE ORDEAL IS OVER, AND THEY CAN NOW LIVE HAPPILY EVER AFTER.


DO YOU WANT TO GO DOWNSTAIRS?


DESCENDING THE STAIRS, TIMMY FINDS THE FRONT DOOR SLIGHTLY OPEN. THE MYSTERIOUS NOISE GROWS LOUDER.


FEELING A SENSE OF URGENCY, TIMMY DECIDES TO LEAVE THE HOUSE. HE STEPS INTO THE BITING COLD, THE SNOW-COVERED LANDSCAPE STRETCHING BEFORE HIM.


TIMMY, DETERMINED, TRUDGES THROUGH THE SNOWY TERRAIN, SEEKING HELP. THE WIND HOWLS, AND THE SNOWFLAKES OBSCURE HIS VISION, MAKING THE JOURNEY MORE CHALLENGING.
AS TIMMY BATTLES THE HARSH ELEMENTS, THE COLD BECOMES TOO MUCH TO BEAR. EXHAUSTED AND FREEZING, HE COLLAPSES IN THE SNOW. THE UNFORGIVING WINTER CLAIMS ANOTHER VICTIM, AND TIMMY'S JOURNEY ENDS TRAGICALLY.
IGNORING THE OPTION TO LEAVE, TIMMY BRAVELY DECIDES TO INVESTIGATE THE MYSTERIOUS NOISE. THE DARKNESS SEEMS TO ENVELOP HIM AS HE VENTURES FURTHER INTO THE HOUSE, GUIDED ONLY BY THE UNSETTLING SOUND.
FOLLOWING THE NOISE LEADS TIMMY TO A LOCKED ROOM. THE MUFFLED SOUNDS PERSIST, RAISING TENSION
OPTING TO UNLOCK THE DOOR, TIMMY DISCOVERS THE SOURCE OF THE NOISE—AN INTRUDER RUMMAGING THROUGH THE FAMILY'S BELONGINGS. SHOCK AND FEAR GRIP HIM AS THE INTRUDER BECOMES AWARE OF TIMMY'S PRESENCE.
TIMMY, QUICK ON HIS FEET, MANAGES TO DODGE THE ADVANCING KILLER. THE INTRUDER STUMBLES FORWARD, CRASHING INTO A SOLID OBJECT, RENDERING THEM UNCONSCIOUS.
WITH THE STRANGER INCAPACITATED, TIMMY SEIZES THE OPPORTUNITY TO CALL THE POLICE. SIRENS WAIL IN THE DISTANCE AS AUTHORITIES ARRIVE TO HANDLE THE SITUATION. THE DANGER IS AVERTED, AND TIMMY, THOUGH SHAKEN, SADLY THE KILLER HAS ALREADY KILLED THE DOG.
DESPITE TIMMY'S EFFORTS, THE KILLER'S STRIKE LANDS. THE ROOM ECHOES WITH THE SOUND OF THE IMPACT AS TIMMY CRUMPLES TO THE FLOOR.
IN THE FINAL MOMENTS OF CONSCIOUSNESS, TIMMY'S VISION BLURS. THE INTRUDER, HAVING SUCCEEDED, LEAVES THE ROOM. THE GAME CONCLUDES WITH A TRAGIC ENDING AS TIMMY'S JOURNEY COMES TO A DEVASTATING END.

2D Game Assets:

Player (Source: Made by myself):

PlayerJUMP.png Player1.png player2.png

Killer (Source: Made by myself):

Bed, Ground, and Tabke(Source: Made by myself):

Phone (Source:

https://www.google.com/url?sa=i&url=https%3A%2F%2Fsustasandesh.com.np%2Fzfa.php%3Fcid%3D111%26kys%3Dpixel%2Bde%2Bt%25C3%25A9l%25C3%25A9phone%26g%3D4&psig=AOvVaw2hM-

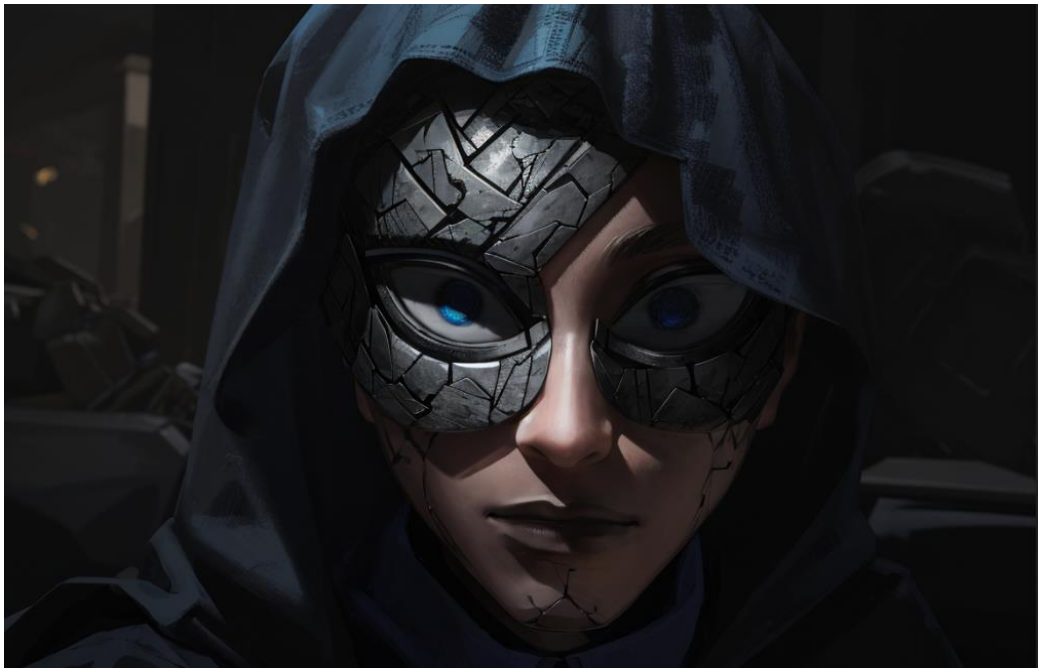
AgY1eJb4fJi4le0fn5&ust=1704897210206000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCNCzrJzD0IMDFQAAAAAdAAAAABAI)

Intro Snowing Cinematic Scene Gif (Source: https://www.deviantart.com/ccelestiiall/art/Cabin-in-the-Snow-723308283)

Game Title (Source: Made by Myself)

Home Screen (Source: From Myself, Using AI)

Killer Screen (Source: From Myself, Using AI)



C. Algorithm

For my Project, I only have one Python file, that is named Main.py:

The Essential & Main Algorithm

Main.py

```
import pygame
from sys import exit
import time
```

- The code starts by Importing the the Pygame, Sys and getting , and Time Module.

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

- We define a Class called Player using the Pygame OOP (Sprite)
- We can called the Pygame Sprite through the syntax -> pygame.sprite.Sprite inside the class argument.

```
def __init__(self):
    super().__init__()
    Player_standing_1 = pygame.image.load('player/Player1.png').convert_alpha()
    Player_standing_2 = pygame.image.load('player/Player2.png').convert_alpha()
```

- We first use the pygame.image.load syntax to display the images for our player, because our player has an Animation of 2 frame, we load 2 images and store them in a variable. They're in the def __init__(self): because, this module is where we want to load all of our assets, and super() so we can use the variable in any other modules

```
    self.Player_index = 0
    self.Player_standing = [Player_standing_1, Player_standing_2]
    self.Player_jump = pygame.image.load('player/PlayerJUMP.png').convert_alpha()

    self.image = self.Player_standing[self.Player_index]
    self.rect = self.image.get_rect(midbottom = (200, 500))
    # we set the rectangle, Gravity and the Jumping sound when certain input is
    #detected later on
    self.gravity = 0
    self.jump_sound = pygame.mixer.Sound('sound/jump.mp3')
    self.jump_sound.set_volume(0.2)
```

- In here we define the self.Player_index, this will be used for shuffling between 2 frame / images in order to create the animation, this can be done by putting all of the images inside the Image list. The self.rect to get the rectangle for the image so we can move the image easily and midbottom so our image can placed directly at the bottom. The self.gravity set to 0 will be used later for our gravity function and the sound will be played.

```
        pass
    def player_input(self):
        #We check the Player Input, and condition. For Jumping for example,
        #the player can only jump, when the rectangle position is at the bottom or 500,
        #Hence stopping an infinite jumping mechanic.
        keys = pygame.key.get_pressed()
        if keys[pygame.K_SPACE] and (self.rect.bottom >= 500) and  (canJump):
            self.gravity = -25
            self.jump_sound.play()
        if keys[pygame.K_w]:
            self.rect.y -= 10
        if keys[pygame.K_d]:
            self.rect.x += 10
        if keys[pygame.K_a]:
            self.rect.x -= 10
        #DO the same for gravity, but when mouse is clicked with the player rectangle
        if pygame.mouse.get_pressed()[0] and self.rect.collidepoint(pygame.mouse.get_pos()):
            if self.rect.bottom >= 500:
                self.gravity = -21
                self.jump_sound.play()
        pass
```

- These will track of our player input, and serves as the function so our player can move the character around. The way it works is that if our player presses W, A and D, it would move the rectangle x and y position, giving the illusion that they're moving. Player can also only jump if their rectangle position is at 500 or higher, stopping an infinite jump glitch. Before Continuing, I wanted to explain how the gravity for the player works.

```
        pass
    def apply_gravity(self):
        self.gravity += 1
        self.rect.y += self.gravity
        if self.rect.bottom >= 500:
            self.rect.bottom = 500
            self.gravity = 0
```

- So basically, first we increment the value of our player rectange Y with the gravity, this in turn would cause our player to fall because the self.gravity increment itself by 1 every frame, this would make the illusion of a gravity. And for how the player can jump is basically very simple, in input section, we

basically define the self.gravity into -21, hence would create an illusion that the player

```
            self.gravity = 0

    def update(self):
        #Where we call all of the function
        self.player_input()
        self.apply_gravity()
        self.animation_state()
        pass
    def animation_state(self):
        #Where the animation is being handled
        if self.rect.bottom < 500:
            self.image = self.Player_jump
        else:
        #The self.Player_index is being increased by 0.1,
        which is the speed of the animation

            self.Player_index += 0.1
            if self.Player_index  >= len(self.
            Player_standing): self.Player_index =0
            self.image = self.Player_standing[int(self.
            Player_index)%2]
```

- In here, we make the update function and that basically would call of the previous function. For the animation_state function, the way it works first it detect if our player on the ground or not, if not, then it would change the self.image into a jumping animation. If on the ground, then it would allow the idle animation, the way this work is the player index would increment itself and by doing so, will shuffled the animation through the list. The %2 makes sure the player index is either 0 or 1, so we can access the information on our list. The self.Player_index = 0 would ensure that the player index would reset so that the animation list can still be accessed.

```python
class Killer(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()

        # Load Killer standing images using a loop
        self.Killer_standing = [pygame.image.load(f'killer/killer{i}.gif').convert_alpha() for i in range(1, 8)]

        # Initialize Killer index, image, and position
        self.Killer_index = 0
        self.image = self.Killer_standing[self.Killer_index]
        self.rect = self.image.get_rect(midbottom=(1200, 505))

    def display_update(self, screen):
        # Update Killer's animation state and display on the screen
        self.animation_state()
        screen.blit(self.image, (self.rect.x, self.rect.y))
        pygame.display.update()

    def animation_state(self):
        # Update Killer's animation index and image, and reset if necessary
        self.Killer_index = (self.Killer_index + 0.1) % len(self.Killer_standing)
        self.image = self.Killer_standing[int(self.Killer_index)]
```

- The Killer class works the same way as the Player class, but the difference it doesn't have an output, and I make the self.killer_standing more shorter because how many the pygame.image.load is. The way it works is, later than using A variable on by one, I use a for loop and store those pygame.image.load inside a list, and then it works the same way as before.

```python
class Intro(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.Intro_Index = 0
        self.Intro_Images = [pygame.image.load(f'Intro/frame{i}.gif').convert_alpha() for i in range(1, 29)]
        self.Intro_image = self.Intro_Images[self.Intro_Index]

    def animation_state_intro(self):
        self.Intro_Index += 0.1
        if self.Intro_Index >= len(self.Intro_Images):
            self.Intro_Index = 0
        self.Intro_image = self.Intro_Images[int(self.Intro_Index) % 28]

    def display_update(self, screen):
        self.animation_state_intro()
        screen.blit(self.Intro_image, (-400, 0))
        pygame.display.update()
```

- The Intro class which is used for the cinematic scene also uses the same logic.

```
#INITIALIZATIONd
pygame.init()

screen = pygame.display.set_mode((1500, 900))
pygame.display.set_caption('Analog Horor 1.2')
MaximumFrameRate = pygame.time.Clock()
bg_music = pygame.mixer.Sound('sound/scaryAudio.mp3')
bg_music.set_volume(0.5)
snow_music = pygame.mixer.Sound('sound/Snow.mp3')
snow_music.set_volume(0.3)
scary_music_intro = pygame.mixer.Sound('sound/scaryIntro.mp3')
scary_music_intro.set_volume(0.3)
loudMusic = pygame.mixer.Sound('sound/LoudNoise.mp3')
loudMusic.set_volume(0.08)
breathing_sound = pygame.mixer.Sound('sound/Breathing.mp3')
breathing_sound.set_volume(0.6)
killer_revealed_sound = pygame.mixer.Sound('sound/killerReveal.mp3')
killer_revealed_sound.set_volume(0.6)
```

- In here we initialize the pygame.init() function for our game (which is the most important part). Then we define the screen height and width using pygame.displayer.set_mode((1500,900)). We also define the pygame caption, and maximum timeframe here using the pygame.time.Clock() function. Every music sounds are defined here, and the volume sound.

```
bg_music.play(loops=-1)
```

- We then play the Bg_music for the whole game.

```
test_font = pygame.font.Font("font/Mangolaine.ttf",50)
home_screen = pygame.image.load('images/homeScreen3.jpg').convert_alpha()
title_screen = pygame.image.load('images/title.png').convert_alpha()
start_surf = test_font.render('Start',False,'Black').convert_alpha()
start_rect = start_surf.get_rect(center = (200, 700))
quit_surf_button = test_font.render('Quit',False,'Black').convert_alpha()
quit_rect_button = quit_surf_button.get_rect(center = (400, 700))
back_surf = test_font.render('Back',False,'White').convert_alpha()
back_rect = back_surf.get_rect(center = (764, 700))
game_over_font = pygame.font.Font("font/Mangolaine.ttf",200)
game_over_surf = test_font.render('GAMEOVER!',False,'Red').convert_alpha()
game_tutorial_font = pygame.font.Font("font/Mangolaine.ttf",70)
game_tutorial_surf = game_tutorial_font.render('Press Space to Continue',False,'White').convert_alpha()
You_Win_surf = test_font.render('GOOD ENDING', False, 'Green').convert_alpha()
bed_surface = pygame.image.load('images/bed.png').convert_alpha()
bed_rect = bed_surface.get_rect(midbottom = (200,500))
killer_screen_surface = pygame.image.load('images/KillerSurfaceImage.jpg').convert_alpha()
phone_surf = pygame.image.load('images/Phone1.png')
hover_phone = pygame.image.load('images/Phone2.png')
phone_rect = phone_surf.get_rect(midbottom = (1100,430))
table_surf =  pygame.image.load('images/table.png').convert_alpha()
table_rect = table_surf.get_rect(midbottom = (1100,510))
```

- In this section we define most of the game assets using pygame.image.load and the game texts & Fonts by using the pygame.font.Font. Then we use the get.rect function to get the objects and fonts surface, this will be use later to create a collision.

```
def render_back_button(mouse_pos, back_surf, back_rect):
    if back_rect.collidepoint(mouse_pos):
        if pygame.mouse.get_focused():
            back_surf = test_font.render('Back', False, 'Green').convert_alpha()
            back_rect = back_surf.get_rect(center=(764, 700))
            pygame.mouse.set_cursor(*pygame.cursors.diamond)
        else:
            back_surf = test_font.render('Back', False, 'White').convert_alpha()
    else:
        back_surf = test_font.render('Back', False, 'White').convert_alpha()

    return back_surf, back_rect
```

- In this function, we make the logic for the back button hover color logic. How it works is using pygame.mouse.get_focused() we change our back button surface text_font color.

```
Player_Gravity = 0
player = Player()
player_group = pygame.sprite.GroupSingle()
player_group.add(player)
player_group.draw(screen)
player_group.update()
```

- In here we define the Player_gravity and call our Player class. (Using OOP Sprite).

```
loadingtext = pygame.image.load('textbox/loadingtext.png').convert_alpha()
text1 = pygame.image.load('textbox/text1.png').convert_alpha()
text2 = pygame.image.load('textbox/text2.png').convert_alpha()
text3 = pygame.image.load('textbox/text3.png').convert_alpha()
text4 = pygame.image.load('textbox/text4.png').convert_alpha()
text5 = pygame.image.load('textbox/text5.png').convert_alpha()
text6 = pygame.image.load('textbox/text6.png').convert_alpha()
text7 = pygame.image.load('textbox/text7.png').convert_alpha()
text8 = pygame.image.load('textbox/text8.png').convert_alpha()
text9 = pygame.image.load('textbox/text9.png').convert_alpha()
text10 = pygame.image.load('textbox/text10.png').convert_alpha()
text11 = pygame.image.load('textbox/text11.png').convert_alpha()
text12 = pygame.image.load('textbox/text12.png').convert_alpha()
text13 = pygame.image.load('textbox/text13.png').convert_alpha()
text14 = pygame.image.load('textbox/text14.png').convert_alpha()
text15 = pygame.image.load('textbox/text15.png').convert_alpha()
text16 = pygame.image.load('textbox/text16.png').convert_alpha()
text17 = pygame.image.load('textbox/text17.png').convert_alpha()
text18 = pygame.image.load('textbox/text18.png').convert_alpha()
text19 = pygame.image.load('textbox/text19.png').convert_alpha()
text20 = pygame.image.load('textbox/text20.png').convert_alpha()
text21 = pygame.image.load('textbox/text21.png').convert_alpha()
text22 = pygame.image.load('textbox/text22.png').convert_alpha()
text23 = pygame.image.load('textbox/text23.png').convert_alpha()
text24 = pygame.image.load('textbox/text24.png').convert_alpha()
text25 = pygame.image.load('textbox/text25.png').convert_alpha()
text26 = pygame.image.load('textbox/text26.png').convert_alpha()
text27 = pygame.image.load('textbox/text27.png').convert_alpha()
text28 = pygame.image.load('textbox/text28.png').convert_alpha()
text29 = pygame.image.load('textbox/text29.png').convert_alpha()
```

- In here we load all of the images for our Visual Novels TextBox

```
decision1 = pygame.image.load('textbox/decision1.png').convert_alpha()
decision2 = pygame.image.load('textbox/decision2.png').convert_alpha()
decision1_rect = decision1.get_rect(topleft = (130, 400))
decision2_rect = decision2.get_rect(topleft = (870, 400))

decision3 = pygame.image.load('textbox/decision3.png').convert_alpha()
decision4 = pygame.image.load('textbox/decision4.png').convert_alpha()
decision3_rect = decision3.get_rect(topleft = (130, 400))
decision4_rect = decision4.get_rect(topleft = (870, 400))

decision5 = pygame.image.load('textbox/decision5.png').convert_alpha()
decision6 = pygame.image.load('textbox/decision6.png').convert_alpha()
decision5_rect = decision5.get_rect(topleft = (130, 400))
decision6_rect = decision6.get_rect(topleft = (870, 400))

decision7 = pygame.image.load('textbox/decision7.png').convert_alpha()
decision8 = pygame.image.load('textbox/decision8.png').convert_alpha()
decision7_rect = decision7.get_rect(topleft = (300, 200))
decision8_rect = decision8.get_rect(topleft = (800, 200))
```

- In here we load all of the images for our decision boxes and get their rectangle for the images for collision purposes.

```
white = (255,255,255)
black = (0,0,0)
```

- Defining Color Variable that's going to be used later

```
run = True
game_active = False
level_1 = False
level_2 = False
level_3 = False
level_4 = False
level_5 = False
```

- In here we define the game states and condition in the variable. These are used to basically control what they game are going to display and what's not going to be displayed. How it works basically it's going to down from one state to another, meaning from the run state being true, into level_5 being true.

```
phone_activated = False
phone_activated2 = False
phone_activated3 = False
phone_activated4 = False
phone_activated5 = False
phone_activated6 = False
phone_activated7 = False
downstair_activated = False
downstair_activated2 = False
downstair_activated3 = False
downstair_activated4 = False
downstair_activated5 = False
downstair_activated6 = False
downstair_activated7 = False
downstair_activated8 = False
downstair_activated9 = False
downstair_activated10 = False
downstair_activated11 = False
downstair_activated12 = False
downstair_activated13 = False
downstair_activated14 = False
downstair_activated15 = False
Introduction = False
```

- In here we define the game states for our Visual Novel stories, these variables are used to control the textboxes. The logic will be shown later in the For Event in the while loop.

```
phone_hovered = False
Killer_activated = False
active = True
active2 = True
active3 = True
canJump = True
game_over = False
game_win = False
```

- The activate, activate2, and activate3 are used for debugging in the code. The Phone hovered is used to make the phone surface animation while the Killer_activated is to control the killer image movement. And the rest of the variable are used for debugging.

```
text_box1 = True
text_box2 = False
text_box3 = False
text_box4 = False
text_box5 = False
text_box6 = False
text_box7 = False
text_box8 = False
text_box9 = False
text_box10 = False
text_box11 = False
text_box12 = False
text_box13 = False
text_box14 = False
text_box15 = False
text_box16 = False
```

- The text box variable serves the same purpose as the variable down_activated and phone_activated, as to control the visual novel text box.

```
intro_sprite = Intro()
killer = Killer()
```

- In here we call the class function for the killer class and the Intro Class.

```
firstCount = 0
```

- The firstCount is used later on to count how long the duration of the Introduction cutscene is. (Later Pygame While main loop).

```
phone_rect = phone_surf.get_rect(midbottom=(1100, 430))
hover_phone_rect = hover_phone.get_rect(bottomleft=phone_rect.bottomleft)
```

- This is to define the phone images and animation.

```
#GAME LOOP
while True:···
```

- This is the for the main pygame loop. Next I'm going to explain what's inside it.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        exit()
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_ESCAPE:
            pygame.quit()
            exit()
```

- This For event ensure to check all of the event in pygame. And the if statement below is for exiting the game by tracking Player's Input.

```
    if event.type == pygame.MOUSEBUTTONDOWN:
        if start_rect.collidepoint((event.pos)) and run:
            scary_music_intro.play()
            time.sleep(1)
            game_active = True
            run = False
            print(run, game_active)
            bg_music.stop()
        elif quit_rect_button.collidepoint((event.pos)) and run:
            time.sleep(0.1)
            pygame.quit()
            exit()
```

- These for event is for the start button. When a collision happen between the start button and the mouse, a music will play, while changing the game_active to true in turn starting the game while the run into false. And the quit one for quitting the game.

```
    exit()
elif back_rect.collidepoint(event.pos) and (text_box15 or phone_activated7 or downstair_activated6 or level_4 or level_5):
    player.rect.x = 200
    run = True
    text_box15 = False
    phone_activated7 = False
    downstair_activated6 = False
    downstair_activated7 = False
    downstair_activated8 = False
    downstair_activated9 = False
    downstair_activated10 = False
    downstair_activated11 = False
    downstair_activated12 = False
    downstair_activated13 = False
    downstair_activated14 = False
    downstair_activated15 = False
    level_4 = False
    level_5 = False
    text_box1 = True
    active = True
    canJump = True
```

- This if statement is for the back button, it will reset all of the game variables so the player can play the game again.

```
    elif phone_rect.collidepoint((event.pos)) and level_1:
        if active:
            phone_activated = True
```

- This will activated the phone cutscene / storypath.

```
if event.type == pygame.MOUSEMOTION:
    mouse_pos = event.pos
    phone_rect = hover_phone.get_rect(midbottom=(1100, 430))
    phone_hovered = phone_rect.collidepoint(mouse_pos)
```

- This is for the Telephone image animation.

```
if event.type == pygame.KEYDOWN and Introduction:
    if event.key == pygame.K_SPACE:
        time.sleep(0.2)
        if text_box1:
            text_box1 = False
            text_box2 = True
        elif text_box2:
            text_box2 = False
            text_box3 = True
        elif text_box3:
            text_box3 = False
            text_box4 = True
```

- This section is the reason why there's so many Boolean variable declared previously, basically this is how the logic works. Everytime a player press a space bar, it would switch / cycle through the next Boolean variable, making the previous variable value into false and making the next one into True. The code for the text_box is actually longer, it cycle until text_box16, but Screenshotting here from text_box1 would be too long.

```
        elif text_box16:
            Introduction = False
            level_1 = True
```

- Once reached the end of the cycle, it would move to the next game states.

```
if event.type == pygame.KEYDOWN and level_1:
    if event.key == pygame.K_SPACE:
        time.sleep(0.2)
        if phone_activated:
            active = False
            phone_activated2 = True
            phone_activated = False
        elif phone_activated3:
            phone_activated4 = True
            phone_activated3 = False
        elif phone_activated4:
            phone_activated5 = True
            phone_activated4 = False
        elif phone_activated5:
            phone_activated6 = True
            phone_activated5 = False
        elif phone_activated6:
            phone_activated7 = True
            phone_activated6 = False
```

- Works the same as before. For our phone storypath / line.

```
if event.type == pygame.KEYDOWN and level_2:
    if event.key == pygame.K_SPACE:
        time.sleep(0.2)
        if downstair_activated:
            downstair_activated = False
            downstair_activated2 = True
        elif downstair_activated3:
            downstair_activated3 = False
            downstair_activated4 = True
        elif downstair_activated4:
            downstair_activated4 = False
            downstair_activated5 = True
        elif downstair_activated5:
            downstair_activated5 = False
            downstair_activated6 = True
        elif downstair_activated7:
            downstair_activated7 = False
            downstair_activated8 = True
        elif downstair_activated8:
            downstair_activated8 = False
            downstair_activated9 = True
        elif downstair_activated9:
            player.rect.x = 200
            downstair_activated9 = False
            level_2 = False
            level_3 = True
```

- Works the same way but for our storyline when Timmy goes downstairs.

```
if event.type == pygame.KEYDOWN and level_4:
    if event.key == pygame.K_SPACE:
        time.sleep(0.2)
        if downstair_activated10:
            downstair_activated10 = False
            downstair_activated11 = True
        elif downstair_activated11:
            downstair_activated11 = False
            downstair_activated15 = True
if event.type == pygame.KEYDOWN and level_5:
    if event.key == pygame.K_SPACE:
        time.sleep(0.2)
        if downstair_activated12:
            downstair_activated12 = False
            downstair_activated13 = True
        elif downstair_activated13:
            downstair_activated13 = False
            downstair_activated14 = True
```

- For our decision storypath.

```
    if event.type == pygame.MOUSEBUTTONDOWN:
        # Get the mouse button state
        mouse_buttons = pygame.mouse.get_pressed()
        if mouse_buttons[0] == 1:
            # Check if the click is within the decision1_rect area
            if decision1_rect.collidepoint(pygame.mouse.get_pos()) and text_box6:
                time.sleep(0.1)
                text_box7 = True
                text_box6 = False
                # Add your logic here
            elif decision2_rect.collidepoint(pygame.mouse.get_pos()) and Introduction:
                text_box16 = True
                text_box6 = False
                # Add your logic here
            elif decision3_rect.collidepoint(pygame.mouse.get_pos()) and phone_activated2:
                phone_activated = False
                phone_activated2 = False
                active = True
                canJump = True
                player.rect.x = 1000
                # Add your logic here
```

```
            elif decision4_rect.collidepoint(pygame.mouse.get_pos()) and phone_activated2:
                phone_activated3 = True
                phone_activated2 = False
                canJump = False
            elif decision5_rect.collidepoint(pygame.mouse.get_pos()) and downstair_activated2:
                downstair_activated7 = True
                downstair_activated2 = False
                player.rect.x = 200
                killer.rect.x = 1200

            elif decision6_rect.collidepoint(pygame.mouse.get_pos()) and downstair_activated2:
                downstair_activated3 = True
                downstair_activated2 = False
            elif decision7_rect.collidepoint(pygame.mouse.get_pos()) and active2 and player.rect.x >= 1200:
                print('1')
                player.rect.x = 300
            elif decision8_rect.collidepoint(pygame.mouse.get_pos()) and active2 and player.rect.x >= 1200:
                print('2')
                level_1 = False
                level_2 = True
                downstair_activated = True
                active2 = False
```

- These if statement inside the for event are for the decision boxes, so when the game detect a collision between the decision box and the mouse clicked, it would changes the game states and other variable.

```
if run:
    firstCount = 0
    Hello = pygame.draw.rect(screen, 'Black', pygame.Rect(0, 0, 1500, 900))
    screen.blit(home_screen, (0, 0))
    screen.blit(title_screen, (130, 50))
    pygame.draw.rect(screen, white, (128, 650, 160, 100),border_radius=10)
    pygame.draw.rect(screen, white, (325, 650, 160, 100),border_radius=10)
    screen.blit(quit_surf_button, quit_rect_button)
    mouse_pos = pygame.mouse.get_pos()
    if start_rect.collidepoint((mouse_pos)):
        if pygame.mouse.get_focused():
            start_surf = test_font.render('Start',False,'Orange').convert_alpha()
            start_rect = start_surf.get_rect(center = (200, 700))
            pygame.draw.rect(screen, 'Orange', start_rect, 100, 10)
            pygame.mouse.set_cursor(*pygame.cursors.diamond)
        else:
            start_surf = test_font.render('Start',False,'Black').convert_alpha()

    elif quit_rect_button.collidepoint((mouse_pos)):
        if pygame.mouse.get_focused():
            quit_surf_button = test_font.render('Quit',False,'Orange').convert_alpha()
            quit_rect_button = quit_surf_button.get_rect(center = (400, 700))
            pygame.mouse.set_cursor(*pygame.cursors.diamond)
        else:
            quit_surf_button = test_font.render('Quit',False,'Black').convert_alpha()
    else:
        if pygame.mouse.get_focused():
            start_surf = test_font.render('Start',False,'Black').convert_alpha()
            quit_surf_button = test_font.render('Quit',False,'Black').convert_alpha()
            pygame.mouse.set_cursor(*pygame.cursors.arrow)
    pygame.draw.rect(screen, 'White', start_rect, 100, 10)
    screen.blit(start_surf,start_rect)
```

- This is the first game states for our while loop python game. The first one is called Run, and on this game state, all of it does is display the main menu with the start and quit button, with the game background and title screen.

```
elif game_active:
    Intro_duration = 300
    start_time = pygame.time.get_ticks()
    intro_sprite.display_update(screen)
    snow_music.play(loops=0)
    firstCount += 1
    print(firstCount)
    if firstCount >= Intro_duration:
            print('Level 1')
            Introduction = True
            snow_music.stop()
            game_active = False
```

- This one for signaling that the game has started, and will show the cinematic intro scene. During the game_active, a music will play and a variable called firstCount would increment itself by one (Because the pygame frame is 60fps, it means that the firstCount would increment itself by 60 every seconds) per frame. If the

firstCount value is bigger than the Intro_duration then the game_activate state would stop and move to the next game state.

```
elif Introduction:
    screen.fill('Black')
    aKey = pygame.key.get_pressed()
    if text_box1:
        screen.blit(game_tutorial_surf, (430, 250))
        screen.blit(text1, (280, 400))
    elif text_box2:
        screen.blit(text2, (280, 400))
    elif text_box3:
        screen.blit(text3, (280, 400))
    elif text_box4:
        screen.blit(text4, (280, 400))
        loudMusic.play(loops=0)
    elif text_box5:
        screen.blit(text5, (280, 400))
    elif text_box6:
        screen.blit(decision1, (130, 400))
        screen.blit(decision2, (870, 400))
        # Outside the event loop
        mouse_buttons = pygame.mouse.get_pressed()
    elif text_box7:
        screen.blit(text6, (280, 400))
    elif text_box8:
        screen.blit(loadingtext, (280, 400))
    elif text_box9:
        breathing_sound.play(loops=1)
        screen.blit(text7, (280, 400))
```

```
    elif text_box9:
        breathing_sound.play(loops=1)
        screen.blit(text7, (280, 400))
    elif text_box10:
        breathing_sound.stop()
        killer_revealed_sound.play(loops=0)
        screen.blit(killer_screen_surface,(170,100))
    elif text_box11:
        screen.blit(killer_screen_surface,(170,100))
        screen.blit(loadingtext, (280, 400))
    elif text_box12:
        screen.blit(text9, (280, 400))
    elif text_box13:
        screen.blit(text10, (280, 400))
    elif text_box14:
        screen.blit(text11, (280, 400))
    elif text_box15:
        mouse_pos10 = pygame.mouse.get_pos()
        mouse_pos10 = pygame.mouse.get_pos()
        if back_rect.collidepoint(mouse_pos10):
            if pygame.mouse.get_focused():
                back_surf = test_font.render('Back', False, 'Red').convert_alpha()
                back_rect = back_surf.get_rect(center=(764, 700))
                pygame.mouse.set_cursor(*pygame.cursors.diamond)
            else:
                back_surf = test_font.render('Back', False, 'White').convert_alpha()
        else:
            back_surf = test_font.render('Back', False, 'White').convert_alpha()
            #draw the back rect
        #position the game over surf in the middle
        screen.blit(game_over_surf, (630, (screen.get_height() / 2) - 20 ))
        screen.blit(back_surf, back_rect)
    elif text box16:
```

The Introduction is where the game display the story through the text and this is where the where the For Event if statement from above comes to hand and handle all

of the logic. Basically, every time= a spacebar is pressed, it will navigate through these if and elif statements.

```
elif level_1:
    screen.fill('Black')
    active2 = True
    screen.blit(Floor_1, (0, 500))
    screen.blit(bed_surface, bed_rect)
    screen.blit(table_surf,table_rect)
    mouse_posX = pygame.mouse.get_pos()
    if active:
        if phone_hovered:
            screen.blit(hover_phone, hover_phone_rect)
        else:
            screen.blit(phone_surf, phone_rect)
    player_group.draw(screen)
    player_group.update()
    Player_Gravity += 1
    if active2:
        if player.rect.x >= 1200:
            screen.blit(text18, (280, 400))
            screen.blit(decision7, (300, 200))
            screen.blit(decision8, (800, 200))
    if player.rect.x >= 1350:
        player.rect.x = 1350
    if phone_activated:
        canJump = False
        screen.fill('Black')
        screen.blit(text13, (280, 400))
    elif phone_activated2:
        screen.fill('Black')
        screen.blit(decision3, (130, 400))
        screen.blit(decision4, (870, 400))
    elif phone_activated3:
```

```
    screen.blit(decision4, (870, 400))
elif phone_activated3:
    screen.fill('Black')
    screen.blit(text14, (280, 400))
elif phone_activated4:
    screen.fill('Black')
    screen.blit(text15, (280, 400))
elif phone_activated5:
    screen.fill('Black')
    screen.blit(text16, (280, 400))
elif phone_activated6:
    screen.fill('Black')
    screen.blit(text17, (280, 400))
elif phone_activated7:
    screen.fill('Black')
    mouse_pos100 = pygame.mouse.get_pos()
    if back_rect.collidepoint(mouse_pos100):
        if pygame.mouse.get_focused():
            back_surf = test_font.render('Back', False, 'Green').convert_alpha()
            back_rect = back_surf.get_rect(center=(764, 700))
            pygame.mouse.set_cursor(*pygame.cursors.diamond)
        else:
            back_surf = test_font.render('Back', False, 'White').convert_alpha()
    else:
        back_surf = test_font.render('Back', False, 'White').convert_alpha()
    screen.blit(You_Win_surf, (630, (screen.get_height() / 2) - 20 ))
    screen.blit(back_surf, back_rect)
```

The level 1 is where the 2D gameplay aspect came to play. Drawing the player, ground, phone, and bed is all done in his game states. This is where all of the player gravity and mobility from the Player class came to play, and on this state depending on the player decision, the story narration would continue using the text boxes.

```
    elif level_2:
        screen.fill('Black')
        if downstair_activated:
            screen.blit(text19, (280, 400))
        elif downstair_activated2:
            screen.blit(decision5, (130, 400))
            screen.blit(decision6, (870, 400))
        elif downstair_activated3:
            screen.blit(text20, (280, 400))
        elif downstair_activated4:
            screen.blit(text21, (280, 400))
        elif downstair_activated5:
            screen.blit(text22, (280, 400))
        elif downstair_activated6:
            mouse_posx2 = pygame.mouse.get_pos()
            back_surf, back_rect = render_back_button(mouse_posx2, back_surf, back_rect)
            screen.blit(game_over_surf, (630, (screen.get_height() / 2) - 20 ))
            screen.blit(back_surf, back_rect)
                #draw the back rect
            #position the game over surf in the middle
            screen.blit(game_over_surf, (630, (screen.get_height() / 2) - 20 ))
            screen.blit(back_surf, back_rect)
        elif downstair_activated7:
            screen.blit(text23, (280, 400))
        elif downstair_activated8:
            screen.blit(text24, (280, 400))
        elif downstair_activated9:
            screen.blit(text25, (280, 400))
```

This code is to display the textbox narration when the player chooses to go downstair. (If the player mouse is detected pressed and collided with the decision8 button rectangle.) The code then will display the game over screen, and the back button if the player chooses a specific story path.

```
elif level_3:
    screen.fill('Black')
    screen.blit(Floor_1, (0, 500))
    player_group.draw(screen)
    player_group.update()
    Player_Gravity += 1
    killer.display_update(screen)
    if player.rect.x >= 1350:
        player.rect.x = 1350
    if player.rect.x <= 0:
        player.rect.x = 0
    if player.rect.x >= 400 or player.rect.x <= 30:
        Killer_activated = True
    if Killer_activated:
        killer.rect.x -= 15
    if player.rect.colliderect(killer.rect):
        level_3 = False
        level_4 = True
        player.rect.x = 200
        downstair_activated10 = True
        Killer_activated = False
    if killer.rect.x <= 0:
        level_5 = True
```

```
level_3 = False
downstair_activated12 = True
player.rect.x = 200
Killer_activated = False
```

This code is to display the textbox narration when the player chooses to go downstair and investigate the noise. In here we call the player class and killer class by:

player_group.draw(screen) and killer.display_update(screen). In here we detect the player rectangle x and y for the gameplay. First, our player can not go off the map using the if statement below:

```
if player.rect.x >= 1350:
    player.rect.x = 1350
if player.rect.x <= 0:
    player.rect.x = 0
```

Then using the code below, we check our player rectangle x location, if its at 400 or 30, the killer_activated variable will return True and if so, the killer images / rectangle will moves to the left.

```
if player.rect.x >= 400 or player.rect.x <= 30:
    Killer_activated = True
if Killer_activated:
    killer.rect.x -= 15
```

```
if player.rect.colliderect(killer.rect):
    level_3 = False
    level_4 = True
    player.rect.x = 200
    downstair_activated10 = True
    Killer_activated = False
if killer.rect.x <= 0:
    level_5 = True
    level_3 = False
    downstair_activated12 = True
    player.rect.x = 200
    Killer_activated = False
```

This code will detect a collision with between the rectangle of our player and the killers, if it detects a collision between them, then it would moves to the next game states by turning the level_3 Boolean value into False, and the level_4 Boolean Value into True, and also the killer activated into false so it would stop moving, and downstair_activated10 Boolean value into True for controlling the narration text box later on. If the we didn't detect any collision and the killer goes off the rail then the same thing happen, but different Boolean Value is being changed because we're going

into another different game states

```
elif level_4:
    screen.fill('Black')
    if downstair_activated10:
        screen.blit(text28, (280, 400))
    elif downstair_activated11:
        screen.blit(text29, (280, 400))
    elif downstair_activated15:
        mouse_posx2 = pygame.mouse.get_pos()
        back_surf, back_rect = render_back_button(mouse_posx2, back_surf, back_rect)
        screen.blit(game_over_surf, (630, (screen.get_height() / 2) - 20 ))
        screen.blit(back_surf, back_rect)
```

The code will display the textbox / narrate the story after the player rectangle collide with the killer rectangle, triggering a bad ending then a display the game over screen with the back button.

```
elif level_5:
    screen.fill('Black')
    if downstair_activated12:
        screen.blit(text26, (280, 400))
    elif downstair_activated13:
        screen.blit(text27, (280, 400))
        mouse_posx2 = pygame.mouse.get_pos()
    elif downstair_activated14:
        mouse_posx2 = pygame.mouse.get_pos()
        back_surf, back_rect = render_back_button(mouse_posx2, back_surf, back_rect)
        screen.blit(You_Win_surf, (630, (screen.get_height() / 2) - 20 ))
        screen.blit(back_surf, back_rect)
```

Finally, the last code level_5 will be triggered if the player dodge the killer and the killer rect.x can be detected located at <= 0. The code will display another textbox to narrate the story for the good ending, hence displaying a You Win screen this time with the back button.

```
pygame.display.update()
MaximumFrameRate.tick(60)
```

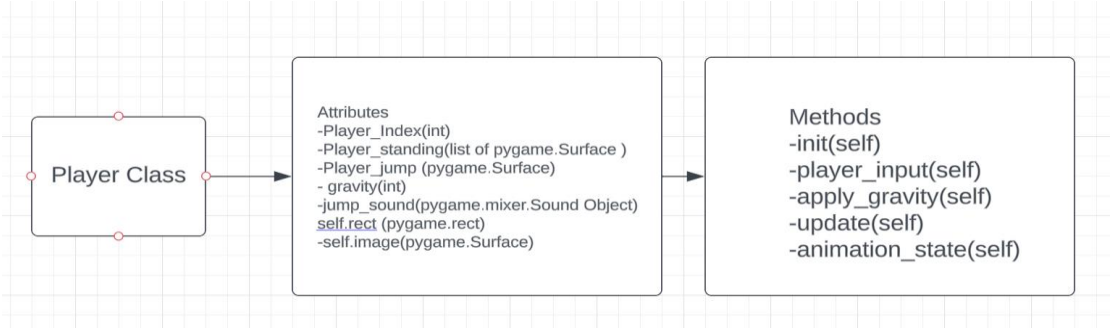
In here, we update the game surface using the pygame.display.update().

Then we set the frame into 60 Frames per second, meaning our surfaces updated every 60 seconds.

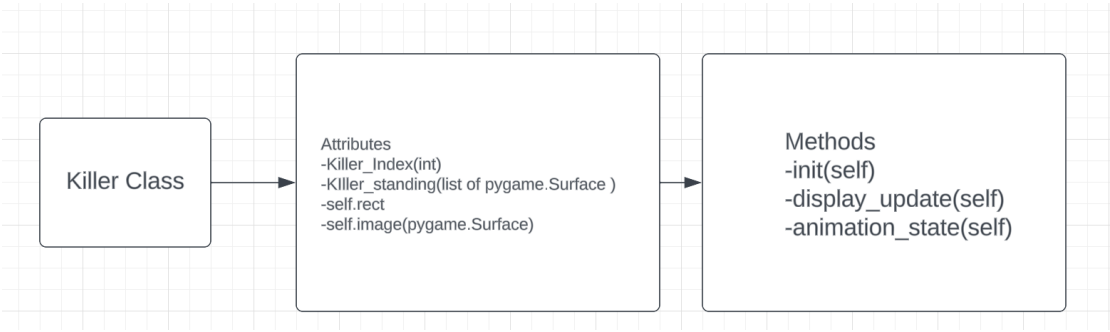
This will activated the phone cutscene / storypath.

D. Diagrams

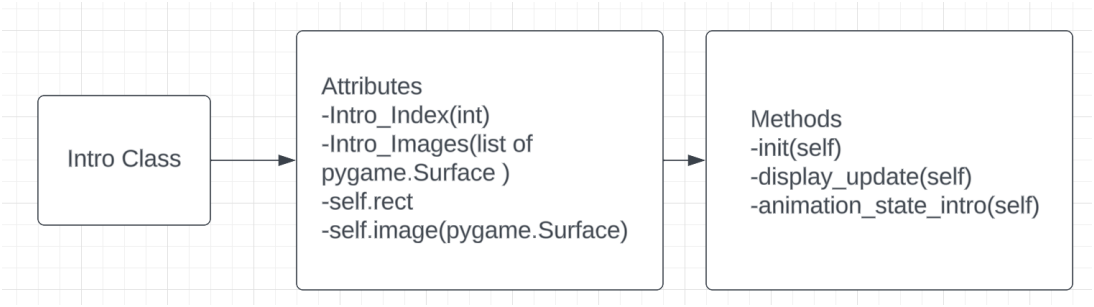
Player Class Diagram



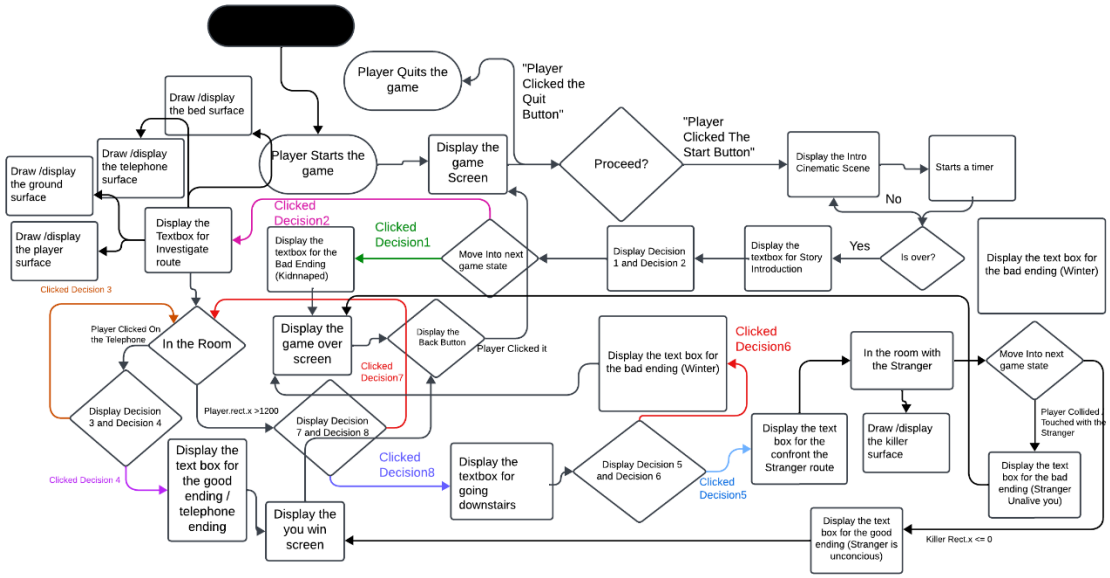
Killer Class Diagram



Intro Class Diagram:



Activities Diagram (https://lucid.app/lucidspark/e1faf0e4-75fb-4401-9c54-e3e17045a9a2/edit?viewport_loc=-433%2C-354%2C3454%2C1795%2C0_0&invitationId=inv_daaa1eb6-e739-4836-b212-768857cdd4e3):



E. Lesson / Learned

This Project is by far the hardest thing I've ever done so in this semester, even more so then the HCI Final Project and I kind of enjoyed the journey. I'm not that knowledgeable in Python, I haven't done learning on my own because I've been to busy learning JavaScript for the HCI Final Project, so this tackling this Algorithm Programming Final Project was quite a challenge. Before doing this project, I watch a 3 Hours Youtube video regarding Pygame basics on the logic and how it works, that's it, so after watching and knowing the Pygame basics I have to figured out how to make a Visual Novel game because they didn't mention how it works in the video. Before tackling the final project, I also did quickly scheme to all of the materials I've done in the Algo Pro course so I can be prepared as possible for the Final Project. It takes me 6 days to both code the games and also create the assets for the game, and I just realize how hard game dev actually is. One of the hardest parts of the Final

Project other then the coding is actually creating the game assets and designing the game plot and story. It takes me a whole day to design some of the pixel arts in the game (Because some of them have animation), and to design the Textboxes (And to create the stories too).

After done doing all of that, I spent 5 days focused on coding and debugging the game, which is the most stressful part. I never realizes how much bug can be on a videogame, and debugging actually one of the reason this final project has been very stressful (Hard to adjust to the pygame FPS mechanic). After days of stressful debugging, I finally cleaned up the code because my code previously was a total mess / spaghetti code. I organize the code, create a function to stop repeatedly using the same code, add comments, etc.

Even though very challenging and stressful, I actually really enjoyed this project and the challenges it provides. The days of debugging have really improved my skills in programming and trained me to learn how to analyze a code and tried to debug it. This Project also really trained me on my problem-solving skills. After this project, I have a huge respect on game developers, because game-dev is definitely is not easy.

F. Screenshots of the Project

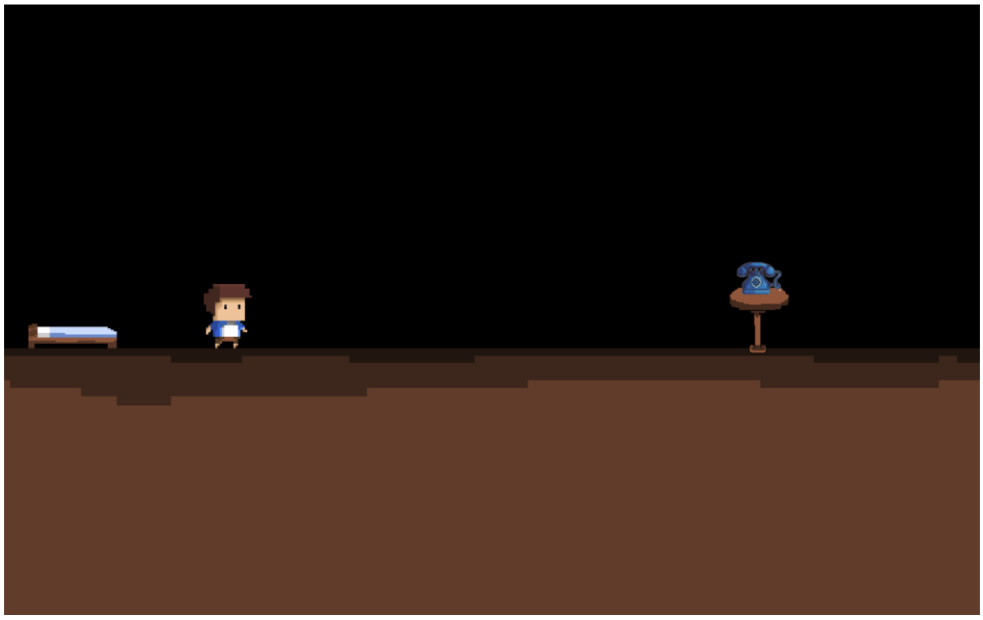
Game starting / menu screen:

Intro cinematic scene:

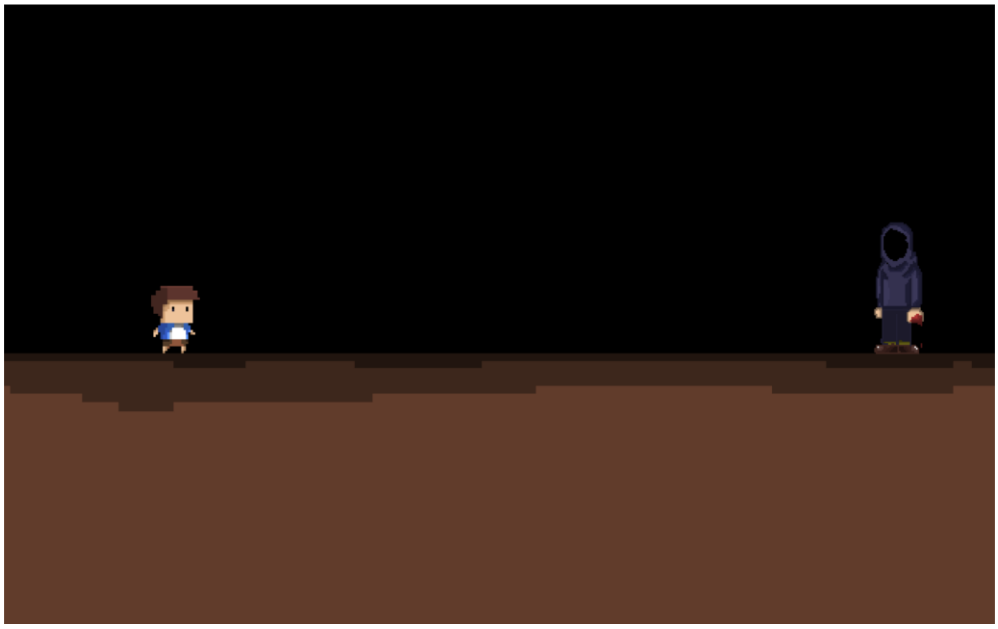
Story narration (Visual Novel Aspect):

Player choosing story path:

2D Game Aspect – Player in a room



Player Confronting the killer:



G. Resources:

Learning Pygame:

- https://www.pygame.org/docs/index.html
- https://www.youtube.com/watch?v=AY9MnQ4x3zk&t=12383s&ab_channel=ClearCode

Assets Creation:

- https://leonardo.ai/
- https://www.canva.com/