Final Project Report

Algorithm and Programming

Computer Science Program
School Of Computing and Creative Arts

Course Code : COMP6047001

Course Name : Algorithm and Programming

Lecturer : Jude Joseph Lamug Martinez, MCS

Name : Kenny Tang

Student ID : 2802517733

Class : L1BC

Bina Nusantara International University
Jakarta
2024

A. Introduction

In this project, I have developed a tower defense game using python . I always wanted to learn a programming language and python happens to be my first programming language. After learning python, I want to improve my python programming skills by doing projects.

I have always liked playing games in my free time and the idea of playing games makes me think of making a game myself. Therefore, I decided to make a game in python. I think that building a game allows me to explore programming concepts in a fun and interactive way.

When it comes to deciding what type of game to make, I have a lot of game ideas that I wanted to make but in the end I decided to make a Tower defense game. I am inspired by the tower defense game called Bloons TD 6. Which is the reason all of the assets I make look very familiar to the Bloons TD 6 game.

B. Project Specification

1. Project Description

This game is called *Bloons Pygame*. It is very similar to a normal tower defense game where the objective is to prevent enemies from traveling a design path and reaching the path endpoint. Then players place turrets strategically to prevent the enemies from reaching the endpoint.

The difference is *Bloons Pygame* enemies are balloons and the turrets are monkeys. The challenge of this game is for players to decide when to manage money, when to upgrade turrets and where to place them to handle the balloons which will get increasingly difficult the higher the waves .

The goal of this game is to stop all balloons from reaching the endpoint and survive 20 waves .

2. Tools / Modules

➔ **Pygame**

Pygame is a Python module that is designed to make video games. I used pygame to make the game.

➔ **Aseprite**

Aseprite is a software program that is specialized on 2D pixel art. I used Aseprite to draw all the assets that are used in the game such as buttons, background, monkey, balloons, and more.

➔ **Youtube**

Youtube is an online platform where users can share and watch videos. I used youtube to download free music and sound effects for the game.

➔ **dafont.com**

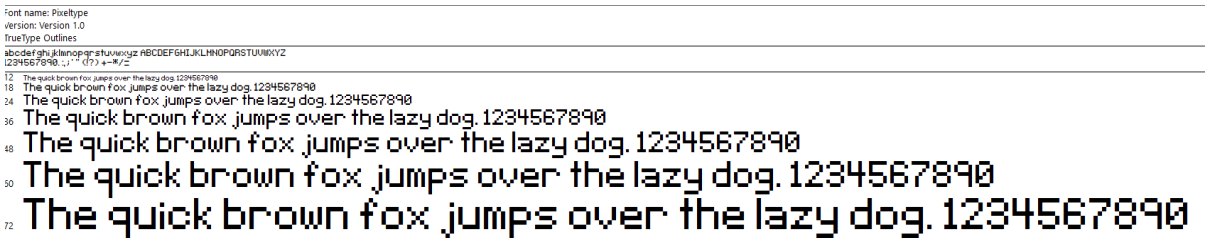
dafont.com is a website where users can download fonts. I used this website to download the fonts and use it in this project.

3. Game Assets

- **Font**

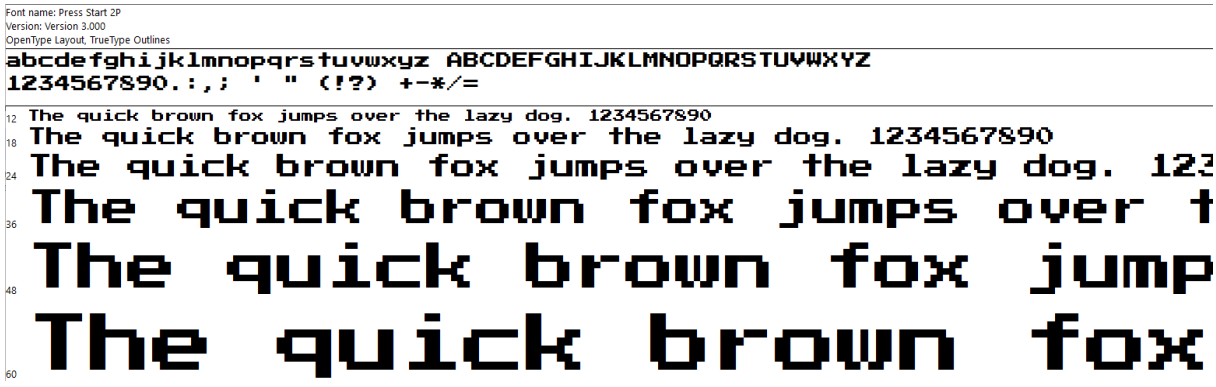
The First Font : pixeltype

Source : https://www.dafont.com/pixeltype.font



The Second Font : PressStart2P

Source : https://www.dafont.com/press-start-2p.font



- **Sound**
  - Game Music
  Source : Jazz Theme
  - Pop Sound Effects
  Source : Bloons TD 6 - Bloon Pop Sound Effect

- **Images**
  - Balloons
  - Balloons Red

  - Balloons Blue

  - Balloons Green

  - Balloons Yellow

  Source : Made by myself Aseprite (original Aseprite file is in github)

  - Game Menu / Buttons
  - Begin Button

  - Fast Forward Button

  - Buy Monkey Button

  - Restart Game Button

- Coin Image

- Health Image

- Music On Image

- Music Off Image

- Upgrade Monkey Button

- Sell Monkey Button

- Play Button

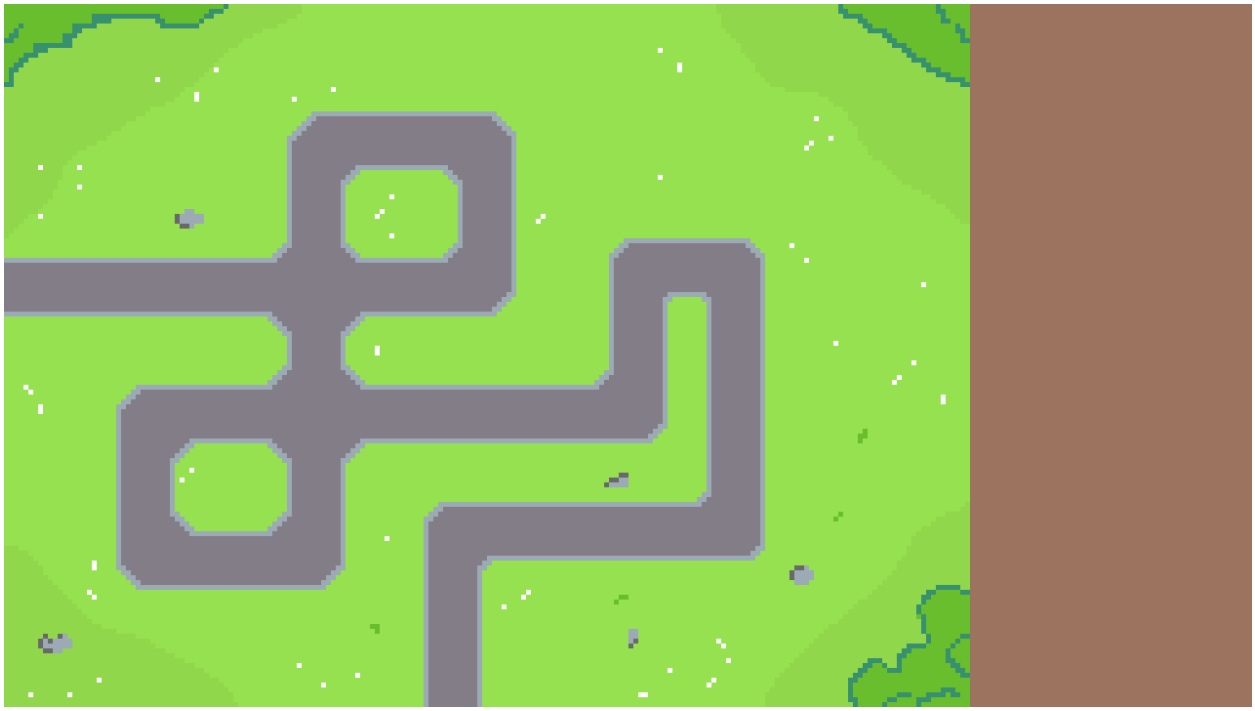
- Cancel Select Button

Source : Made by myself using Aseprite (original Aseprite file is in github)

- Background / Map

- Main Menu Screen

- Game Screen



Source : Made by myself using Aseprite (original Aseprite file is in github)

- Monkey and Animation

- Monkey

- Monkey Animation

Source : Made by myself Aseprite (original Aseprite file is in github)

4. Solution Design

○ Problem Overview

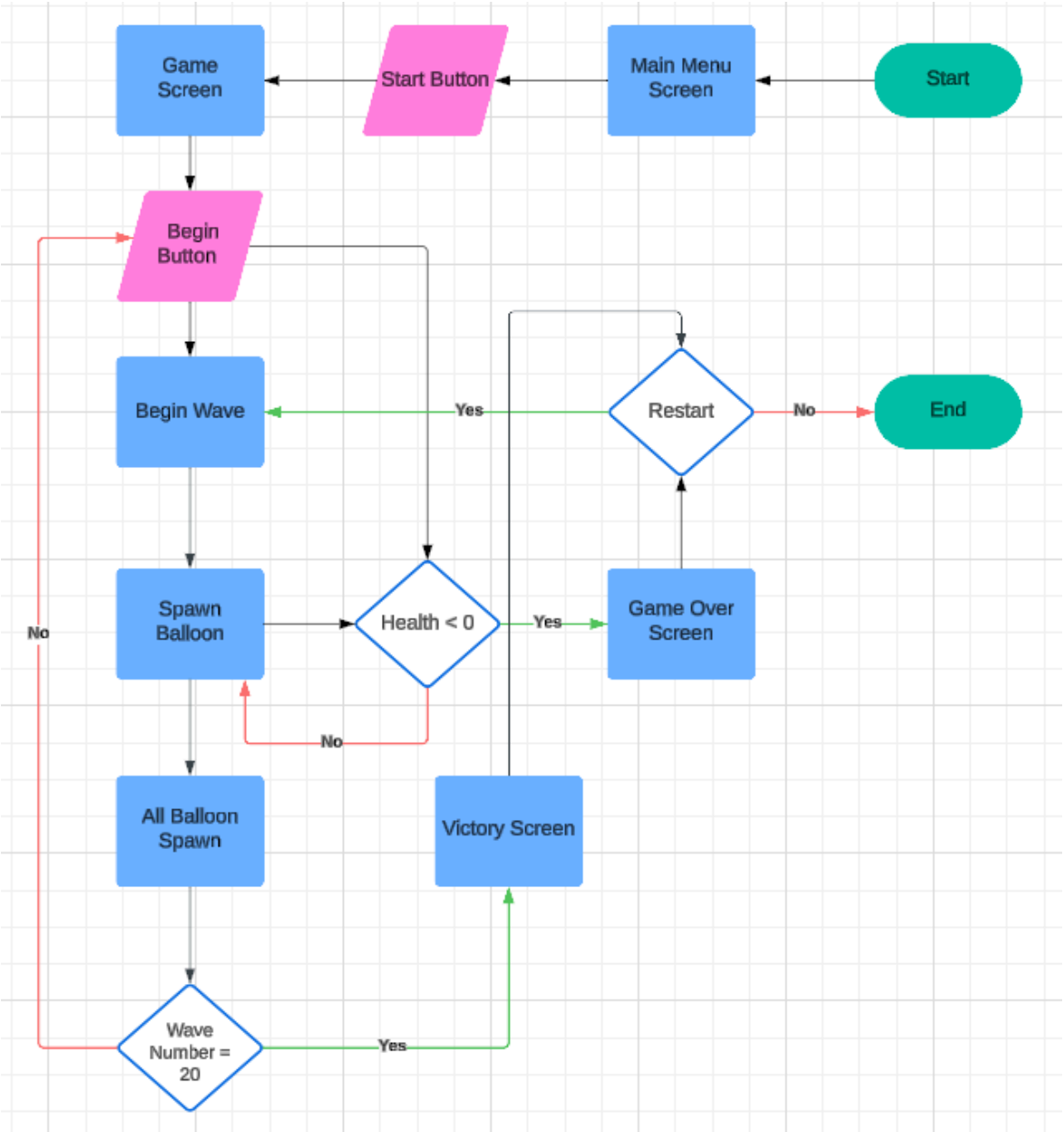
Making a Tower defense game in which players place monkeys along the path to stop balloons from reaching the endpoint. The monkey will shoot darts at the balloons when entering the monkey range, and the player must strategically place the monkey to defend against the balloons which gets harder the higher the waves.

○ Solution Overview

The game will be developed using Pygame in Python. Pygame has the tools for rendering graphics, handling user input and managing the game loop. The core components of the game include:

- **Monkeys** : Monkeys can be placed along the path to shoot darts at the balloons.
- **Balloons** : Balloons will follow a path and will try to reach the endpoint. Different types of balloons will be added the higher the wave, each with different speeds and health.
- **Path**: A path where balloons travel and monkeys are placed strategically to prevent them from reaching the endpoint path.
- **Game Mechanics**: wave completion, player health, money for placing and upgrading monkeys, and detection for balloon hits.
- **User Interface**: Displays player healths, money, wave progression, and allows players to place monkeys.

○ Game Walkthrough (Flowchart made using Lucidchart)



5. Detailed Code Explanation

- *game.py*

```
import pygame
from monkey import Monkey
from balloons import Balloons
from map import Map
from monkeyMenu import MonkeyMenu

```

Importing all the necessary modules to the main module which include Monkey class from monkey.py, Balloons class from Balloons.py, Map class from map.py, MonkeyMenu class from MonkeyMenu.py.

```
# --- Constants ---
WIDTH, HEIGHT = 1280, 720
FPS = 60
CAPTION = "Tower Defense Window"
FONT_SIZE = 50
POP_SOUND_VOLUME = 0.5
MUSIC_VOLUME = 0.3
BUY_MONKEY_COST = 150
UPGRADE_MONKEY_COST = 100
SELL_MONKEY_BASE_COST = 120
SELL_MONKEY_LVL2_COST = 200
SELL_MONKEY_LVL3_COST = 280
SELL_MONKEY_LVL4_COST = 360
SPAWN_COOLDOWN = 500
MESSAGE_DURATION = 2000
TOTAL_WAVE = 20
GAME_OUTCOME = 0
LOSE = -1
WIN = 1
GAME_PATH = [
    (-50, 280), (490, 280), (490, 135), (320, 135), (320, 565),
    (150, 565), (150, 415), (650, 415), (650, 260),
    (750, 260), (750, 530), (460, 530), (460, 750)
]
```

Initializing all the constants that are needed for the game :

- WIDTH, HEIGHT is the size of the window screen in pixel

- FPS is Frames per second of the game
- CAPTION is The title for the game
- FONT_SIZE is font size for all the text in the game
- POP_SOUND_VOLUME is the volume for the pop sound effect
- MUSIC_VOLUME is the volume for game music
- BUY_MONKEY_COST is the cost to purchase the monkey
- UPGRADE_MONKEY_COST is the cost to upgrade the monkey to a higher level
- SELL_MONKEY_BASE_COST, SELL_MONKEY_LVL2_COST, SELL_MONKEY_LVL3_COST, SELL_MONKEY_LVL4_COST are the amount of money you get from selling the monkey
- SPAWN_COOLDOWN is the cooldown in microsecond for the balloons to spawn
- MESSAGE_DURATION is the cooldown in microsecond for the text to appear
- TOTAL_WAVE is the number of wave
- GAME_OUTCOME is a constant to decide the state of the game whether is it WIN or LOSE
- LOSE is a constant to decide when the game is lost
- WIN is a constant to decide when the game is win
- GAME_PATH is a list of checkpoints for the balloons to go through.

```
# --- Initialize Pygame ---
pygame.init()
pygame.mixer.init()
pygame.display.set_caption(CAPTION)
screen = pygame.display.set_mode((WIDTH, HEIGHT))
clock = pygame.time.Clock()
```

```
font = pygame.font.Font("assets/font/Pixeltype.ttf", FONT_SIZE)
```

- pygame.init() is Initialize all Pygame modules
- pygame.mixer.init() is Initialize the Pygame mixer for sound
- pygame.display.set_caption(CAPTION) is Set the window title bar text to the value of CAPTION
- Screen is create the game window with a size of WIDTH x HEIGHT pixels.
- Clock is create a Clock object to manage frame rate
- Font is loading a custom font file (Pixeltype.ttf) and FONT_SIZE determines the size of the text to be rendered.

```
# --- Sounds ---
pop_sound_effect = pygame.mixer.Sound("assets/sound/popSound.mp3")
pop_sound_effect.set_volume(POP_SOUND_VOLUME)

# --- Load Map Image ---
map_image = pygame.image.load("assets/Images/Map/map2.png").convert()
mainmenu_image = pygame.image.load("assets/Images/Map/map1.png").convert()

# --- Load Monkey Assets ---
monkey_image = pygame.image.load("assets/Images/Monkey/Monkey.png").convert_alpha()
monkey_sheet = pygame.image.load("assets/Images/Monkey/animationMonkey-sheet.png").convert_alpha()
upgrade_monkey_image = pygame.image.load("assets/Images/GameMenu/upgradeMonkey.png").convert_alpha()

# --- Load Menu Button Assets ---
monkeymenu_button = pygame.image.load("assets/Images/GameMenu/buyMonkey.png").convert()
cancelmenu_button = pygame.image.load("assets/Images/GameMenu/cancelMenu.png").convert_alpha()
upgrade_monkey_button = pygame.image.load("assets/Images/GameMenu/upgradeMonkey.png").convert_alpha()
sell_button = pygame.image.load("assets/Images/GameMenu/sell.png").convert_alpha()
start_button = pygame.image.load("assets/Images/GameMenu/beginButton.png").convert_alpha()
play_button = pygame.image.load("assets/Images/GameMenu/playButton.png").convert_alpha()
fast_forward_button = pygame.image.load("assets/Images/GameMenu/FastForward.png").convert_alpha()
restart_button = pygame.image.load("assets/Images/GameMenu/restartButton.png").convert_alpha()
health_image = pygame.image.load("assets/Images/GameMenu/Health.png").convert_alpha()
coin_image = pygame.image.load("assets/Images/GameMenu/Coin.png").convert_alpha()
music_on = pygame.image.load("assets/Images/GameMenu/musicOn.png").convert_alpha()
music_off = pygame.image.load("assets/Images/GameMenu/musicOff.png").convert_alpha()

# --- Load Balloon Assets ---
balloons_images = {
```

```
    "weak": pygame.image.load("assets/Images/Balloons/Balloons_Red.png").convert_alpha(),
    "medium": pygame.image.load("assets/Images/Balloons/Balloons_Blue.png").convert_alpha(),
    "strong": pygame.image.load("assets/Images/Balloons/Balloons_Green.png").convert_alpha(),
    "very strong": pygame.image.load("assets/Images/Balloons/Balloons_Yellow.png").convert_alpha()
}
```

Load the image and store it in a variable,

- .convert() is to convert the image to the same pixel format as the display surface.
- .convert_alpha() is Similar to convert() but also preserves transparency by enabling per-pixel alpha values.
- .setvolume() is setting the volume of the pop sound effect to 50%.
- balloons_images is a dictionary that stores balloon images. where the keys are the colour of balloons and the name is switch for understandability as well as the value of each key is the balloon image

```
class TowerDefenseGame:
    def __init__(self):
        # --- Game state variables ---
        self.running = True
        self.game_over = False
        self.wave_started = False
        self.put_monkey = False
        self.selected_monkey = None
        self.invalid_time = None
        self.music_state = None

        # --- Time and game progress management ---
        self.last_balloon_spawn = pygame.time.get_ticks()

        # --- Game-related constants ---
        self.buy_monkey = BUY_MONKEY_COST
        self.upgrade_monkey = UPGRADE_MONKEY_COST
        self.sell_monkey = SELL_MONKEY_BASE_COST
        self.spawn_cooldown = SPAWN_COOLDOWN
        self.message_duration = MESSAGE_DURATION
        self.total_wave = TOTAL_WAVE
        self.game_outcome = GAME_OUTCOME
        self.lose = LOSE
        self.win = WIN
        self.music_volume = MUSIC_VOLUME

        # --- Map setup ---
        self.map_instance = Map(map_image)
        self.map_instance.process_balloons()

        # --- Sprite groups ---
        self.balloons_group = pygame.sprite.Group()
```

```
        self.monkey_group = pygame.sprite.Group()

        # --- Menu and UI setup ---
        # Monkey-related menus and buttons
        self.monkeymenu = MonkeyMenu(monkeymenu_button, 1000, 10, True)
        self.cancelmenu = MonkeyMenu(cancelmenu_button, 1150, 60, True)
        self.upgrademonkey = MonkeyMenu(upgrade_monkey_button, 1010, 240, True)
        self.sellbutton = MonkeyMenu(sell_button, 1030, 160, True)

        # Game control buttons:
        self.restartbutton = MonkeyMenu(restart_button, 500, 330, True)
        self.startbutton = MonkeyMenu(start_button, 1150, 620, True)
        self.playbutton = MonkeyMenu(play_button, 540, 540, True)
        self.fastforwardbutton = MonkeyMenu(fast_forward_button, 1150, 620, False)

        # Music control buttons:
        self.musicon = MonkeyMenu(music_on,930,10,True)
        self.musicoff = MonkeyMenu(music_off,930,10,True)
```

Initializing all the attributes needed for the game

- self.running : attribute to check if the game is running
- self.game_over : attribute to check the game condition
- self.wave_started : attribute to check if the wave has started
- self.put_monkey : attribute to check if the player is placing a monkey
- self.selected_monkey : attribute to store value if the player select a monkey
- self.invalid_state : attribute to store value if there is an invalid action
- self.music_state : attribute to track if the music is on / off
- self.last_balloon_spawn : attribute to control spawn cooldown
- All game-related constants is fill with the constants
- self.map_instance is to create an instance of the game map using the Map class.
- self.map_instance.process_balloons() is to pre-processes the balloon paths on the map.
- self.balloons_group = pygame.sprite.Group() is a group to manage all balloon sprites.

- self.monkey_group = pygame.sprite.Group() is a group to manage all monkey tower sprites.
- All the menu and UI setup attribute is to make a button using the MonkeyMenu class

```
def handle_events(self):
        """Handle user input and quit events."""
        for event in pygame.event.get():
                # Stop the game loop when closing the game
                if event.type == pygame.QUIT:
                    self.running = False

                # If mouse button is pressed and the mouse button
                # that is getting pressed is Left mouse button
                if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
                    mouse_position = pygame.mouse.get_pos() # Getting mouse cursor Position
                    if mouse_position[0] < 950 and mouse_position[1] > 75:
                        self.clear_select()
                        if self.put_monkey == True:

                            # Able to put monkey if there is enough money
                            if self.map_instance.money >= self.buy_monkey:
                                if Map.valid_position(mouse_position[0],mouse_position[1]) == True:
                                    monkey = Monkey(monkey_sheet,mouse_position,pop_sound_effect)
                                    self.monkey_group.add(monkey)
                                    self.map_instance.money -= self.buy_monkey
                                else:
                                    self.invalid_time = pygame.time.get_ticks()
                                    self.draw_text("Invalid Placement","red","black",500,75)
                        else:
                            self.selected_monkey = self.select_monkey(mouse_position)
```

The handle_events function is a function handle user input :

```
        for event in pygame.event.get():
```

- is a for loop that make sure the program can always track and handle the user input

```
                if event.type == pygame.QUIT:
```

- is a if statement to handle input from user if they want to quit the program

```
- if event.type == pygame.MOUSEBUTTONDOWN and
  event.button == 1:
-
```

- Is a condition that return True a mouse button is pressed down and the button pressed was the left mouse button
- The mouse_position variable stores the mouse position in a tuple with its horizontal and vertical coordinate

```
if mouse_position[0] < 950 and mouse_position[1] > 75:
```

- Return true, if the mouse position for x coordinate is less than 950 and the y coordinate is more than 75
- self.clear_select() will clear the selection if the mouse position is inside the range, making sure that only one monkey can be selected at a time

```
if self.put_monkey == True:

                        # Able to put monkey if there is enough money
                        if self.map_instance.money >= self.buy_monkey:
                            if Map.valid_position(mouse_position[0],mouse_position[1]) == True:
                                monkey = Monkey(monkey_sheet,mouse_position,pop_sound_effect)
                                self.monkey_group.add(monkey)
                                self.map_instance.money -= self.buy_monkey
                            else:
                                self.invalid_time = pygame.time.get_ticks()
                                self.draw_text("Invalid Placement","red","black",500,75)
```

- Checking the condition three times, it will check if the monkey is placed or not, then it will check if the player has enough money than the cost to buy the monkey, then it will check if the placement of the monkey is valid or not .
- If True , the monkey variable will be a new instance from the Monkey class. It will also add that monkey to the monkey group as well as decreasing the player's total money.
- If False , pygame.time.get_ticks() will track the time since the game started and store it to the variable self.invalid_time. Then a text will appear using the draw_text function

```
else:
```

```
                                self.selected_monkey =
self.select_monkey(mouse_position)
```

- If monkey is not placed, the selected_monkey attribute will contain the value of the function select_monkey(mouse_position)

```
    def draw_text(self,text, text_color, outline_color, x, y):
        """Draws text with an outline effect."""

        # The width of the outline
        outline_width = 2

        # Draw the outline
        for i in range(-outline_width, outline_width + 1):
            for j in range(-outline_width, outline_width + 1):

                # Making sure the original text is not affected
                if i != 0 or j != 0:
                    # Displaying the text around the original text
                    outline_text = font.render(text, True, outline_color)
                    screen.blit(outline_text, (x + i, y + j))

        # Displaying the text
        txt = font.render(text, True, text_color)
        screen.blit(txt, (x, y))
```

The draw text function is a function to render a font with an outline color. How it works is we set how thick the outline we want it to be, then we render the outline using a nested for loop over the range of -outline_width to outline_width which in this example is in the range of -2, -1, 0, 1, 2 . For each iteration, it will display the text based on the outline color at the x add with i and y add with j location . I also make sure the original text is not overwritten by using the if statement . After the loop to display the font in the outline_color, the function will display the text in the text color we want at the x and y location .

```
    def select_monkey(self, mouse_position):
        """Selects a monkey at the given mouse position."""

        for monkey in self.monkey_group:

            # If the mouse position is inside the monkey rectangle, it returns true
            if monkey.rect.collidepoint(mouse_position):
```

```
                monkey.selected = True
                return monkey

        return None
```

The select_monkey function accepts one argument which is the mouse_position. The function will get the monkey from the monkey_group and check if the mouse position is inside that monkey rectangle.

If it is True, the attribute monkey.selected will be True and will show the range of the selected monkey and the function will return that monkey

If it is False, the function will return None.

```
    def clear_select(self):
        """Clears selection from all monkeys."""

        for monkey in self.monkey_group:
            monkey.selected = False
```

The clear_select function will get a monkey from the monkey_group and change the attribute monkey_selected to False.

```
    def name(self):
        """Determines the wave number for display."""

        # Showing current wave number
        if self.map_instance.level <= 19:
            return self.map_instance.level + 1
        else:
            return 20
```

The name function will check if the map level is below and equal to 19

- If it is True, it will return the map level which get increase by 1

- If it is False, it will return 20

```
    def handle_main_game(self):
        """Runs core game state logic before rendering and event handling."""

        # Check for game-over conditions
        if self.game_over == False:

            # If the health is 0 , game over
            if self.map_instance.health <= 0:
                self.game_over = True
                self.game_outcome = LOSE

            # If complete all the wave, win
            if self.map_instance.level + 1 > self.total_wave:
                self.game_over = True
                self.game_outcome = WIN

            # Update groups
            self.balloons_group.update(self.map_instance)
            self.monkey_group.update(self.balloons_group, self.map_instance)

            # Display map and balloons
            self.map_instance.draw(screen)
            self.balloons_group.draw(screen)
```

handle_main_game is the main function, this method will check if the game_over attribute is False . If it is False,

- It will check if the health is lower or equal to 0. If True, the game_over attribute will change to True and game_outcome will change to lose.
- It will also check if the map level is more than the number of total waves. If True, the game_over attribute will change to True and game_outcome will change to win.

After checking , the balloons group and the monkey group will have the update method with the argument of map attribute. After updating the map, the balloon will be displayed on the screen.

```
# Handle balloon spawning
            if self.wave_started != True:
                # Show start button if wave is not start yet
                if self.startbutton.draw(screen):
                    self.wave_started = True

            else:
                # Normal game speed and if the fastforward button is hold the game will fast forward
                self.map_instance.game_speed = 1
                if self.fastforwardbutton.draw(screen):
```

```
                    self.map_instance.game_speed = 2

                # Cooldown before next balloon come out
                if pygame.time.get_ticks() - self.last_balloon_spawn > (self.spawn_cooldown /
self.map_instance.game_speed):

                    # Displaying all the balloons inside the list
                    if self.map_instance.spawned_balloons < len(self.map_instance.balloon_list):

                        # Get the balloon with the balloon stats from the map balloon list
                        balloon_type =
self.map_instance.balloon_list[self.map_instance.spawned_balloons]
                        balloons = Balloons(balloon_type, balloons_images, GAME_PATH)
                        self.balloons_group.add(balloons)
                        self.map_instance.spawned_balloons += 1

                        # Update to a new time, to restart the cooldown
                        self.last_balloon_spawn = pygame.time.get_ticks()
```

The function then will check if the wave has started or not, if it hasn't started there will be a begin button that is ready to be clicked. When it is clicked the wave will start, which is below the else statement. The begin button will change to a fast forward button and the game will start with a normal speed. If the fast forward button is being held, the game will go at a faster speed.

```
if pygame.time.get_ticks() - self.last_balloon_spawn > (self.spawn_cooldown /
self.map_instance.game_speed):
```

This if statement ensures that the balloons did not all appear at once but appear one at a time. How it works is pygame,time.get_ticks() will track the time since the game started and its time will increase over time , then it will subtract with self.last_balloon_spawn which has a value of pygame.time.get_ticks(), however it is store in a variable that is why the time will not change. Because the pygame,time.get_ticks() keeps increasing the expression between pygame.time.get_ticks() - self.last_balloon_spawn will get bigger which will make the if statement true overtime. the code shows an illusion of a cooldown, and I decide on how long the cooldown the balloon to be by using self.spawn_cooldown attribute which in this code I input as 0.5 s / 500 ms.

```
# Displaying all the balloons inside the list
                    if self.map_instance.spawned_balloons < len(self.map_instance.balloon_list):

                        # Get the balloon with the balloon stats from the map balloon list
                        balloon_type =
self.map_instance.balloon_list[self.map_instance.spawned_balloons]
                        balloons = Balloons(balloon_type, balloons_images, GAME_PATH)
                        self.balloons_group.add(balloons)
                        self.map_instance.spawned_balloons += 1

                        # Update to a new time, to restart the cooldown
                        self.last_balloon_spawn = pygame.time.get_ticks()
```

After checking the cooldown, there will be another if statement that checks if there are still balloons to spawn in that wave. It will use indexing from the balloon list to get a balloon and store it in balloon_type. After getting the balloon_type, the function will create a new instance using the Balloons class with the argument of that balloon_type. It will then add that balloon to the group and add self.map_instance.spawned_balloons by 1 and update self.the last_balloon_spawn to reset the cooldown. The purpose of self.map_instance.spawned_balloons attribute get increase by 1 is to make sure all balloon from the balloon list is being used

```
            # Handle wave completion
             if self.map_instance.check_wave_complete() == True:

                # Rewarding player with money
                self.map_instance.money += 100
                self.map_instance.level += 1

                # Making sure player can prepare before the next wave
                self.wave_started = False

                # Reset time tracking, wave and processing balloons
                self.last_balloon_spawn = pygame.time.get_ticks()
                self.map_instance.reset_wave()
                self.map_instance.process_balloons()
```

The function will handle the wave completion by using the check_wave_complete() method in the Map class. If the method return True :

1. The block of code will add the player money by 100, add the wave number by 1,

2. change the self.wave_started to ensure the player can prepare for the next wave, without this the player will have to have an auto skip wave feature.
3. The block of code will reset the cooldown, the wave, as well as reprocess the balloons

```
        # If the sound image is pressed the music will mute
        if self.music_state == None:
            if self.musicon.draw(screen) == True:
                # Mute music
                self.music_volume = 0
                pygame.mixer.music.set_volume(self.music_volume)
                pop_sound_effect.set_volume(self.music_volume)

                # Remove the music on image
                self.musicon.remove_menu()

                # Displaying the music off image
                self.musicoff.draw(screen)
                self.music_state = False

        # If the sound image is pressed the music will unmute
        elif self.music_state == False:
            if self.musicoff.draw(screen) == True:
                # Unmute music
                self.music_volume = 0.1
                pygame.mixer.music.set_volume(self.music_volume)
                pop_sound_effect.set_volume(self.music_volume * 5)

                # Remove the music off image
                self.musicoff.remove_menu()

                # Displaying the music on image
                self.musicon.draw(screen)
                self.music_state = None
```

Both of the first if and elif statements are used to change the music icon and ensure the music turns on and off. How it works is if the music_state is none the icon is music on , if it is press:

1. Change the music volume to 0 and update music and sound effect volume by using pygame.mixer.music.set_volume() and pop_sound_effect.set_volume()
2. Remove music on icon, add music off icon , and change music_state value to False so the elif statement can work.

The elif statement work the same way, if it is press:

1. Change the music volume to 0.1 to make sure it is not too loud and update music and sound effect volume by using pygame.mixer.music.set_volume() and pop_sound_effect.set_volume() * 5 to make sure the sound effect is louder
2. Remove music off icon, add music on icon , and change music_state value to None so the if statement can work.

```
# Displaying each monkey from the monkey group to the screen
            for monkey in self.monkey_group:
                monkey.draw(screen)
```

The for loop will display each monkey from the monkey class to the screen

```
            # When the menu is clicked, player can place the monkey
            if self.monkeymenu.draw(screen) == True:
                self.put_monkey = True
```

The if statement returns True if the monkey menu is pressed or clicked, it will change self.put_monkey value to True.

```
# When the player can place the monkey, they can sell the monkey as well
            if self.put_monkey == True:
                if self.cancelmenu.draw(screen) == True:
                    self.put_monkey = False

                # Make the position of cursor same with monkey image
                cursor_rect = monkey_image.get_rect()
                cursor_position = pygame.mouse.get_pos()
                cursor_rect.midbottom = cursor_position

                # Displaying the monkey
                if cursor_position[0] <= 975:
                    screen.blit(monkey_image,cursor_rect)
```

If the statement is True, it will check :

- if the cancelmenu is being pressed. If True, it will change self.put_monkey value to False

Then it will make a rectangle and store it in cursor_rect and get the mouse position and store it in a variable cursor_position. The mid bottom of the rectangle will be the same position as the cursor position. The monkey will get display if the x axis is less than or equal to 975

```
# When monkey is selected, range is visible and monkey can be upgraded
            if self.selected_monkey != None:

                # If the level is not maxed the button, it will appear and show its cost
                if self.selected_monkey.upgrade_level < 4:
                    screen.blit(coin_image,(1140,230))
                    self.draw_text("$100","white","black",1190,240)

                    # If the upgrade button is pressed , upgrade the monkey by one level and reduce the
money
                    if self.upgrademonkey.draw(screen):
                        if self.map_instance.money >= self.upgrade_monkey:
                            self.selected_monkey.upgrade()
                            self.map_instance.money -= self.upgrade_monkey

                # Displaying coin image to the screen
                screen.blit(coin_image,(1140,150))

                # The money get increase the higher the level
                match self.selected_monkey.upgrade_level:
                        case 1:
                            self.sell_monkey = SELL_MONKEY_BASE_COST
                        case 2:
                            self.sell_monkey = SELL_MONKEY_LVL2_COST
                        case 3:
                            self.sell_monkey = SELL_MONKEY_LVL3_COST
                        case 4:
                            self.sell_monkey = SELL_MONKEY_LVL4_COST

                # Displaying the cost to the screen
                self.draw_text(f"${self.sell_monkey}","white","black",1190,160)

                # If the sell button is clicked, monkey get remove and player get money
                if self.sellbutton.draw(screen):
                    self.selected_monkey.sell_monkey()
                    self.selected_monkey = None
                    self.map_instance.money += self.sell_monkey
```

If the monkey is selected, it will check if the selected monkey upgrade level is less than 4, if it is True:

1. Display the coin image in (1140,230) location
2. Show the text using draw_text() method
3. It will check if upgrade monkey button is getting pressed, If True

- It will check if the player has enough money to upgrade the monkey which is what this code means.

```
if self.map_instance.money >= self.upgrade_monkey:
```

- If the player has enough money , the selected monkey will upgrade the monkey using the upgrade() method and subtract the player money.

```
# The money get increase the higher the level
            match self.selected_monkey.upgrade_level:
                    case 1:
                        self.sell_monkey = SELL_MONKEY_BASE_COST
                    case 2:
                        self.sell_monkey = SELL_MONKEY_LVL2_COST
                    case 3:
                        self.sell_monkey = SELL_MONKEY_LVL3_COST
                    case 4:
                        self.sell_monkey = SELL_MONKEY_LVL4_COST
```

After checking, it will display the coin image then change the self.sell_monkey amount based on the level of the monkey. The sell_monkey text is display using the draw_text method.

```
            # If the sell button is clicked, monkey get remove and player get money
            if self.sellbutton.draw(screen):
                self.selected_monkey.sell_monkey()
                self.selected_monkey = None
                self.map_instance.money += self.sell_monkey
```

If the sell button is pressed, the selected monkey will

- Get removed using the sell_monkey() method
- self.selected_monkey value will be None
- Increase the player money base on the self.sell_monkey value

```
        else:
            pygame.draw.rect(screen,"dodgerblue",(300,200,500,300),border_radius = 30)

            # Game over screen
            if self.game_outcome == self.lose:
                self.draw_text("GAME OVER","red","black",470,230)
                self.draw_text("Restart","white","black",490,440)
```

```
        # Win screen
        elif self.game_outcome == self.win:
            self.draw_text(" YOU WIN !","yellow","black",480,230)
            self.draw_text("Restart","white","black",490,440)

        # If player click the restart button, it will restart the game
        if self.restartbutton.draw(screen) == True:

            # Restarting Attributes
            self.game_over = False
            self.wave_started = False
            self.put_monkey = False
            self.selected_monkey = None
            self.last_balloon_spawn = pygame.time.get_ticks()
            self.map_instance = Map(map_image)
            self.map_instance.process_balloons()

            # Empty the group for balloon and monkey
            self.balloons_group.empty()
            self.monkey_group.empty()
```

If the self.game_over is not False these code block will run, starting with displaying a rectangle then the code will check the game_outcome is equals to self.lose or self.win, if it is self.lose:

- It will display the game over text as well as the restart text both using the draw_text() method

  If it is self.win

- It will display the you win text as well as the restart text both using the draw_text() method

After checking the game_outcome, the code will check if the restart button is getting pressed or not, if it is pressed, It will restart all the attributes including :

```
# Restarting Attributes
                self.game_over = False
                self.wave_started = False
                self.put_monkey = False
                self.selected_monkey = None
                self.last_balloon_spawn = pygame.time.get_ticks()
                self.map_instance = Map(map_image)
```

```
                    self.map_instance.process_balloons()
```

And after restarting it will also empty the balloon and monkey group.

```
    def handle_text_delay(self):
        """ Handling text delay """

        # Checking if the time has value or not
        if self.invalid_time != None:

            # Track the time since game started
            current_time = pygame.time.get_ticks()

            # Give delay before the text stop displaying
            if current_time - self.invalid_time < self.message_duration:
                self.draw_text("Invalid Placement","red","black",450,75)
            else:
                self.invalid_time = None
```

The handle_text_delay method is the let text be display for a certain amount of time how it works is , it will first check if self.invalid_time is not equal to None, if True :

- It will track the time since the game start using pygame.time.get_ticks() and store the value in current_time
- It will then show the “invalid placement” texts for message_duration using the draw_text() method. If current_time - self.invalid_time is lesser than self.message_duration
- If current_time - self.invalid_time is bigger than self.message_duration, it will change the value self.invalid_time to None

```
    def render_ui(self):
        # Displaying health and coin images
        screen.blit(health_image,(30,15))
        screen.blit(coin_image,(150,10))

        # Displaying the value of health and coin as well as the number of wave
        self.draw_text(str(self.map_instance.health),"white","black",70,20)
        self.draw_text("$"+ str(self.map_instance.money),"white","black",200,20)
```

```
        self.draw_text("Wave " + str(self.name()) + " / 20","yellow","black",500,20)

        # # A line showing the game path
        # pygame.draw.lines(screen,"grey0", False, GAME_PATH)
```

The render_ui method does not take any arguments, it will display the health and coin image in (30,15) and (150,10) locations then it will display those 3 texts using the draw_text() method. The commented code is a code to display lines of the balloon path, it can be uncommented if it is needed

```
def main_menu(self):
        ''' Main menu screen '''

        # Loop the menu
        while self.running:
            # A loop to make sure game is always ready for user input
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False

            # Image of main menu screen
            screen.blit(mainmenu_image,(0,0))

            # If the button is pressed the it will go to the game screen
            if self.playbutton.draw(screen):
                self.main_loop()

            # Update screen to the latest change
            pygame.display.update()
```

The main_menu method is the main menu screen, starting with looping the game to make sure the display is always being shown using a while loop, then using a for loop to get input from player and make sure the player can quit with “ if event.type == pygame.QUIT “. After that, displaying the main menu image to the screen using blit() method from pygame and if the self.playbutton is pressed the self.main_loop() will run which is the method for looping the whole game

pygame.display.update() ensure all updates are reflected to the screen.

```
    def main_loop(self):
        """Main game loop."""

        # Game Music and set the music to loop
        pygame.mixer.music.load("assets/sound/gameMusic.mp3")
        pygame.mixer.music.play(-1)

        # Set music volume
        pygame.mixer.music.set_volume(self.music_volume)

        # Loop the game
        while self.running:
            self.handle_main_game()
            self.handle_events()
            self.handle_text_delay()
            self.render_ui()

            # Update screen to the latest change
            pygame.display.update()

            # limit the screen to have 60 FPS
            clock.tick(FPS)

        # Quit the program
        pygame.quit()
```

The main_loop method will start with loading the music asset and play the music in an infinite loop because of pygame.mixer.music.play(-1), then the volume will change depending on the value of self.music_volume. After that the method has a while loop all the previous method which is self.handle_main_game(), self.handle_events, self.handle_text_delay, and self.render_ui(). Then pygame.display.update() ensures all updates are reflected to the screen. clock.tick(FPS) make sure the game is at 60 frames per second. pygame.quit() make sure the program is close.

```
# Checking if the script is run directly or imported as a
module
if __name__ == "__main__":
    game = TowerDefenseGame()
    game.main_menu()
```

Checking if the script is run directly or imported as a module, if it is run directly we create a new instance call game and run the main_menu() method from the TowerDefenseGame() class.

- *balloons.py*

```
import pygame
from balloonsData import balloons_data
```

Importing pygame module and balloons data to the balloons.py

```
class Balloons(pygame.sprite.Sprite):
    def __init__(self, balloon_type, images, checkpoints):
        '''
        Represents balloon in the game. Balloons move along a defined
path and
        interact with the game map by either reaching the end or
being destroyed by the player.
        '''
        super().__init__()  # Initialize the sprite using pygame
Sprite class.

        # Set the balloon image
        self.image = images.get(balloon_type)

        # Store the path checkpoints and next checkpoint
        self.checkpoints = checkpoints
        self.next_checkpoint = 1

        # Set the balloon health and speed from balloons_data
        self.health = balloons_data.get(balloon_type)["health"]
        self.speed = balloons_data.get(balloon_type)["speed"]

        # Set the initial position of the balloon and create a
rectangle
        self.position = pygame.Vector2(self.checkpoints[0])
        self.rect = self.image.get_rect(center=self.position)
```

Making a new class Balloon, inherits the sprite class from the pygame module and initializes all the necessary attributes which are in the code above and the explanation is in the code.

```
    def update(self, map):
```

```
            '''
            Updates the balloon's state during each game frame:
            - Moves the balloon along its path.
            - Checks if the balloon is still alive (has health
left).

            '''
            self.move(map)
            self.check_alive(map)
```

The update method updates the balloon state during each game frame by using the move and check_alive method with map as the argument.

```
def check_alive(self, map):
        '''
        Checks if the balloon's health is zero.
        If so, it rewards the player and remove the balloon from the
game.

        '''
        if self.health <= 0:
            # Remove balloon, reward the player with money
            # and increment the killed balloons count
            map.money += 3
            map.killed_balloons += 1
            self.kill()
```

The check_alive method checks if the health attribute is less than zero. If the health is less or equal to 0, the block of code will run which are increase the player money by 3, increase the number of killed balloons by 1 and remove the balloon by using the kill() method from the pygame sprite class.

```
def move(self, map):
       '''
       Moves the balloon along the path defined by checkpoints. If it reaches the end,
       it decreases the player's health and increments the missed balloon count.

       '''
       # Check if the balloon has more checkpoints to reach
       if self.next_checkpoint < len(self.checkpoints):
```

```
            # Set the target to the next checkpoint and
            # calculate the distance from current position to target
            self.target = pygame.Vector2(self.checkpoints[self.next_checkpoint])
            self.movement = self.target - self.position
        else:
            # If the balloon reaches the last checkpoint,
            # remove balloon, decrease player health and increase missed balloons count
            self.kill()
            map.health -= 1
            map.missed_balloons += 1


        distance = self.movement.length()

        # Move the balloon towards the next checkpoint
        if distance >= (self.speed * map.game_speed):
            self.position += self.movement.normalize() * self.speed * map.game_speed
        else:
            # If the balloon is close enough to the checkpoint,
            # move directly to it and set the next checkpoint
            if distance != 0:
                self.position += self.movement.normalize() * distance
            self.next_checkpoint += 1

        self.rect.center = self.position
```

The move method move the balloon from one checkpoint to the other checkpoint which is in the GAME_PATH constants in the main module and if the balloon reaches the end of the path it will remove the balloon using the kill() method. Because GAME_PATH is in a list, the checkpoint will change when the balloon is in the checkpoint location using self.movement.length() to track the balloon distance.

- *balloonsData.py*

balloonsData.py has two variables which are balloons_data and balloons_spawn_data.

```
# Data for each type of balloon
balloons_data = {
    "weak":{
        "health":5,
        "speed":2
    },
    "medium":{
        "health":10,
        "speed":3
    },
    "strong":{
        "health":20,
        "speed":4
    },
    "very strong":{
        "health":30,
        "speed":6
    },
}
```

Balloons_data is a variable that stores the balloon stats by using a dictionary with the balloon's type as the key and their health and speed in a dictionary as their values.

```
        # Number of Balloons that spawn each wave
        balloons_spawn_data = [
            {# Wave 1
                "weak":20,
                "medium":0,
                "strong":0,
                "very strong":0
            },
            {# Wave 2
                "weak":35,
                "medium":0,
                "strong":0,
                "very strong":0
            },
            {# Wave 3
                "weak":25,
                "medium":5,
                "strong":0,
                "very strong":0
            },
            {# Wave 4
                "weak":35,
                "medium":15,
                "strong":0,
                "very strong":0
            },
            {# Wave 5
                "weak":5,
                "medium":25,
                "strong":0,
                "very strong":0
            },
            {# Wave 6
                "weak":15,
                "medium":15,
                "strong":5,
```

```
            "very strong":0
        },
        {# Wave 7
            "weak":20,
            "medium":20,
            "strong":10,
            "very strong":0
        },
        {# Wave 8
            "weak":10,
            "medium":20,
            "strong":15,
            "very strong":0
        },
        {# Wave 9
            "weak":0,
            "medium":0,
            "strong":30,
            "very strong":0
        },
        {# Wave 10
            "weak":0,
            "medium":100,
            "strong":0,
            "very strong":0
        },
        {# Wave 11
            "weak":10,
            "medium":10,
            "strong":15,
            "very strong":5
        },
        {# Wave 12
            "weak":0,
            "medium":15,
            "strong":10,
            "very strong":10
        },
        {# Wave 13
            "weak":0,
            "medium":50,
            "strong":25,
            "very strong":0
        },
        {# Wave 14
            "weak":50,
            "medium":15,
            "strong":10,
            "very strong":10
        },
        {# Wave 15
            "weak":20,
            "medium":15,
            "strong":15,
            "very strong":10
        },
        {# Wave 16
            "weak":0,
            "medium":0,
            "strong":40,
            "very strong":10
        },
        {# Wave 17
            "weak":0,
            "medium":0,
            "strong":10,
            "very strong":15
        },
        {# Wave 18
            "weak":0,
            "medium":0,
            "strong":80,
            "very strong":0
        },
        {# Wave 19
            "weak":0,
            "medium":0,
            "strong":10,
            "very strong":30
        },
        {# Wave 20
            "weak":10,
            "medium":10,
            "strong":10,
```

```
            "very strong":50
        },
    ]
```

Balloons_spawn_data is a variable that stores the number of balloons that will spawn in the wave by using a list. Using a dictionary to specify the number of balloons that will spawn according to their type of balloons.

- *map.py*

```
import random
from balloonsData import balloons_spawn_data
```

Importing random module and balloons data to the balloons.py

```
# Constants for health and money
HEALTH = 100
MONEY = 650
```

Constants for the player starting money and health

```
# Defines the Map class
class Map:
    def __init__(self,map_image):
        # set the image, health, money, wave and speed of the game
        self.image = map_image
        self.health = HEALTH
        self.money = MONEY
        self.level = 0
        self.game_speed = 1

        # Attributes for list of balloons and counting the balloons
        self.balloon_list = []
        self.spawned_balloons = 0
        self.killed_balloons = 0
        self.missed_balloons = 0
```

Making a new class Map and initializes all the necessary attributes which include the player initial health and money attribute, the wave number attribute using self.level and the speed of the game using self.game_speed. Self.balloon_list is to all the balloons in the current wave , the last three attributes are to track the amount of balloons per wave.

```
    def draw(self,screen):
            '''Draw the map image to the screen.'''
            screen.blit(self.image,(0,0))
```

The draw method is used to draw the map image to the screen at 0, 0 location

```
def check_wave_complete(self):
```

```
        '''Check if the wave is complete '''
        if (self.killed_balloons + self.missed_balloons) ==
len(self.balloon_list):
            return True
```

check_wave_complete method checks if the wave is complete by adding the amount of killed_balloons and missed_balloons, if it is equal to the length of the balloon list the method will return True.

```
    def reset_wave(self):
            '''Reset wave data for next wave.'''
            self.balloon_list = []
            self.spawned_balloons = 0
            self.killed_balloons = 0
            self.missed_balloons = 0
```

reset_wave method reset the wave by emptying the balloon_list to prepare for the next wave and reassign the value of spawned_balloons, killed_balloons and missed_balloons.

```
 def process_balloons(self):
        '''Process balloon spawns for the current level using balloon
spawn data.'''
        if self.level < len(balloons_spawn_data):
            balloons = balloons_spawn_data[self.level]
            for balloon_type in balloons:
                spawn_balloons = balloons[balloon_type]
                for balloon in range(spawn_balloons):
                    self.balloon_list.append(balloon_type)
            # Randomize the balloons so it is not in order
            random.shuffle(self.balloon_list)
```

process_balloon method processes the balloon spawn for the current level by appending the balloon to self.balloon_list and using balloon_spawn_data from balloonsData.py for the amount of each type of balloon it needed to append.

```
    def valid_position(x,y):
        '''Check if a given (x, y) position is valid for placing a monkey.'''
        if 75 <= y <= 110:
            return True

        if x <= 280 and 75 <= y <= 260:
            return True
```

```
if 360 <= x <= 450 and 180 <= y <= 260:
    return True

if 530 <= x <= 950 and y <= 250:
    return True

if x <= 270 and 330 <= y <= 390:
    return True

if 370 <= x <= 600 and 330 <= y <= 390:
    return True

if 530 <= x <= 600 and y <= 390:
    return True

if 690 <= x <= 710 and 310 <= y <= 500:
    return True

if 790 <= x <= 950 and 75 <= y <= 720:
    return True

if x <= 110 and y >= 330:
    return True

if 190 <= x <= 270 and 460 <= y <= 540:
    return True

if 360 <= x <= 710 and 460 <= y <= 510:
    return True

if 360 <= x <= 410 and y >= 460:
    return True

if x <= 410 and y >= 610:
    return True

if 510 <= x <= 950 and y >= 590:
    return True

return False # else return false
```

Valid_position method ensures that players cannot place monkeys in the path.

- *monkey.py*

```
import pygame
import math
from monkeyData import monkey_data
```

Importing pygame, math and monkey_data module to monkey.py.

```
# Constants for animation speed and monkey damage
animation_delay = 110
monkey_damage = 5
```

Animation_delay is a variable to change the frame per second (FPS) of the monkey animation. Monkey_damage variable is the damage the monkey will deal to balloons.

```
# Defines the Monkey class and inherit the pygame.sprite.Sprite
class Monkey(pygame.sprite.Sprite):
    def __init__(self,sprite_sheet,position,sound):
        '''Initialize the monkey sprite '''
        super().__init__() # Initialize the sprite using pygame Sprite class.

        # set the range and cooldown for monkey depends on the upgrade level
        self.upgrade_level = 1
        self.range = monkey_data[self.upgrade_level - 1].get("range")
        self.cooldown = monkey_data[self.upgrade_level - 1].get("cooldown")

        # tracking the time for the last attack and time for frame update
        self.last_attack = pygame.time.get_ticks()
        self.update_time = pygame.time.get_ticks()

        # set attributes for set of monkey images for animation ,
        # monkey position and sound effect
        self.sprite_sheet = sprite_sheet
        self.position = position
        self.sound = sound

        # attribute for list of monkey frames for animation and
        # attribute for image from the list of frames and
        # track the frames using a frame_index attribute
        self.animation_list = self.load_images()
        self.frame_index = 0
        self.image = self.animation_list[self.frame_index]

        # attribute for checking if monkey is selected and
        # attribute for current target and a rectangle for the monkey
        self.selected = False
        self.target = None
        self.rect = self.image.get_rect(midbottom = position)
```

```
        # Attribute to show surface of the monkey attack range
        self.range_image = pygame.Surface((self.range * 2,self.range * 2))
        self.range_image.fill((0,0,0))
        self.range_image.set_colorkey((0,0,0))
        self.range_image.set_alpha(60)
        self.range_rect = self.range_image.get_rect()
        self.range_rect.center = self.rect.center
        pygame.draw.circle(self.range_image,"grey100",(self.range,self.range),self.range)
```

Making a new class Monkey, inherits the sprite class from the pygame module and initializes all the necessary attributes which are in the code above and the explanation is in the code.

```
def load_images(self):
        '''Load the monkey's animation frames from the sprite sheet.'''
        size  = self.sprite_sheet.get_height()
        animation_list = []
        # looping 8 times because the frames is 8 frames
        for i in range(8):
            temp_image = self.sprite_sheet.subsurface(i * size,0,size,size)
            animation_list.append(temp_image)
        return animation_list
```

Load_images method loads the monkey animation frames from the sprite sheet by getting the height of the sprite sheets and making a new list called animation_list. The code loop for 8 times using range() function because the animation consists of 8 frames. Inside the loop, self.sprite_sheet.subsurface get each of the frames from the sprite sheet and appends to the list , After the loop is done, the function will return animation_list.

```
def pick_target(self,balloons_group):
        '''Pick a balloon within the monkey range and deal damage to it.'''
        x_distance = 0
        y_distance = 0
        for balloons in balloons_group:
            # if the balloon is still alive
            if balloons.health > 0:
                # Calculating the distance from the monkey to the balloon
                x_distance = balloons.position[0] - self.position[0]
                y_distance = balloons.position[1] - self.position[1]
                distance = math.sqrt(x_distance ** 2 + y_distance ** 2 )
                if distance < self.range:
                    self.target = balloons
```

```
                    self.target.health -= monkey_damage
                    self.sound.play()
                    break # make sure monkey can only target one balloon at a
time
```

The pick_target method will use a for loop to get the balloon from the balloon group then check if the balloon health is more than 0. If it is True, it will :

- Calculate the x distance by subtracting the balloon x position with the monkey x position
- Calculate the y distance by subtracting the balloon y position with the monkey y position
- Using math module and pythagoras theorem to count the hypotenuse and store it in a distance variable
- If the range of the monkey is more than the distance variable, it will set the target to that one balloon from the balloon group, decrease the balloon health by the monkey damage, and play the pop sound effect

```
def play_animation(self):
        '''Play the monkey attack animation and reset the animation
after.'''
        self.image = self.animation_list[self.frame_index]
        # change frames with delay
        if pygame.time.get_ticks() - self.update_time > animation_delay:
            self.update_time = pygame.time.get_ticks()
            self.frame_index += 1
            if self.frame_index >= len(self.animation_list):
                # reset animation, target, and timer for last attack
                self.frame_index = 0
                self.last_attack = pygame.time.get_ticks()
                self.target = None
```

play_animation method will play the monkey attack animation by using the animation_list from load_images() method. The method will start by choosing one of the frames and store it in self.image. The method will change the frames after a certain amount of time using the if statement and increase self.frame_index by 1 to change the image that is stored in self.animation_list list. To make sure index is not out of range, I use another if statement, if the value is more than the length of self.animation_list it will reset self.frame_index to 0, reset target, and reset timer for last attack.

```
def upgrade(self):
        ''' Upgrade the monkey stats '''

        # upgrading the level and change the range and
        # cooldown according to the monkeydata
        self.upgrade_level += 1
        self.range = monkey_data[self.upgrade_level - 1].get("range")
        self.cooldown = monkey_data[self.upgrade_level - 1].get("cooldown")

        # setting a new range according to the upgraded range
        self.range_image = pygame.Surface((self.range * 2,self.range * 2))
        self.range_image.fill((0,0,0))
        self.range_image.set_colorkey((0,0,0))
        self.range_image.set_alpha(60)
        self.range_rect = self.range_image.get_rect()
        self.range_rect.center = self.rect.center
        pygame.draw.circle(self.range_image,"grey100",(self.range,self.range),self.range)
```

upgrade method will upgrade the monkey level by 1. It will change the range and cooldown based on the level of the monkey and the amount of range and cooldown is in monkey_data. It will also create the same exact range with the initial but with different size.

```
    def sell_monkey(self):
        # remove the monkey
        self.kill()
```

Removing the monkey

```
def update(self,balloons_group, map):
        ''' Update the monkey state '''
        # if the target is picked, it will play the animation
        if self.target:
            self.play_animation()
        else:
            # if the cooldown has passed, it will pick a new target
            if pygame.time.get_ticks() - self.last_attack > (self.cooldown / map.game_speed):
```

```
                self.pick_target(balloons_group)
```

update method will update monkey animation, if the target is picked, play_animation() method will run, if the target is not picked, it will pick a balloon from the balloon group after a certain amount of time.

```
def draw(self,screen):
        '''Draw the monkey and range on the screen.'''
        screen.blit(self.image,self.rect)
        if self.selected:
            # show the range if the monkey is selected
            screen.blit(self.range_image,self.range_rect)
```

The draw method draws the monkey and the range to the screen, but the screen will only get drawn to the screen if the monkey is selected.

- *monkeyData.py*

```
monkey_data = [
    { # Level 1 Monkey
        "range":90,
        "cooldown":1500
    },
    { # Level 2 Monkey
        "range":110,
        "cooldown":1200
    },
    { # Level 3 Monkey
        "range":125,
        "cooldown":1000
    },
    { # Max Level Monkey
        "range":150,
        "cooldown":900
    }
]
```

monkeyData.py has one variable which stores a list with each of the elements containing a dictionary about the value of range and cooldown.

- *monkeyMenu.py*

```
import pygame
```

Import pygame module to monkeyMenu.py

```
# Defines the MonkeyMenu class
class MonkeyMenu(pygame.sprite.Sprite):
    def __init__(self,image,x,y,single_click):
        super().__init__()
        # Set the attributes
        self.image = image
        self.rect = self.image.get_rect(topleft = (x,y))
        self.click = False
        self.single_click = single_click
```

Making a new class MonkeyMenu, inherits the sprite class from the pygame module and initializes all the necessary attributes which include:

- Self.image is a attribute that store the image
- Self.rect makes a rectangle for the image
- Self.click and self.single click are the attribute to decide if the instance is click or hold

```
def draw(self,screen):
        position = pygame.mouse.get_pos()
        active = False
        # if mouse position is in the rectangle the condition will be True
        if self.rect.collidepoint(position):
            # if left click is press it will return True
            if pygame.mouse.get_pressed()[0] == 1 and self.click == False:
                # check if it is toggle or hold
                if self.single_click == True:
                    self.click = True
                active = True
        if pygame.mouse.get_pressed()[0] == 0:
            self.click = False
```

```
        screen.blit(self.image,self.rect)

        return active
```

Draw method gets the position of the mouse using pygame.mouse.get_pos().

```
if self.rect.collidepoint(position) == True:
```

If the mouse position is inside the rectangle this if statement will return True, and inside the if statement there is another if statement which will check if the left mouse button is getting pressed/ clicked. If it is True, it will check if it is single click or not, if it is self.click will change to True and change active variable to True. If the mouse button is not pressed / clicked , self.click becomes False.

Then the image will draw to the screen using blit() method and return the variable active

```
def remove_menu(self):
        ''' remove the menu '''
        self.kill()
```

remove_menu method remove the image using kill() method from pygame sprite class

6. Screenshots of the Project

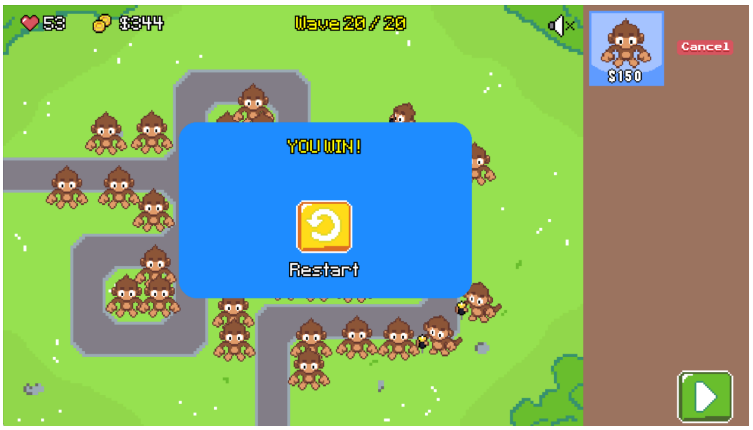
- *Main Menu Screen :*

- *Game Screen :*

- *Balloons Spawned and Monkey Placed :*

- *Game Over Screen :*

- *Victory Screen :*



- *Video Demo :* https://youtu.be/bjAo3_KKE0E

- *A3 - Poster :*

7. References

Coding With Russ. (2023, June 19). *Tower Defence Tutorial in Pygame*. YouTube. https://www.youtube.com/watch?v=Y82xjb8lW6M&list=PLjcN1EyupaQkm_kjeferSwzsOQNFpkyJp

Pixel Overload. (2023, June 20). *Bloons Tower Defence in Pixel Art*. YouTube. https://www.youtube.com/watch?v=fey2oM6hQik