Project Report: Turn-Based Game

Hao Chen

December 8, 2024

Northeastern University

CS 5001

Table of Content

Project Description and Explaination

This program is a turn-based game implemented using Python's pygame library.
Library listIt: pygame, random.
Player and enemy characters with health, attack, defense, and special abilities.
The background image and enemy image were generated by AI, and all the rest of the image material was original by me. Background music and audio material to group Internet free open source material.

The whole program is divided into four parts:
Classes
Initialization and Assets
Define Functions
Main Loop

Classes

This section organizes characters, games, and sound materials into their respective classes for easy subsequent calls. As for why the image material was not written as class, it was because the image material was first added to the program, I forgot to write class at that time, and at that time I was studying the animation effect (deleted later, image move too fast and can not slow down), there were a lot of code needed to capture the position of the image, the impact was too great, so I gave up the modification. Adding music material later was the realization that writing a class was a good way to do it

Charactor shows the character's name, hp, and attack power, as well as attack, defense, special powers, and AI logic

Set the role to participate in the Game, switch the round and judge the end of the game in the game, by writing these two classes, if I want to add multiple roles to let the player choose one to play, it will be very convenient. As for why I didn't do this, it's because now I draw the player's image myself, and that's what I want the character to look like, so I didn't add character selection

The last class is the one that manages all the sounds

Initialization and Assets

This is very similar to my original version of the code, mainly some of the most basic Settings of pygame, screen, fonts, colors, and very important variables that will be needed later.

Define Functions

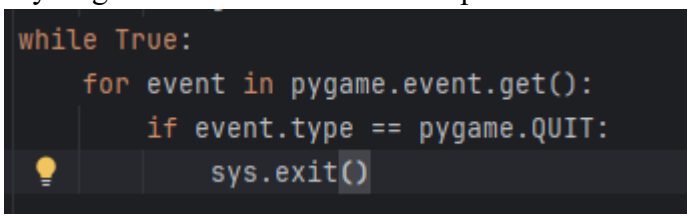
These are all functions needed in the main loop of the game, most of which are UI-

related, such as drawing a health Bar, drawing a button, drawing a start menu, etc. There are also some functional functions, such as initializing the game, special CD, etc

Main Loop

The main point I want to explain in this part is the logic of mouse click button. It can be said that buttons are not real buttons or fake buttons, they are just some images drawn on the screen, but their.rect properties are clear, so when capturing mouse action, as long as the coordinates of the mouse click area are the same as the.rect of the button image, You can think of it as clicking a button. All button functions are implemented according to this logic.

My original version of the main loop was:



Is to close the window by referring to sys, sys.exit(), later found not to use, set a variable running = True, when needed to end it equal to False.

The main loop's structure is: first judge the game state, menu, playing, or end. Then the next level is draw UI for each state. In playing state, it is divided into two parts by attribution of turn, the player Part and the enemy part, depending on whose turn it is, the player part performs different actions by collecting the position of the player's mouse click, while the enemy part performs actions by accepting random results generated by the AI (AI logic in the Class Part)

Finally refresh the screen

UI and graph

The graphical user interface could enhances the game's usability and aesthetics. Health bars and numeric indicators provide clear visual feedback on the status of the player and enemy. Buttons for actions are intuitive, changing color on hover and disabling when unavailable. Temporary action flags appear near characters to indicate their actions, while the action log provides a detailed record of recent moves, distinguishing player and enemy actions with color coding. When the game ends, a dedicated game-over screen displays the winner and offers options to replay or quit, seamlessly transitioning between gameplay and conclusion.

Soure Code

```
import pygame
import random


# Classes and Game Logic------------------------------------------------------------

# Use Class to integrate charactor
class Charactor:
    def __init__(self, name, health, attack_power, status):
        self.name = name
        self.health = health
        self.attack_power = attack_power
        # Status for defend set
        self.status = status
        self.special_cooldown = 0

    # Attack method
    def attack(self, other):
        # Let charactor stop defend status in a new turn
        self.status = "Normal"

        # When other is defended, half damage
        if other.status == "defend":
            other.health -= self.attack_power / 2
        else:
            other.health -= self.attack_power

    # Defend method
    def defend(self):
        self.status = "defend"

    # Special Ability method
    def special(self, other):
        # Could be used only cooldown
        if self.special_cooldown == 0:
            # Stop the defend status
            self.status = "Normal"
            # Double damage
            other.health -= self.attack_power * 2
            if other.status == "defend":
                other.health -= self.attack_power
            # Use three rounds apart, that is, the fourth round is available
```

```
                self.special_cooldown = 4
                return True
            else:
                return False

    # Enemy's action logic
    def AI(self, other):
        # When special cooldown, special first.
        if self.health < 50 and self.special_cooldown == 0:
            self.special(other)
            action_log.append(("Enemy used a special attack!", "enemy"))
            return "special"
        # When health < 50, increase the priority of defense
        elif self.health < 50:
            self.defend()
            action_log.append(("Enemy defended.", "enemy"))
            return "defend"
        elif self.special_cooldown == 0:
            self.special(other)
            action_log.append(("Enemy used a special attack!", "enemy"))
            return "special"
        else:
            # Normal condition, random choose
            action = random.choice(["Attack", "Defend"])
            if action == "Attack":
                self.attack(other)
                action_log.append((f"Enemy attacked for {self.attack_power} damage.",
"enemy"))
                return "attack"
            else:
                self.defend()
                action_log.append(("Enemy defended.", "enemy"))
                return "defend"

# class Game to manage the game conveniently
class Game:
    def __init__(self, player, enemy):
        self.player = player
        self.enemy = enemy
        # Default start with player
        self.turn = "player"

    # Switch turn between player and enemy
    def switch_turn(self):
```

```
            self.turn = "enemy" if self.turn == "player" else "player"

        # Check whether game is over
        def is_over(self):
            return self.player.health <= 0 or self.enemy.health <= 0

# Sound materials collection
class SoundManager:
        def __init__(self):
            pygame.mixer.init()
            self.bgm_playing = True
            self.bgm = "sound/bgmusic.mp3"
            self.sounds = {
                "attack": pygame.mixer.Sound("sound/attack.mp3"),
                "defend": pygame.mixer.Sound("sound/defend.mp3"),
                "win": pygame.mixer.Sound("sound/win.mp3"),
                "lose": pygame.mixer.Sound("sound/lose.mp3"),
                "special": pygame.mixer.Sound("sound/special.mp3")
            }
            self.volume = 0.5
            pygame.mixer.music.load(self.bgm)
            pygame.mixer.music.set_volume(self.volume)
            pygame.mixer.music.play(-1)

        #Toggle background music on or off
        def toggle_bgm(self):
            if self.bgm_playing:
                pygame.mixer.music.pause()
            else:
                pygame.mixer.music.unpause()
            self.bgm_playing = not self.bgm_playing

        # Play the corresponding sound effects
        def play_sound(self, sound_name):
            if sound_name in self.sounds:
                self.sounds[sound_name].set_volume(self.volume)
                self.sounds[sound_name].play()



# Initialization and Assets--------------------------------------------------------------------

pygame.init()
```

```
# Screen setup
screen = pygame.display.set_mode((1440, 900))
pygame.display.set_caption("A Game")

# Fonts, large font for title
font = pygame.font.Font(None, 30)
large_font = pygame.font.Font(None, 50)

# Images
bg_image = pygame.image.load("image/background.png")
music_icon = pygame.image.load("image/music.png")
mute_icon = pygame.image.load("image/mute.png")
player_image = pygame.image.load("image/player.png")
enemy_image = pygame.image.load("image/enemy.png")
attack_flag = pygame.image.load("image/attack.png")
defense_flag = pygame.image.load("image/defend.png")

# Colors will be used in this program
GRAY = (170, 170, 170)
DARK_GRAY = (100, 100, 100)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
WHITE = (255, 255, 255)

# Game variables
action_log = []
sound_manager = SoundManager()

# Instance Charactor and Game
player = Charactor("Player", 100, random.randint(10, 15), "Normal")
enemy = Charactor("Enemy", 100, random.randint(10, 15), "Normal")
game = Game(player, enemy)

# Initialize flag
player_flag = None
enemy_flag = None
player_flag_timer = 0
enemy_flag_timer = 0

# Start with Menu page
game_state = "menu"

# To start the main loop
running = True
```

```
# Define Functions-------------------------------------------------------------------------

# For special, it will CD with turns
def reduce_cooldowns():
    if player.special_cooldown > 0:
        player.special_cooldown -= 1
    if enemy.special_cooldown > 0:
        enemy.special_cooldown -= 1

# Draw Health Bar
def draw_health_bar(x, y, current_hp, label):
    max_hp = 100
    bar_width, bar_height = 400, 40
    health_ratio = current_hp / max_hp

    # Two color to indicate current hp and lost hp
    pygame.draw.rect(screen, RED, (x, y, bar_width, bar_height))
    pygame.draw.rect(screen, GREEN, (x, y, bar_width * health_ratio, bar_height))

    # Text to indicate current hp
    health_text = font.render(f"{label}: {current_hp}/{max_hp}", True, WHITE)
    screen.blit(health_text, (x, y + bar_height + 5))

# Draw Action Log
def draw_action_log():
    # Start position, lower right corner
    log_x, log_y = 1000, 750
    # Area size
    log_width, log_height = 400, 100
    log_surface = pygame.Surface((log_width, log_height), pygame.SRCALPHA)
    #Set transparency
    log_surface.fill((0, 0, 0, 128))
    screen.blit(log_surface, (log_x, log_y))

    y = log_y
    # Show lasted 5 message in action log
    for log, log_type in action_log[-5:]:
        # action from player is green, form enemy is red
        color = GREEN if log_type == "player" else RED
        text_surface = font.render(log, True, color)
        screen.blit(text_surface, (log_x, y))
        y += 20
```

```
# Draw Button
def draw_button(x, y, width, height, text, cooldown=0):
    # If mouse on button, give a highlight feedback
    mouse = pygame.mouse.get_pos()
    color = GRAY if x < mouse[0] < x + width and y < mouse[1] < y + height else DARK_GRAY
    # For special, if the CD is not ready, no light to indicate it can not be used
    if cooldown > 0:
        color = DARK_GRAY
    pygame.draw.rect(screen, color, (x, y, width, height))
    text_surface = font.render(text, True, WHITE)
    screen.blit(text_surface, (x + 10, y + 10))

# Start Menu
def draw_start_menu():
    title_text = large_font.render("A Game", True, WHITE)
    screen.blit(title_text, (1440 // 2 - 75, 330))
    draw_button(1440 // 2 - 100, 400, 200, 50, "Start Game")
    draw_button(1440 // 2 - 100, 500, 200, 50, "Exit")

# Display flags
def draw_flags():
    if player_flag and pygame.time.get_ticks() < player_flag_timer:
        screen.blit(player_flag, (250, 350))
    if enemy_flag and pygame.time.get_ticks() < enemy_flag_timer:
        screen.blit(enemy_flag, (900, 350))

# Reset the game to its initial state.
def restart_game():
    global player, enemy, game, action_log, game_state
    attack_power = random.randint(10, 15)
    player = Charactor("Player", 100, attack_power, "Normal")
    enemy = Charactor("Enemy", 100, attack_power, "Normal")
    game = Game(player, enemy)
    action_log.clear()
    game_state = "running"


# Main Game Loop-------------------------------------------------------------------------------

while running:
    # Load Background Image
    screen.blit(bg_image, (0, 0))
```

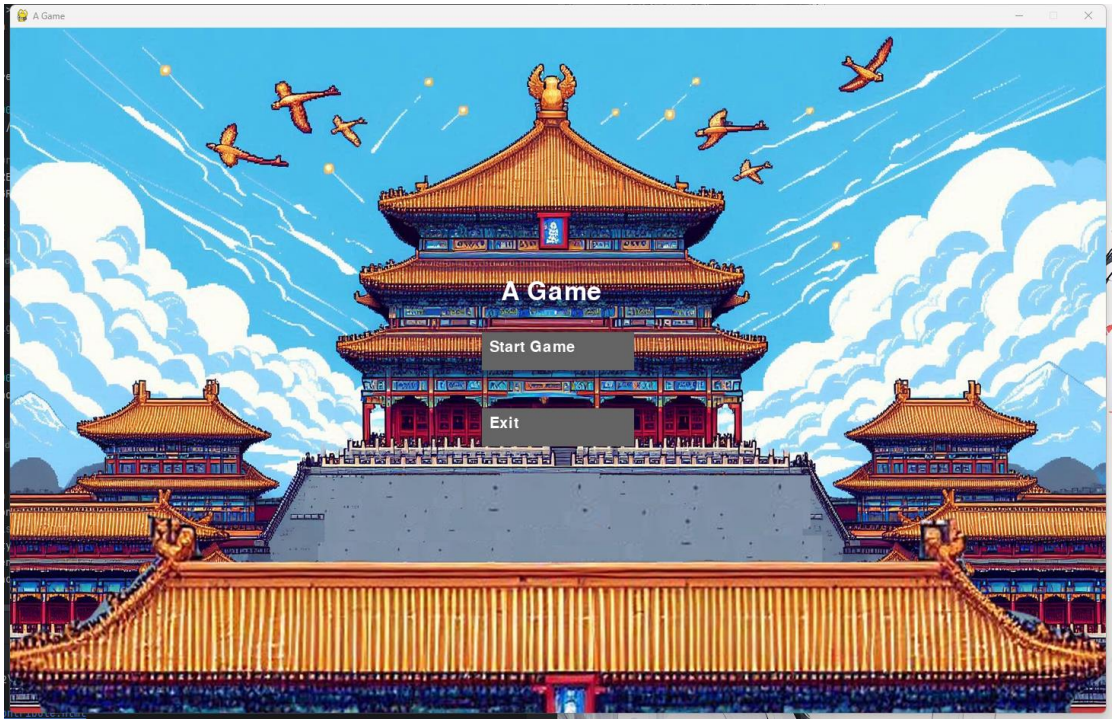
```
if game_state == "menu":
    # Draw the start menu
    draw_start_menu()
    for event in pygame.event.get():
        # Could stop the game by click "X" on windows
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.MOUSEBUTTONDOWN:
            # Button X coordinate
            if 1440 // 2 - 100 <= event.pos[0] <= 1440 // 2 + 100:
                # Y coordinate
                # Start button
                if 400 <= event.pos[1] <= 450:
                    restart_game()
                # Exit Button
                elif 500 <= event.pos[1] <= 550:
                    running = False
# Game page
elif game_state == "running":
    # Create UI
    draw_health_bar(50, 50, player.health, "Player Health")
    draw_health_bar(1000, 50, enemy.health, "Enemy Health")
    draw_action_log()
    draw_flags()

    draw_button(50, 400, 100, 40, "Attack")
    draw_button(50, 500, 100, 40, "Defend")
    draw_button(50, 600, 100, 40, "Special", player.special_cooldown)

    # Music button image call
    if sound_manager.bgm_playing:
        screen.blit(music_icon, (10, 850))
    # Mute button image call
    else:
        screen.blit(mute_icon, (10, 850))

    # Load charactor image
    screen.blit(player_image, (200, 500))
    screen.blit(enemy_image, (1000, 500))

    # Player's handle
    for event in pygame.event.get():
        # Close window to stop the game
```

```
        if event.type == pygame.QUIT:
            running = False
        # Click action get
        if event.type == pygame.MOUSEBUTTONDOWN:
            # Mute and play music switch
            if 10 <= event.pos[0] <= 60 and 850 <= event.pos[1] <= 900:
                sound_manager.toggle_bgm()

            # Game Play, player's turn
            elif game.turn == "player":
                # Attack button area
                if 50 <= event.pos[0] <= 150 and 400 <= event.pos[1] <= 440:
                    player.attack(enemy)
                    # Add action into log
                    action_log.append(("Player attacked!", "player"))
                    # Music effect call
                    sound_manager.play_sound("attack")
                    # Flag Call
                    player_flag = attack_flag
                    player_flag_timer = pygame.time.get_ticks() + 1000
                    # Switch turn
                    game.switch_turn()
                # Defense button area
                elif 50 <= event.pos[0] <= 150 and 500 <= event.pos[1] <= 540:
                    player.defend()
                    action_log.append(("Player defended!", "player"))
                    sound_manager.play_sound("defend")
                    player_flag = defense_flag
                    player_flag_timer = pygame.time.get_ticks() + 1000
                    game.switch_turn()
                elif 50 <= event.pos[0] <= 150 and 600 <= event.pos[1] <= 640:
                    # Check CD
                    if player.special(enemy):
                        action_log.append(("Player used special!", "player"))
                        sound_manager.play_sound("special")
                        player_flag = attack_flag
                        player_flag_timer = pygame.time.get_ticks() + 1000
                        game.switch_turn()

    # Enemy's turn
    if game.turn == "enemy" and not game.is_over():
        # Receive an action
        enemy_action = enemy.AI(player)
        if enemy_action == "attack":
```

```
                    enemy_flag = attack_flag
                elif enemy_action == "defend":
                    enemy_flag = defense_flag
                elif enemy_action == "special":
                    enemy_flag = attack_flag
                    # Enemy's other action does not have music effect, but special has, to indicate player a higher damage
                    sound_manager.play_sound("special")
                enemy_flag_timer = pygame.time.get_ticks() + 1000
                reduce_cooldowns()
                game.switch_turn()

            # When Game Over, set winner and Call music effect
            if game.is_over():
                winner = "Player" if player.health > 0 else "Enemy"
                sound_manager.play_sound("win" if winner == "Player" else "lose")
                game_state = "end"

        # End page
        elif game_state == "end":
            # End state
            winner = "Player" if player.health > 0 else "Enemy"

            # Indicate the Winner
            title_text = large_font.render(f"{winner} Wins!", True, WHITE)
            screen.blit(title_text, (1440 // 2-100, 350))

            # Create Button to replay and quit
            draw_button(1440 // 2 - 100, 400, 200, 50, "Replay")
            draw_button(1440 // 2 - 100, 500, 200, 50, "Quit")
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                if event.type == pygame.MOUSEBUTTONDOWN:
                    # Check Button area, Same X coordinate range
                    if 1440 // 2 - 100 <= event.pos[0] <= 1440 // 2 + 100:
                        # Replay button Y coordinate range
                        if 400 <= event.pos[1] <= 450:
                            restart_game()
                        # Quit button Y coordinate range
                        elif 500 <= event.pos[1] <= 550:
                            running = False
        # Refresh the screen
        pygame.display.flip()
```

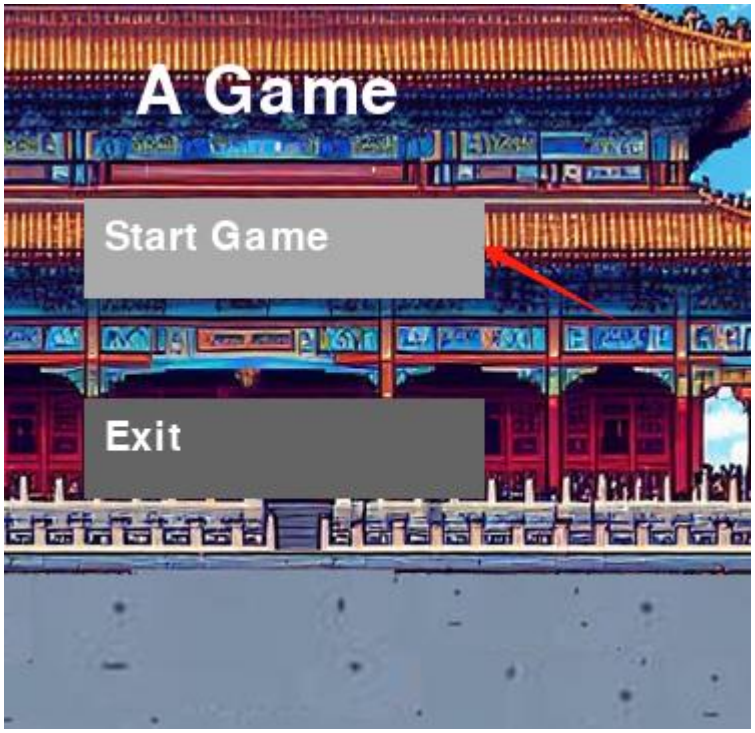
ScreenShot

Menu

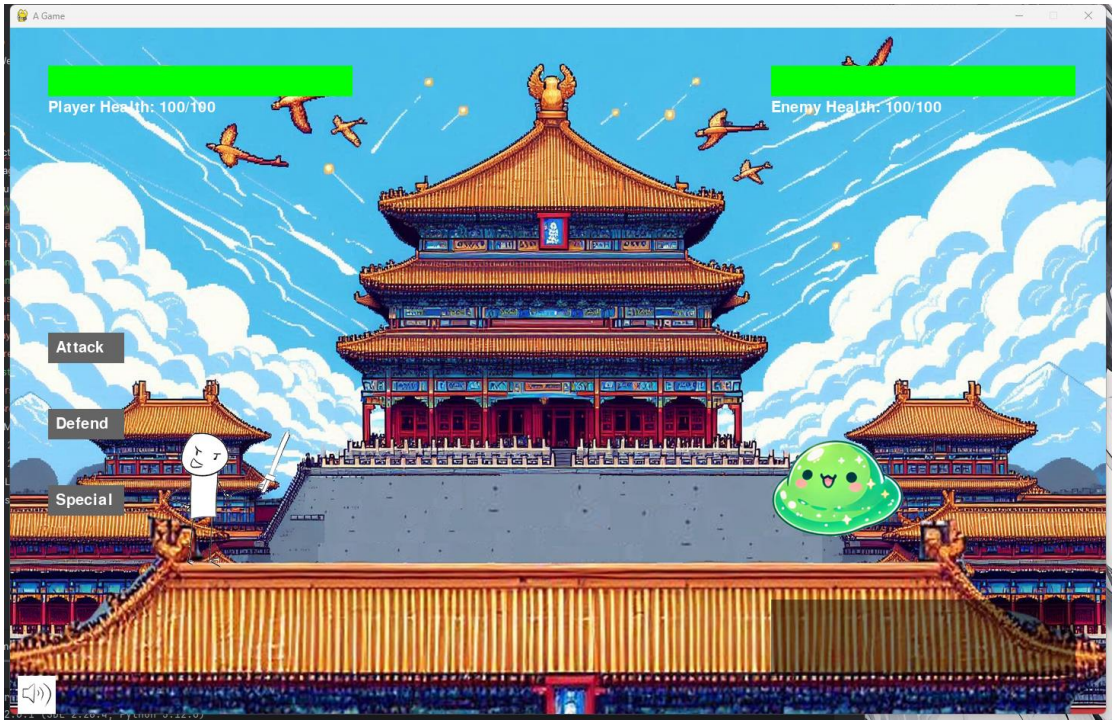


Button Feedback:

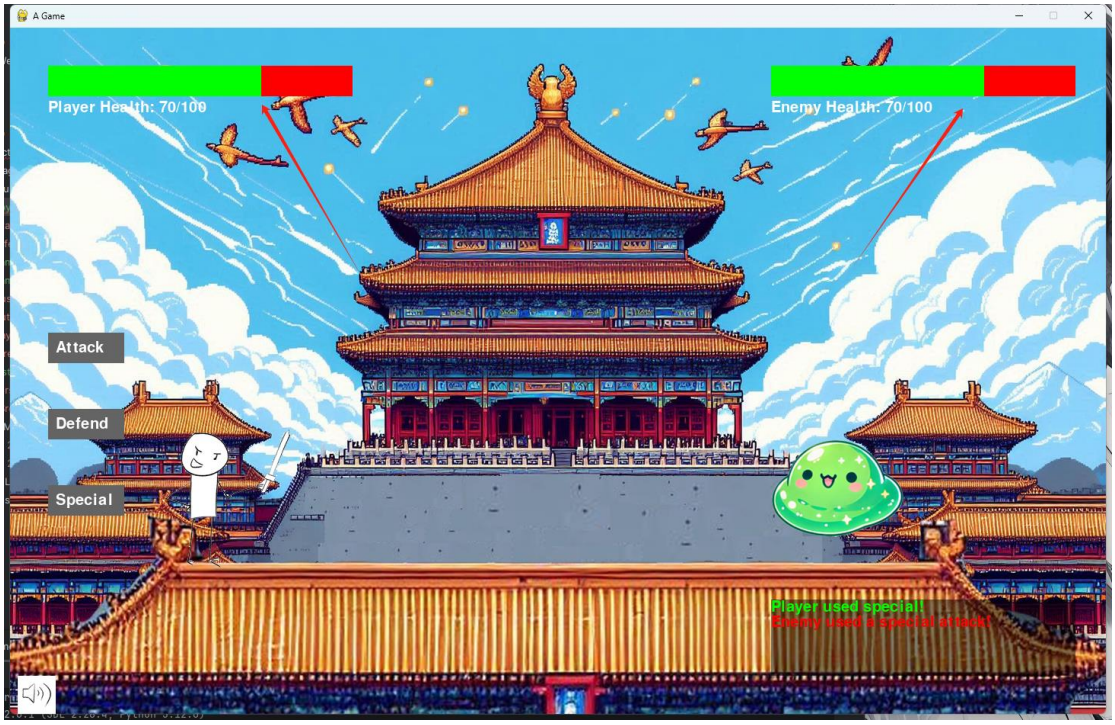
When mouse on a button, change color lighter



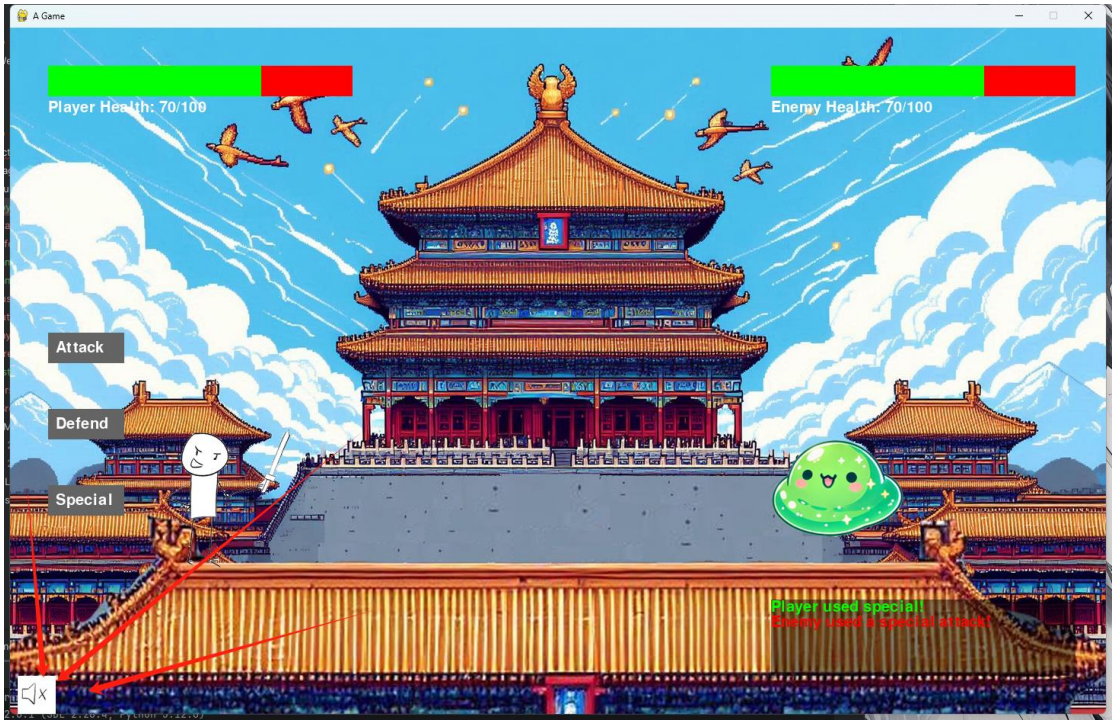
Game



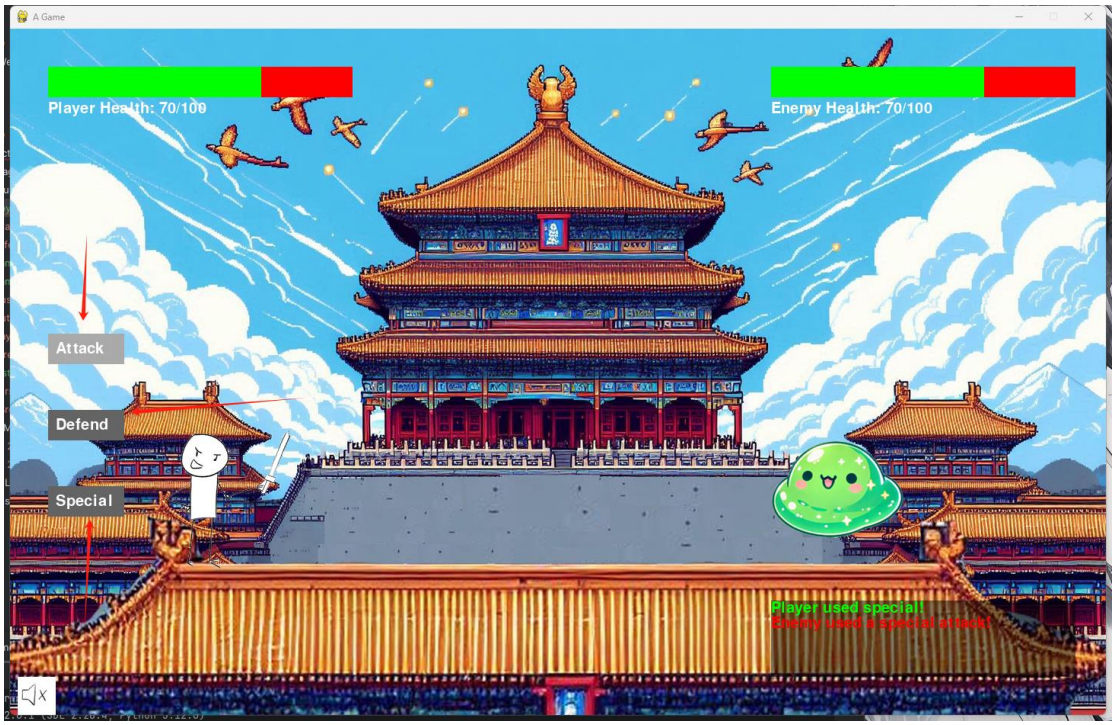
Helath Bar



Mute button:



Action Button

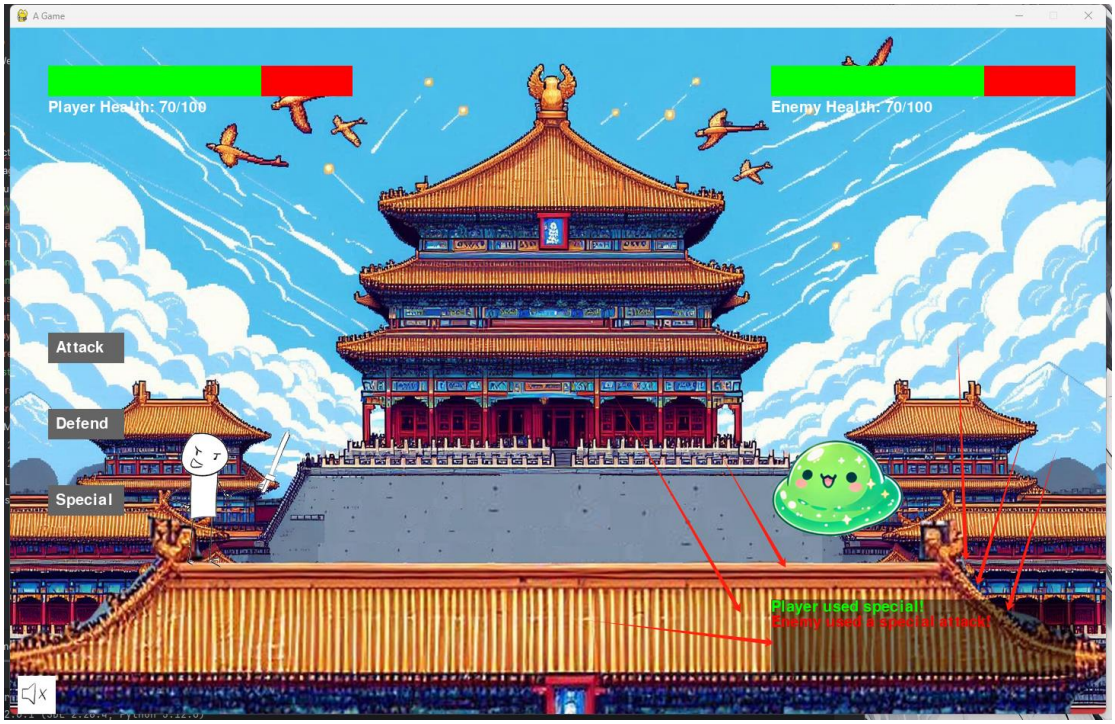


Characters Image

Player

Enemy

Action Log

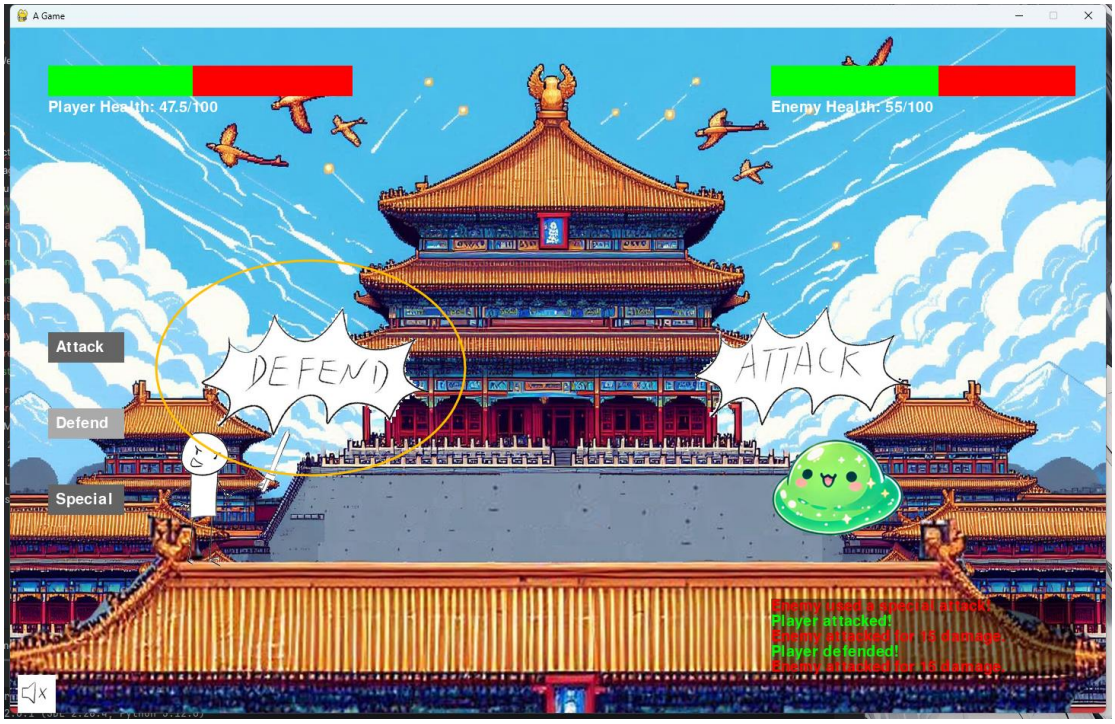


Attack flag:

as animation

Defend flag

as animation



END

Version Log

Important

Many details of debug code changes could not be displayed because there was no screenshot at that time, so the text description in this file is from my memory, and the code screenshot is from a part of my test file that is still retained (there is also a lot of code that I deleted after the test was completed)

Initial Version

A basic structure of a turn-based game using pygame, just basic 3 buttons, but the system will automatically compute all turns after player do the first acton, I can not fix this, so totally give up this version and change a thought to rerwrte another version.

Initial Code

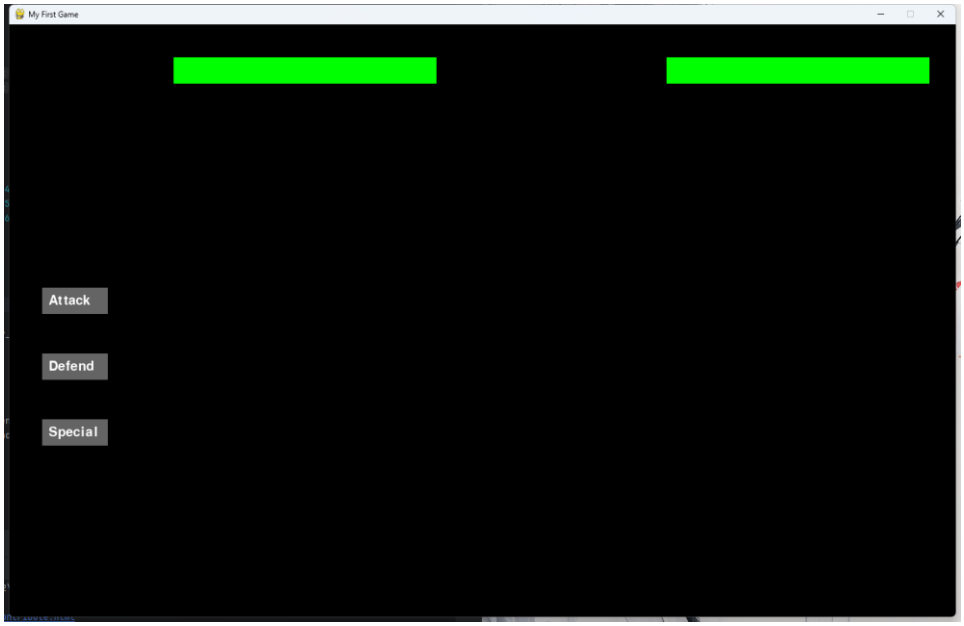
```python
import pygame
import sys
import random

pygame.init()

# Create Windows
screen = pygame.display.set_mode((1440, 900))
pygame.display.set_caption("My First Game")

# Fonts
font = pygame.font.Font( name: None, size: 30)

# Def color
RED = (255, 0, 0)
GREEN = (0, 255, 0)
GRAY = (170, 170, 170)
DARK_GRAY = (100, 100, 100)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# Variables
player_hp = 100
enemy_hp = 100
turn_count = 1
player_turn = True
action_log = []

# Define Functions
# Enemy AI
def enemy_AI():  1 usage  new *
    global player_hp,player_turn
    action = random.choice(["Attack", "Defend"])
    if action == "Attack":
        damage = random.randint( a: 10, b: 15)
        player_hp -= damage
        action_log.append(f'Enemy attacked for {damage} damage.')

    else:
        action_log.append("Enemy defended.")

    player_turn = True
```

```python
# End Turn
def end_turn():  3 usages  new *
    global player_turn, turn_count
    player_turn = False
    turn_count += 1

# Attack
def attack_action():  1 usage  new *
    global enemy_hp, player_turn

    damage = random.randint( a: 10,  b: 15)
    enemy_hp -= damage
    action_log.append(f'Player attacked for {damage} damage.')
    end_turn()

# Defend
def defend_action():  1 usage  new *
    global player_turn
    action_log.append("Player defended.")
    end_turn()

# Special Attack
def special_action():  1 usage  new *
    global enemy_hp, player_turn
    damage = random.randint( a: 20,  b: 30)
    enemy_hp -= damage
    action_log.append(f"Player used a special attack for {damage} damage.")
    end_turn()

# Health Bar
def draw_health_bar(x,y, current_hp):  2 usages  new *
    max_hp = 100
    bar_width, bar_height = 400,40
    health_ratio = current_hp / max_hp
    pygame.draw.rect(screen, RED,  rect: (x, y, bar_width, bar_height))
    pygame.draw.rect(screen,GREEN,  rect: (x,y, bar_width * health_ratio, bar_height))
```


```python
# End Turn
def end_turn():  3 usages  new *
    global player_turn, turn_count
    player_turn = False
    turn_count += 1

# Attack
def attack_action():  1 usage  new *
    global enemy_hp, player_turn

    damage = random.randint( a: 10,  b: 15)
    enemy_hp -= damage
    action_log.append(f'Player attacked for {damage} damage.')
    end_turn()

# Defend
def defend_action():  1 usage  new *
    global player_turn
    action_log.append("Player defended.")
    end_turn()

# Special Attack
def special_action():  1 usage  new *
    global enemy_hp, player_turn
    damage = random.randint( a: 20,  b: 30)
    enemy_hp -= damage
    action_log.append(f"Player used a special attack for {damage} damage.")
    end_turn()

# Health Bar
def draw_health_bar(x,y, current_hp):  2 usages  new *
    max_hp = 100
    bar_width, bar_height = 400,40
    health_ratio = current_hp / max_hp
    pygame.draw.rect(screen, RED,  rect: (x, y, bar_width, bar_height))
    pygame.draw.rect(screen,GREEN,  rect: (x,y, bar_width * health_ratio, bar_height))
```

```python
# End Turn
def end_turn():  3 usages  new *
    global player_turn, turn_count
    player_turn = False
    turn_count += 1

# Attack
def attack_action():  1 usage  new *
    global enemy_hp, player_turn

    damage = random.randint( a: 10,  b: 15)
    enemy_hp -= damage
    action_log.append(f'Player attacked for {damage} damage.')
    end_turn()

# Defend
def defend_action():  1 usage  new *
    global player_turn
    action_log.append("Player defended.")
    end_turn()

# Special Attack
def special_action():  1 usage  new *
    global enemy_hp, player_turn
    damage = random.randint( a: 20,  b: 30)
    enemy_hp -= damage
    action_log.append(f"Player used a special attack for {damage} damage.")
    end_turn()

# Health Bar
def draw_health_bar(x,y, current_hp):  2 usages  new *
    max_hp = 100
    bar_width, bar_height = 400,40
    health_ratio = current_hp / max_hp
    pygame.draw.rect(screen, RED,  rect: (x, y, bar_width, bar_height))
    pygame.draw.rect(screen,GREEN,  rect: (x,y, bar_width * health_ratio, bar_height))
```

Initial Game



Version 2

This version is the basic version of my final program. Black background, health bar, three button, no action log, no any image, no any sounds.

Version 2.1

Add action log

```
def draw_action_log():  1 usage  new *
    y = 450
    for log in action_log[-5:]:
        text_surface = font.render(log, antialias: True, WHITE)
        screen.blit(text_surface, dest: (1000, y))
        y += 20
```

```
if event.type == pygame.MOUSEBUTTONDOWN and game.turn == 'player':
    if 50 <= event.pos[0] <= 150 and 400 <= event.pos[1] <= 440:
        player.attack(enemy)
        action_log.append(f"Player attacked for {player.attack_power} damage.")
    elif 50 <= event.pos[0] <= 150 and 500 <= event.pos[1] <= 540:
        player.defend()
        action_log.append("Player defended.")
    elif 50 <= event.pos[0] <= 150 and 600 <= event.pos[1] <= 640:
        player.special(enemy)
        action_log.append(f"Player used special for {player.attack_power * 2} damage.")
    game.switch_turn()
```

Version 2.2

Try to let enemy wait 1 second to action, failed, delete.

```
if game.turn == 'enemy' and not game.is_over():
    pygame.time.wait(1000)  # Wait 1 second (1000 milliseconds) before the Enemy reacts
    enemy.AI(player)
    action_log.append(
        f"Enemy attacked for {enemy.attack_power} damage." if enemy.status == "Normal" else "Enemy defended.")
    game.switch_turn()
```

```
        game.switch_turn()
enemy_action_timer = pygame.time.get_ticks() + 1000
```

```
current_time = pygame.time.get_ticks()
if enemy_action_timer and current_time >= enemy_action_timer:
```

Version 2.3

Add background image.

Add characters image, error:"libpng warning: iCCP: known incorrect sRGB profile",

Fixed: In PhotoShop, save as for web

```
player_image = pygame.image.load("image/player.pny")
enemy_image = pygame.image.load("Image/enemy.png")
```

Version 2.4

Make the action log translucency

```
log_surface.fill((0, 0, 0, 128))
```

Version 2.5

Add CD for sepcial

Add cool down in calss

```
class Charactor:  new *
    def __init__(self, name, health, attack_power, status):  new *
        self.name = name
        self.health = health
        self.attack_power = attack_power
        self.status = status
        self.special_cooldown = 0  # Start with no cooldown
```

Modify the special Method

```python
def special(self, other):  new *
    if self.special_cooldown == 0:
        self.status = "Normal"
        other.health -= self.attack_power * 2
        if other.status == "defend":
            other.health -= self.attack_power
        self.special_cooldown = 3
        return True
    else:
        return False
```

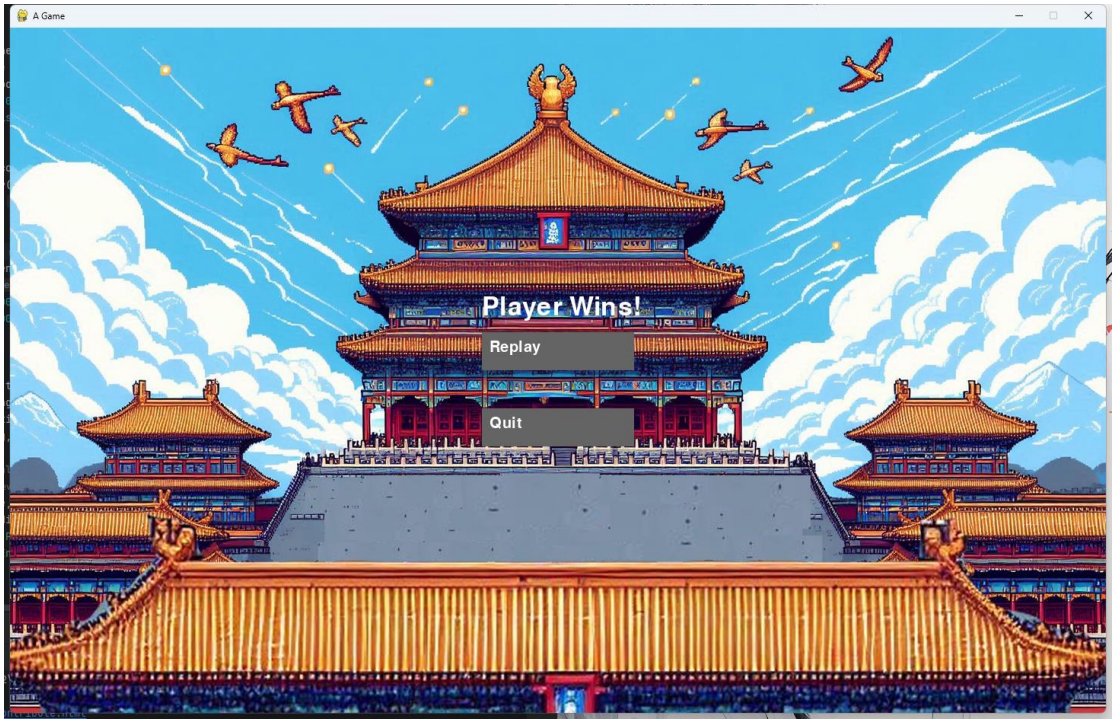
Game result

Version 2.6

Add end page

```
def draw_end_screen(winner):  new *
    # Display the winner message
    text_surface = font.render( text: f"{winner} Wins!",  antialias: True, WHITE)
    screen.blit(text_surface,  dest: (1440 // 2 - 100, 900 // 2 - 100))

    # Draw Replay button
    draw_button(1440 // 2 - 150, 900 // 2,  width: 100,  height: 40,  text: "Replay")

    # Draw Quit button
    draw_button(1440 // 2 + 50, 900 // 2,  width: 100,  height: 40,  text: "Quit")
```

Game result



Version 2.7

Add flag for attack and defense

```
attack_flag = pygame.image.load("image/attack.png")
defense_flag = pygame.image.load("image/defense.png")
```

```
player_flag = None
enemy_flag = None
player_flag_timer = 0
enemy_flag_timer = 0
```

Version 2.8

Try to add animation, failed, image move tooooooooooooooo fast, can not slow down.

Version 2.9

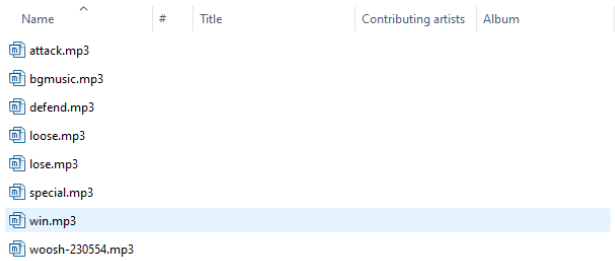
Enhance enemy AI, set prefer special and Defense when hp < 50

```
def AI(self, other):  new *
    if self.health < 50 and self.special_cooldown == 0:
        self.special(other)
    elif self.health < 50:
        self.defend()
    elif self.special_cooldown == 0:
        self.special(other)
    else:
        action = random.choice(["Attack", "Defend"])
        if action == "Attack":
            self.attack(other)
        else:
            self.defend()
```

Version 2.10

Add music effect

```
# Sound materials collection
class SoundManager:  1 usage  new *
    def __init__(self):  new *
        pygame.mixer.init()
        self.bgm_playing = True
        self.bgm = "sound/bgmusic.mp3"
        self.sounds = {
            "attack": pygame.mixer.Sound("sound/attack.mp3"),
            "defend": pygame.mixer.Sound("sound/defend.mp3"),
            "win": pygame.mixer.Sound("sound/win.mp3"),
            "lose": pygame.mixer.Sound("sound/lose.mp3"),
            "special": pygame.mixer.Sound("sound/special.mp3")
        }
        self.volume = 0.5
        pygame.mixer.music.load(self.bgm)
        pygame.mixer.music.set_volume(self.volume)
        pygame.mixer.music.play(-1)

    #Toggle background music on or off
    def toggle_bgm(self):  1 usage  new *
        if self.bgm_playing:
            pygame.mixer.music.pause()
        else:
            pygame.mixer.music.unpause()
        self.bgm_playing = not self.bgm_playing

    # Play the corresponding sound effects
    def play_sound(self, sound_name):  5 usages  new *
        if sound_name in self.sounds:
            self.sounds[sound_name].set_volume(self.volume)
            self.sounds[sound_name].play()
```



Version 2.11

Fix but, when I click the mute button, trun skipped. Change the mouse action get

```
if event.type == pygame.MOUSEBUTTONDOWN:
    # Check if click was on the mute button
    if 10 <= event.pos[0] <= 60 and 850 <= event.pos[1] <= 900:  # Mute button area
        sound_manager.toggle_bgm()
        continue  # Do not process further actions for this click

    # Handle player's turn actions
    if game.turn == "player":
        if 50 <= event.pos[0] <= 150 and 400 <= event.pos[1] <= 440:  # Attack
            player.attack(enemy)
            action_log.append(f"Player attacked for {player.attack_power} damage.")
            sound_manager.play_sound("attack")
            player_flag = attack_flag
            player_flag_timer = pygame.time.get_ticks() + 1000  # Show flag for 1 second
        elif 50 <= event.pos[0] <= 150 and 500 <= event.pos[1] <= 540:  # Defend
            player.defend()
            action_log.append("Player defended.")
            sound_manager.play_sound("defend")
            player_flag = defense_flag
            player_flag_timer = pygame.time.get_ticks() + 1000  # Show flag for 1 second
        elif 50 <= event.pos[0] <= 150 and 600 <= event.pos[1] <= 640:  # Special
            if player.special(enemy):
                action_log.append(f"Player used special for {player.attack_power * 2} damage.")
                sound_manager.play_sound("attack")
                player_flag = attack_flag
                player_flag_timer = pygame.time.get_ticks() + 1000  # Show flag for 1 second
            else:
                action_log.append("Special is on cooldown!")
        game.switch_turn()
```

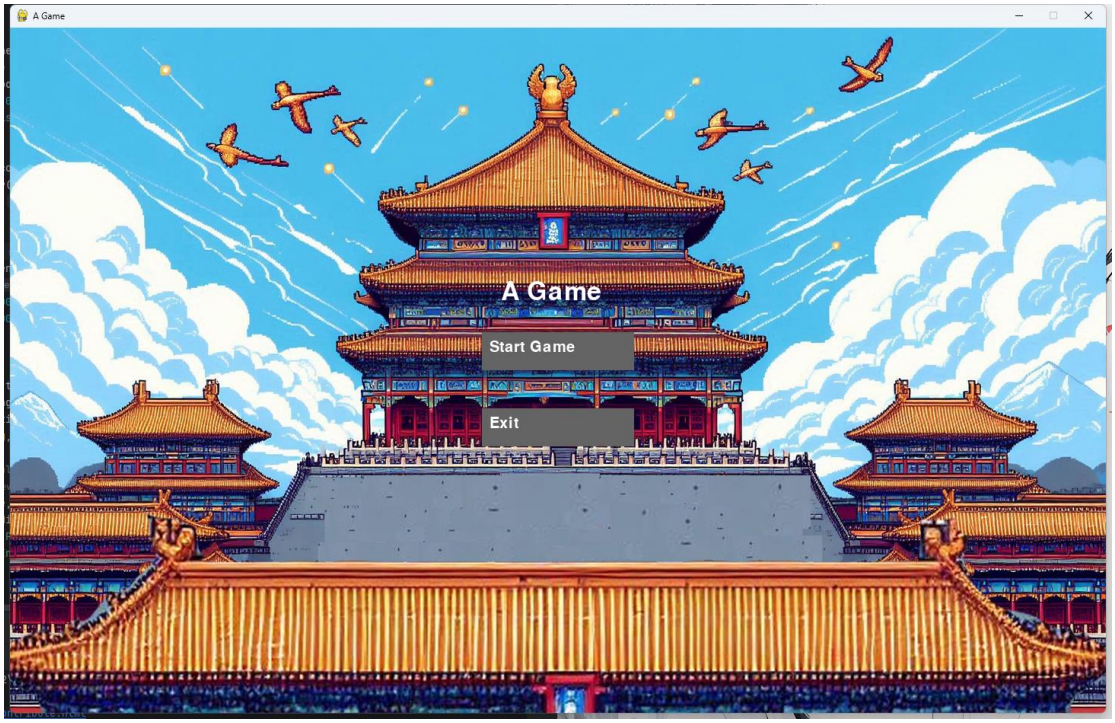
Version 3.0

Add Start Menu, add text under health bar

```
# Start Menu
def draw_start_menu():  1 usage  new *
    title_text = large_font.render( text: "A Game",  antialias: True, WHITE)
    screen.blit(title_text,  dest: (1440 // 2 - 75, 330))
    draw_button(1440 // 2 - 100,  y: 400,  width: 200,  height: 50,  text: "Start Game")
    draw_button(1440 // 2 - 100,  y: 500,  width: 200,  height: 50,  text: "Exit")
```

```
# Text to indicate current hp
health_text = font.render( text: f"{label}: {current_hp}/{max_hp}",  antialias: True, WHITE)
screen.blit(health_text,  dest: (x, y + bar_height + 5))
```

Game result



Test File

```python
from Game1 import Charactor, Game, SoundManager


# Test the Charactor class
def test_charactor_attack():  new *
    player = Charactor( name: "Player",  health: 100,  attack_power: 20,  status: "Normal")
    enemy = Charactor( name: "Enemy",  health: 100,  attack_power: 20,  status: "Normal")

    # Player attacks Enemy
    player.attack(enemy)
    assert enemy.health == 80  # Full damage applied
    assert player.status == "Normal"  # Status reset after attack

    # Enemy defends and Player attacks
    enemy.defend()
    player.attack(enemy)
    assert enemy.health == 70  # Half damage applied when defending


def test_charactor_defend():  new *
    char = Charactor( name: "Player",  health: 100,  attack_power: 20,  status: "Normal")
    char.defend()
    assert char.status == "defend"  # Status set to defend


def test_charactor_special():  new *
    player = Charactor( name: "Player",  health: 100,  attack_power: 20,  status: "Normal")
    enemy = Charactor( name: "Enemy",  health: 100,  attack_power: 20,  status: "Normal")

    # Use special when not on cooldown
    result = player.special(enemy)
    assert result is True
    assert enemy.health == 60  # Double damage applied
    assert player.special_cooldown == 4  # Cooldown set

    # Attempt to use special when on cooldown
    result = player.special(enemy)
    assert result is False
```

```python
def test_charactor_AI():  new *
    enemy = Charactor( name: "Enemy",  health: 30,  attack_power: 20,  status: "Normal")
    player = Charactor( name: "Player",  health: 100,  attack_power: 20,  status: "Normal")

    # Special attack when health < 50
    action = enemy.AI(player)
    assert action == "special"
    assert player.health == 60  # Double damage applied

    # Defend when health < 50 and special on cooldown
    enemy.special_cooldown = 4
    action = enemy.AI(player)
    assert action == "defend"
    assert enemy.status == "defend"


# Test the Game class
def test_game_switch_turn():  new *
    player = Charactor( name: "Player",  health: 100,  attack_power: 20,  status: "Normal")
    enemy = Charactor( name: "Enemy",  health: 100,  attack_power: 20,  status: "Normal")
    game = Game(player, enemy)

    # Switch turns
    assert game.turn == "player"
    game.switch_turn()
    assert game.turn == "enemy"
    game.switch_turn()
    assert game.turn == "player"


def test_game_is_over():  new *
    player = Charactor( name: "Player",  health: 0,  attack_power: 20,  status: "Normal")  # Player has 0 health
    enemy = Charactor( name: "Enemy",  health: 100,  attack_power: 20,  status: "Normal")
    game = Game(player, enemy)

    # Check if the game is over
    assert game.is_over() is True

    player.health = 50
    enemy.health = 0  # Enemy has 0 health
    assert game.is_over() is True

    player.health = 50
    enemy.health = 50  # Both players have health
    assert game.is_over() is False
```

```python
# Test the SoundManager class
def test_sound_manager_toggle_bgm(mocker):  new *
    mocker.patch("pygame.mixer.music.pause")
    mocker.patch("pygame.mixer.music.unpause")

    sound_manager = SoundManager()
    assert sound_manager.bgm_playing is True

    # Toggle background music off
    sound_manager.toggle_bgm()
    assert sound_manager.bgm_playing is False

    # Toggle background music on
    sound_manager.toggle_bgm()
    assert sound_manager.bgm_playing is True


def test_sound_manager_play_sound(mocker):  new *
    mocker.patch("pygame.mixer.Sound.play")

    sound_manager = SoundManager()
    sound_manager.play_sound("attack")
    assert "attack" in sound_manager.sounds
```

Edge Cases and Limitations

One problem with the game is that the enemy's actions take place immediately after the end of the player's turn, which is too fast, making it appear that the player and the enemy are acting at the same time. If you actually develop the game later, the time of the set up turns and the time of the enemy's action will be improved. Another problem was the failure of animation addition. Since I added the flag, I assumed that when the character attacked, the flag would appear and move with the player to achieve the animation effect. As a result, the character image and the flag image could not be fully synchronized, and the character image moved too fast and seemed to flash, so I had to delete the animation effect. But I think flag can also be seen as an animation, and I like it.

The enemy AI, while functional, operates on relatively simple decision-making logic based primarily on health and cooldowns. This simplicity could make it predictable over time. Lastly, the game’s design is resolution-dependent, optimized specifically for 1440x900 screens. Users on devices with smaller or larger screens may experience misaligned elements, reducing visual fidelity.

Future Improvements

The addition of equipment and props will significantly improve the integrity of the game, but the complexity of programming will also increase accordingly. At present, there is no means to restore HP in the game, which is too monotonous for combat, adding props or skills to restore HP can improve the situation.

Turn-based battles are mostly applicable to RPGs, and experience value is one of the most important elements of RPGs, if you need to expand the game's country model in the future, adding experience value and leveling system is also a good choice.

In terms of gameplay, as I said in the description section, class Charactor makes it relatively easy to add multiple characters for the player to choose from, and the player's choice of characters with different attributes and abilities to fight against the enemy will make the game significantly more playable.