Project: Infinity Castle

A Demon Slayer-Themed Angry Birds Game

Hari Hara Teja

April 29, 2025

A two-player game inspired by Angry Birds and Demon Slayer

Contents

1 Introduction

1.1 Project overview

This is a simple project of creating a two-player angry birds game using Python and the Pygame module. I am interested in the Demon Slayer anime. So, I wanted to modify the game with Demon Slayer characters. The objective of the project is to learn the Pygame libraries and handle large codebases in different files and modules.

1.2 Instructions to run the game

- Install Python 3.8 or above
- Install Pygame module using `pip install pygame`
- Download the project files and extract them
- Open the terminal and navigate to the project directory
- Run the command `python main.py`
- The game will start and open the main interface

1.3 Game play

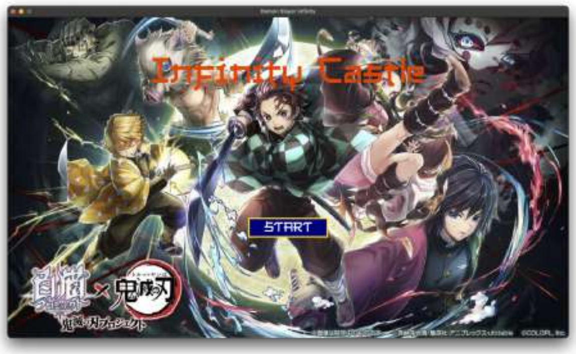

Figure 1: Start screen

- Click on the start button to start the game (Figure 1)

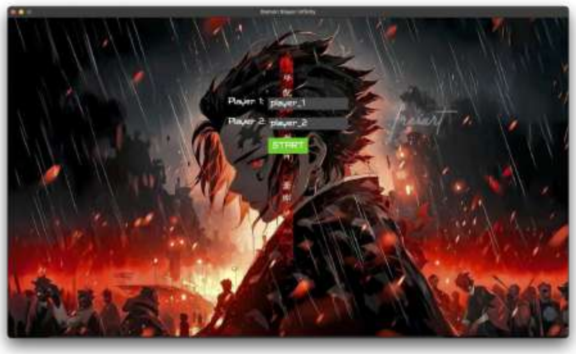

Figure 2: Main menu

- Now you can enter your names and hit the start button to select the characters of your choice (Figure 2)

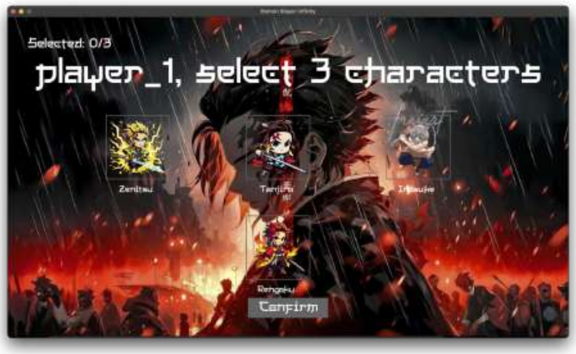

Figure 3: Character selection screen

- On screen you can select three characters out of 4 characters for each player (Figure 3)
- Please select characters carefully as each has different abilities and makes different damage to the different blocks
- Rengoku will damage all the blocks equally
- Tanjiro will damage the block type-ice more than the other blocks
- Zenitsu will damage the block type-wood more than the other blocks
- Inosuke will damage the block type-stone more than the other blocks
- Ice type has health of 25
- Wood type has health of 50
- Stone type has health of 75
- The blocks are the actual villains in the game and they will take damage based on the character deployed

- Click on the confirm button to confirm your selection and start your game

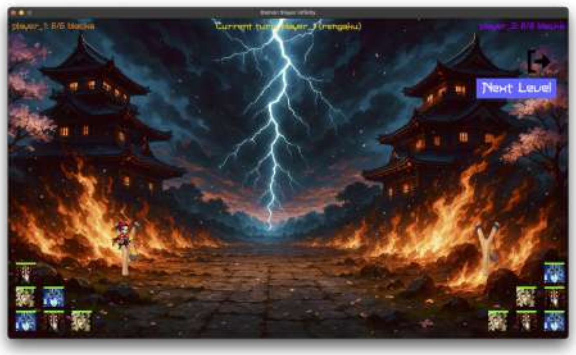

Figure 4: Playing the game

- The game will start and you can play the game using the mouse
- Place the mouse on the characters and drag the characters to deploy them. When hitting the blocks, blocks take damage based on the character deployed (Figure 4)
- The game will end when all the blocks are destroyed or all the characters are deployed

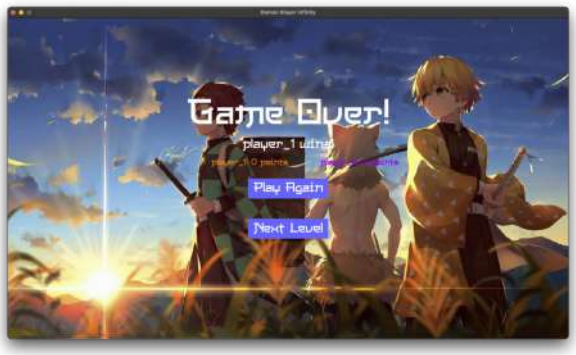

Figure 5: Game over screen

- The game will show the score and the winner of the game. The winner will be decided based on minimum number of characters deployed (Figure 5)
- Click on the play again button to play the game again
- Click on the exit button to exit the game
- Click on the next level button to play the next level of the game or you can exit the game

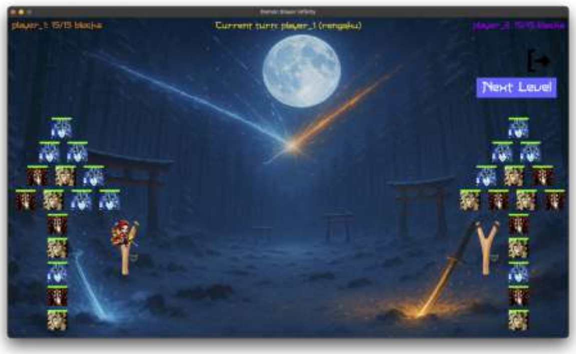

Figure 6: Next level screen

1.4 Key features

- Two player game
- Each character has its own unique abilities
- Each character has its own unique damage
- Each block has its own unique health, they are the actual villains in the anime with health allocated based on their hierarchy
- To increase the fun factor I have introduced some special effects for the blocks and characters
- Rengoku resembles the fire and is the strongest character in the game
- Tanjiro resembles the crimson flame and is the main character in the anime
- Zenitsu resembles the thunder and is a friend of Tanjiro
- Inosuke resembles the wild boar and is the friend of Tanjiro
- There are three villain types in the game
- Each character has its own unique sound effects

2 Project structure

2.1 Directory layout

The project follows a standard directory layout for Python projects. The main files and directories are as follows:

```
infinity_castle/
+-- main.py
+-- declare.py
+-- calculations.py
+-- blocks.py
+-- player.py
+-- play.py
+-- set.py
+-- interface.py
+-- images/
|   +-- start.jpg
|   +-- main_menu.jpeg
|   +-- battlefield.jpeg
|   +-- next_level.png
|   +-- game_over.jpeg
|   +-- block1.png
|   +-- block2.png
|   +-- block3.png
|   +-- rengoku.png || rengoku!.png
|   +-- tanjiro.png || tanjiro!.png
|   +-- zenitsu.png || zenitsu!.png
|   +-- inosuke.png || inosuke!.png
|   +-- special_block.png
+-- music/
|   +-- demonslayertheme.mp3
|   +-- playing_ds.mp3
|   +-- game_over_bgm.mp3
|   +-- rengoku_bgm.mp3
|   +-- tanjiro_bgm.mp3
|   +-- zenitsu_bgm.mp3
|   +-- inosuke_bgm.mp3
|   +-- special_hit_bgm.mp3
+-- start.otf
```

2.2 File descriptions

- **main.py**: The main file that runs the game. It initializes the game and starts the main loop.
- **declare.py**: This file contains all the global variables and constants used in the game.
- **calculations.py**: This file contains all the calculations and logic for the game.

- **blocks.py**: This file contains the class for the effects for blocks used in the game.
- **player.py**: This file contains the class for the players used in the game.
- **play.py**: This file contains the class for the game play.
- **set.py**: This file contains all the settings and configurations for the player.
- **interface.py**: This file contains all the interface related code for the game.
- **images/**: This directory contains all the images used in the game.
- **music/**: This directory contains all the music files used in the game.
- **start.otf**: This file contains the font used in the game.

2.3 Module descriptions

- **pygame**: This module is used for creating the game window and handling the game events.
- **random**: This module is used for generating random numbers.
- **numpy**: This module is used for handling the arrays and matrices.
- **math**: This module is used for handling the mathematical calculations.

3 Class descriptions

- **ui**: This class is used for handling the user interface of the game. It contains all the methods for drawing the interface and handling the buttons.
  - **__init__**: This method is used for initializing the ui class. It contains fonts required for the game.
  - **Methods:**
    - **draw_menu**: This method is used for drawing the main menu of game. It contains the start button and background image.
    - **draw_player_info**: This method is used for drawing the player information layout on the screen it contains the player name and start button.
    - **take_input**: This method is used for taking the player input from the user. It contains the text box and the start button.
    - **draw_character_selection**: This method is used for drawing the character selection screen. It contains the character images and the confirm button.
    - **draw_game_over**: This method is used for drawing the game over screen. It contains the game over image, winner name, scores, play again button and next level button.
- **declare**: This class is used for handling the global variables and constants used in the game. It contains all the global variables and constants used in the game.
  - **__init__**: This method is used for initializing the declare class. It contains all the global variables and constants used in the game.
- **player**: This class is used for handling the player information and the player actions. It contains all the methods for handling the player actions.
  - **__init__**: This method is used for initializing the player class. It contains the character type, initial position, position, velocity, wind, etc.
  - **Methods:**
    - **load_image**: This method is used for loading the character image. It contains the character type and the image path.
    - **start_drag**: This method is used for starting the drag of the character. It contains the mouse position and the character image.
    - **update_drag**: This method is used for updating the drag of the character. It contains the mouse position and the character image.
    - **end_drag**: This method is used for ending the drag of the character. It contains the mouse position and the character image.
    - **update**: This method is used for updating the character position, velocity, wind etc.
    - It also contains the methods for drawing trajectory, updating wind particles, creating wind particles.
- **blocks**: This class is used for handling the blocks in the game. It contains all the methods for handling the blocks.

- __init__: This method is used for initializing the blocks class. It contains the block type, initial position, health, etc.
- **Methods:**
  - **draw_block**: This method is used for drawing the block on the screen. It consists of block image, position, health, etc. It also used for showing special effects on the block.
  - **draw_health_bar**: This method is used for drawing the health bar of the block. It contains the block image, position, health, etc.
  - I have created many methods in this module to handle the special effects during the game in reaction with collision of character with block.
  - I have introduced a special block which will appear randomly while running the game. It has some bonus points. **SpecialBlock**
  - There are many methods I have just included the important ones; the remaining methods are for special effects.
  - **update**: This method is used for updating the block, health, etc.

- **set**: This class is used for handling player and blocks combinedly. In simple way it links blocks of player.
  - **__init__**: This method is used for initializing the set class. It contains the player blocks, slingshots etc.
  - **Methods:**
    - **add_block**: This is used to add the blocks to the corresponding player.
    - **initialise_blocks**: This method is used for loading the images for the game. It contains the image path and type of block.
    - **change_character**: This method is used for changing the characters randomly after the turn is completed.
    - There are some other methods like drawing_blocks(), is_castle_destroyed(), draw_slingshot() etc.
- **play**: This class is used for handling the game play, music and deciding winner etc.
  - **__init__**: This method is used for initializing the play class. It contains the player blocks, slingshots, game state music etc.
  - **Methods:**
    - **play_background_music**: This method is used for playing the background music. It contains the music path and volume.
    - **handle_events**: This method is used for handling the events of the game like mouse button down, mouse button up, mouse motion etc.
    - **update**: This method is used for updating the game state. It contains the game state and player blocks.
    - **draw**: This method is used for drawing the game on the screen. It contains the game state and player blocks.
    - There are some other methods like run(), draw_game_over() etc.
- There are some other classes like **calculations**, **main** etc. which are used for handling the calculations and main game startup.

4 Technical implementation

4.1 Game loop

The game loop is the main loop of the game. It runs continuously until the game is over. The game loop handles all the events and updates the game state. The game loop is implemented in the main.py file. The main loop is as follows:

```
def run(self):
    if self.game_state == "menu":
        self.play_background_music(self.menu_music)
    while self.running:
        if self.current_game_state != self.game_state:
            if self.game_state == "menu":
                self.play_background_music(self.menu_music)
            elif self.game_state == "playing":
                self.play_background_music(self.playing_music)
                pygame.mixer.music.set_volume(0.2)
                self.effect_channel.set_volume(1.0)
            elif self.game_state == "game_over":
                self.play_background_music(self.game_over_music)
            self.current_game_state = self.game_state
        self.handle_events()
        self.update()
        self.draw()
        self.dt = self.timer/ 1000
        self.timer = self.clock.tick(60)
    pygame.quit()
    sys.exit()
```

4.2 Event handling

The event handling is done in the handle_events() method. The event handling is done using the pygame library. In handle_events() method we handle the events of the game. There are different handling methods inside the handle_events() method like handling_menu_events(), handling_playing_events(), handling_game_over_events() etc. For example: handling_playing_events() method is as follows:

```
def handle_playing_events(self, event):
    """Handle events during gameplay"""
    current_player = self.players[self.current_player_idx]
    player_x = current_player.player_x

    if event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1 and not player_x.is_thrown:  # Left mouse button
            mouse_pos = pygame.Vector2(event.pos)
            player_x.start_drag(mouse_pos)
            exit_button = self.ui.draw_exit(self.screen)
            if exit_button.collidepoint(event.pos):
                self.game_state = "menu"
                self.background = self.menu_background
                self.current_player_a_blocks = player_a_blocks
                self.current_player_b_blocks = player_b_blocks

    elif event.type == pygame.MOUSEMOTION:
        if player_x.is_dragging:
            mouse_pos = pygame.Vector2(event.pos)
            player_x.update_drag(mouse_pos)

    elif event.type == pygame.MOUSEBUTTONUP:
        if event.button == 1 and player_x.is_dragging:
            mouse_pos = pygame.Vector2(event.pos)
            player_x.end_drag(mouse_pos)
```

Above code is used to handle the events of the game like mouse button down, mouse button up, mouse motion etc, while playing the game and also handling the events of the game like exit button. There are different files for handling all the events of the characters and blocks.

4.3 Game state management

The game state management is done using the game_state variable. The game_state variable is used to manage the different states of the game. The different states of the game are:

- menu: The main menu of the game
- playing: The game is being played
- game_over: The game is over
- next_level: The next level of the game

5 Basic features

5.1 Game interface

Developed a UI with a main menu and enter the players names before starting the main game. The start of the game will take you to the gameplay screen. Some music was added to it to get the feel of Demon Slayer anime.

5.2 2-Player Gameplay

Players launch projectiles competing to destroy structures. Turn-based gameplay. The deployable characters will be generated randomly based on selection of characters.

5.3 Projectile mechanics

Implemented a gravity-based launching system where players use mouse input (drag and pull) to control the angle and force.

5.4 Game Over Conditions

Define winning and losing conditions based on destroying the opponent structure/fort to an extent/entirely.

5.5 Projectile effects

Each character has unique abilities towards the different blocks.

```
"zenitsu": {
    "image": "images/zenitsu!.png",
    "damage_multiplier": {"wood": 1.0, "ice": 0.8, "stone": 0.8},
     "song" : "music/zenitsu_bgm.mp3"
},
"tanjiro": {
    "image": "images/tanjiro!.png",
    "damage_multiplier": {"wood": 0.8, "ice": 1.0, "stone": 0.8},
    "song" :"music/tanjiro_bgm.mp3"
},
"inosuke": {
    "image": "images/inosuke.png",
    "damage_multiplier": {"wood": 0.8, "ice": 0.8, "stone": 1.0},
    "song" :"music/inosuke_bgm.mp3"
},
"rengoku": {
    "image": "images/rengoku!.png",
    "damage_multiplier": {"wood": 1, "ice": 1, "stone": 1},
    "song" : "music/rengoku_bgm.mp3"
}
```

6 Advanced features

6.1 Sound effects

Added sound effects: background music for different states, character-specific sounds for launching projectiles, and block special block hit sounds.

6.2 Game Level

The game has two levels. The first level is the main game and the second level has more blocks and a different background.

6.3 Custom Theme for the game

The game has a custom theme based on the anime ”Demon Slayer”. The characters and blocks are designed based on the anime characters. The game has a custom font and background music based on the anime.

6.4 Dynamic Terrain

Introduced elements like wind and moving obstacles for the bonus points.

6.5 Scoring System

Assigned scores based on the number of throws required to destroy the blocks. It defines the art of attacking multiple blocks at time and gives bonus points for doing so within some throws.

6.6 Destructable elements

As the blocks are here the main villains in the anime, I didn't want to change their expressions or so and I didn't find some solid images for blocks to change when hit by characters. So I thought to draw a health bar for the blocks to show the destruction of blocks (villains).

6.7 Special effects

I have added some special effects to the blocks and characters. The special effects are based on the anime characters.

- For Rengoku I have added fire effects; mixture of red and yellow colored circles of different sizes will be generated.
- For Tanjiro I have added a crimson flame effect; mixture of red colored circles of different sizes will be generated.
- For Zenitsu I have added a thunder effect; mixture of blue and white circles and lines of different sizes will be generated.

- For Inosuke I have added a wild boar effect; mixture of brown colored circles of different sizes will be generated.
- For depicting various effects I have used the health bar, here the health bar won't change suddenly, but it will decrease gradually.

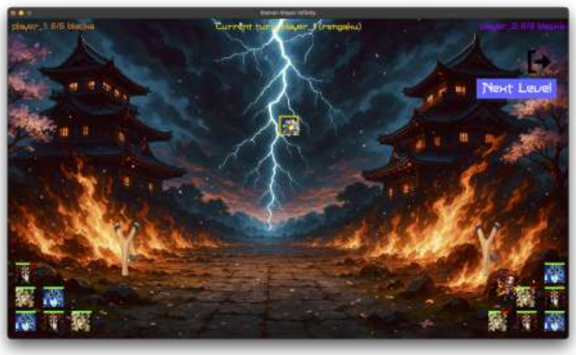

Figure 7: special effects

Learnings

- I have learned a lot of things from this project. I have learned how to use the Pygame module and how to handle the events in the game.
- I have learned how to create a game with sound effects and music.
- I have learned how to create a game with special effects and animations.
- The main thing I learned is how to handle a large codebase, organize the code in different files and modules, and be patient while debugging errors.
- our brain is better than the AI for some cases, like debugging the code and handling the errors.

7 Project journey

I choose this project beacuse, I used to like the demon slayer anime and I wanted to create a game based on it. I started learning the Pygame module and I found it interesting. I started creating the game and I faced many challenges in the beginning. I started with the basic game and then I added the features one by one. I faced many challenges in the beginning like handling the events, creating the game loop, etc. But I learned a lot from these challenges and I enjoyed the process of learning.

- I started with the basic version of the game which consists of only single player and no special effects ,and with out any interface or organisation of code.
- First I learnt ,hoe to control the mouse events to drag and leave the characters. from pygame documentation.[3]
- I referred the code organisation and structure from the git hub refeference.[1]
- I started with the basic version of the game which consists of only single player and no special effects ,and with out any interface or organisation of code.
- I have used git hub copilot auto fill feature to autofill the code and I have used the chapgpt to verify my errors,
- I was bad at handling corner cases and I used to get errors like index out of range, list index out of range, etc. I used to get frustrated with these errors and I used to spend a lot of time debugging the code.
- I have used stackoverflow to get the solutions and examples for the functions and methods I used in the code.[2]
- Stackoverflow is a great platform to get the solutions for the errors and I have used it a lot in this project.
- I struggled a lot for finding images for the characters and blocks. I have used the images from internet and some of them are ai generated.
- I learnt insertion of music from stackoverflow and I got some issues with the background music and character music , I took help of chatgpt to resolve this problem.It solved it using channels I didn't understand fully but I used it directly.
- I took help of chatgpt to resolve the issues with the special effects , and with colour codes.
- I took help help of chatgpt to create draw trajectory lighting effect
- In this project, I learned a lot of things about pygame. Surfaces are the main things that I took time on. At that, I was very slow but over time I got used to it. This project made me increase my patience. I got to know some meaning about debugging in the process. But creating a game was a very nice experience for me.
- **Finally I have created a game which is based on the anime Demon Slayer and I am happy with the project.**

References

[1] Estevao Fon. *GitHub Repository: Angry Birds Python.* Accessed: 2025-04-28. URL: https://github.com/estevaofon/angry-birds-python.

[2] Stack Overflow. *Stack Overflow Functions.* Accessed: 2025-04-28. URL: https://stackoverflow.com/questions.

[3] pygame-ce. *pygame-ce documentation.* Accessed: 2025-04-28. URL: https://www.pygame.org/docs/ref/mouse.html.