Mystic Mayhem

1v1 Angry Birds on Pygame

Project Report

Saksham Khandelwal
24B0965
CS108: SSL

April 29, 2025

Contents

1 Abstract

This project presents a turn based 1v1 game inspired by Angry Birds having a twist of wizard and spells, developed using Python and the Pygame library. Players take alternate turns to launch explodable projectiles at each other's towers which are subjected to gravity and drag, having a collision effect with the tower blocks. The objective is to strategically and precisely throw the spell orbs to destroy the opponent's tower. The game includes different pages for different actions, such as the player selection screen, main game world, leader board, and so on, and contains animations and background music to enhance the gaming experience. This project demonstrates the use of Python and its powerful game development library Pygame with integration of game design and physics implementations to offer engaging and competitive game play.

2 Introduction

The motivation behind this project came from my interest in exploring game development. While working on this, I wanted to create a unique gaming experience that blends strategy and fun, something that could be engaging and visually appealing. I decided to go with somewhat different theme of the game, mechanics inspired by *Angry Birds*, but with a twist. I replaced the traditional slingshot and birds with wizards and magic orbs. This added a layer of creativity, allowing me to experiment with both visual style and gameplay mechanics.

The game is set in a mystical wizarding realm where arcane magic and elemental forces dictate the outcome of fierce one on one duels. Each player assumes the role of a powerful wizard standing atop a customized tower composed of enchanted blocks. These towers serve not only as visual symbols of each player's domain, but also as destructible structures that determine the victor. The goal is to demolish the opponent's tower by launching spell orbs that behave under simulated physics including gravity, drag, collision response, and optional mid air explosion.

The core mechanic is turn based: players alternate launching their orbs by clicking and dragging in the desired direction. Strategy is vital, as the player must take into account angle, force, and timing. The spell system currently includes basic orb types but is designed for extensibility with more complex spells and effects in future iterations.

Overall, this project serves as an exploration of applying fundamental programming and design concepts such as event handling, physics simulation, asset management, and user interface design into the construction of a cohesive and enjoyable 2D game using Python and Pygame.

3 Project Architecture

3.1 Tools and Libraries

The module used are:

- `pygame-ce` - It is the community maintained fork of the original Pygame library, offering features for building games and multimedia applications.
- `os` - It lets Python interact with the operating system, including file paths and was used to load assets with correct path handling.
- `random` - It provides functions to generate random numbers and choices. It was used to create randomized ball types, tower structures, or gameplay variations to increase replayability.
- `math` - This module provides basic mathematical functions and constants. It was mostly used for trigonometric calculations and motion physics.
- `numpy` - It is a powerful library for numerical computing and array operations. It was used for vector based physics calculations and handling numerical arrays efficiently.

3.2 Directory Structure

The project directory is as follows:

```
.
├── states
│   ├── credits.py
│   ├── end_game.py
│   ├── game_world.py
│   ├── leaderboard.py
│   ├── loading.py
│   ├── main.py
│   ├── options.py
│   ├── pause_menu.py
│   ├── select.py
│   ├── state.py
│   └── tutorial.py
├── assets
│   ├── fonts
│   ├── sprites
│   └── sound
├── game.py
├── leaderboard.txt
└── options.txt
```

- **game.py** - The main game loop loading assets and calling last state update and render functions.
- **states** - Contains different python files for different states
- **assets** - Contains all the images, sounds, and fonts used in the game.
- **leaderboard.txt** - Contains the wins and losses of each player along with their ranking
- **options.txt** - Contains the audio and resolution settings to apply them at start of game.

4 Game Instructions and Gameplay

4.1 Instructions to Run the Game

Make sure python is installed on your system along with all modules mentioned above.

Also ensure that `pygame-ce` and only `pygame-ce` are only installed, not the traditional pygame. If you have the traditional pygame installed, run:

```
pip uninstall pygame
pip install pygame-ce
```

The game can be launched by using the suitable command of the following two in terminal opened in the directory containing game.py:

```
python game.py
python3 game.py
```

4.2 Gameplay

4.2.1 Loading and Intro Screen

The game starts with the loading screen during which all assets used during the game are loaded at once and then intro screen is showed from which you can proceed by pressing the Enter key

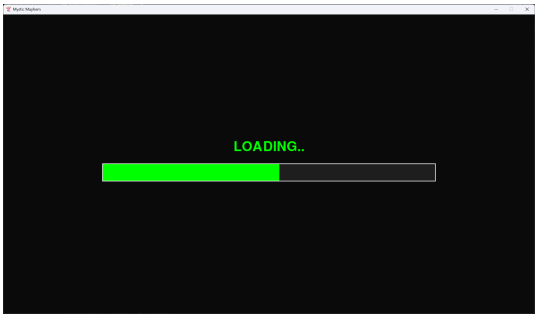


Figure 1: Loading Screen

Figure 2: Intro Screen

4.2.2 Main Menu

After Loading, you will be greeted with a Main Menu, from which you can choose to:

- Start Game
- Options to change Volume of music and sound effects, change resolution of the game window, or to view the tutorial
- Credits
- Leaderboard to view ranking of all players

You can select any of these by pressing on these buttons by mouse or using arrow keys and pressing Enter key to press. To exit, Press the Esc key or close the pygame window

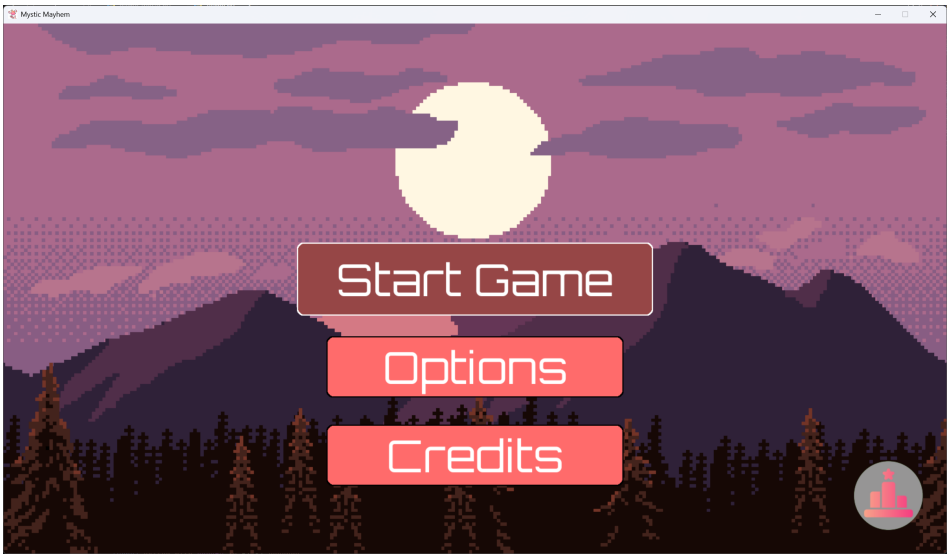


Figure 3: Main Menu

4.2.3 Tower Selection

Clicking on Start Game will lead you to the name input and tower selection page where you will be required to type your name and select your desired tower structure, you can also select random tower in which the game would randomly choose a tower for you. One more thing to add is you can also leave the name empty, the game would select one of the default names in such a case.To confirm the selection press ready.

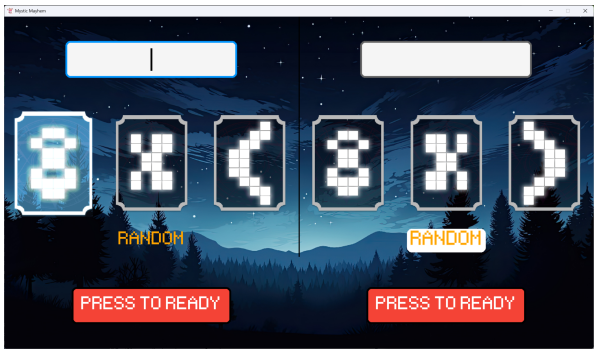
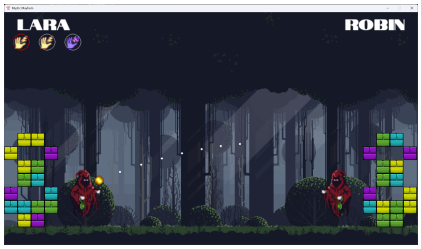


Figure 4: Select Screen

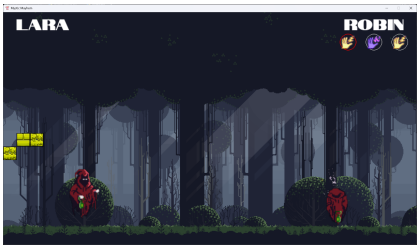
4.2.4 Game World

Once both players are ready, the game starts after a 3 second countdown, leading to the main game world, a 1v1, turn based battle inspired by Angry Birds, set in a wizard themed fantasy world. Players take turns selecting a spell and launching a magical orb at the opponent's tower by dragging in the desired direction. Upon release, the orb is affected by gravity and air drag. If you begin dragging but decide not to throw, you can reposition the orb to the center and release the mouse to cancel the action.
When the orb hits a block in the opponent's tower, it deals damage to that block and the adjacent blocks in the four cardinal directions. You can also trigger the orb's explosion manually in mid air by right clicking or pressing the "O" key, causing area damage to nearby blocks.
The goal is to strategically demolish all the blocks in the opponent's tower to win the match.

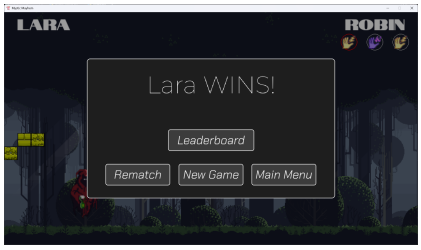
(a) Start



(b) Launch



(c) Death



(d) End Screen

Figure 5: Screenshots from different stages of the game world

4.2.5 Other Screens

The game includes several additional screens and features that enhance the overall experience:

- A pause menu can be accessed by pressing the `Esc` key, offering options to resume the game, return to the main menu, or adjust settings.
- The leaderboard displays detailed player statistics, including rankings, win/loss records, and the most frequently selected and best performing tower for each player.
- Various sound effects and smooth animations are integrated throughout the game to create an engaging and visually appealing experience.
- A tutorial section is available in the options menu, providing clear explanations of game mechanics and instructions on how to play.

5 Key Implementation Details

5.1 Asset Loading

All the assets including images with their desired dimensions, fonts with their desired size, and music are loaded all at once when the game start by defining list of assets to add and running it. For example:

Listing 1: Image Loading

```
for img_file, alias, dim in self.image_files:
    path = os.path.join(self.sprite_directory, img_file)
    self.assets[alias] = LoadScaledImage(path, scaling_dim=(int(dim[0] * self.
        scale_factor), int(dim[1] * self.scale_factor)))
    progress += 1
    self.draw_loading_bar(progress, total_assets)
```

5.2 Game Loop

Listing 2: State Management

```
def game_loop(self):
    while self.playing:
        if self.state_stack[-1].state != "Select":
            self.get_events()
        self.update()
        self.render()
        self.clock.tick(self.FPS)

def update(self):
    self.state_stack[-1].update(self.actions)

def render(self):
    self.state_stack[-1].render(self.screen)
    pygame.display.flip()
```

5.3 Physics and Collision

5.3.1 Ball Motion

Basic gravity and drag implementation and checking for corner cases - hitting game window boundary

Listing 3: Ball Motion Physics

```
self.velocity += gravity
self.position += self.velocity
self.velocity *= (1 - air_drag)

# Boundary Cases
```

5.3.2 Block Damaging

- Case 1: Explosion of ball
  Distance between centre of ball and centre of block is calculated and damage is done based on that distance range.
- Case 2: Collision of ball
  Collision is detected using spritecollide function and closest block(as sprite is square png) is said to be hit and damage is done on this block and its adjacent blocks based on ball type and block type

  No complex physics is added, i.e., the block only gets damaged and the ball is killed on collision and blocks are also not under gravity influence.

5.4 Leaderboard Management

After the end of match, the information of that match is passed on to recalculate the leaderboard that contains all the info (player name, wins, losses, tower stats) and updating the information of players that played currently new data like ranking, best tower,etc. is calculated and saved in leaderboard.txt

6 Challenges and Solutions

6.1 Turn Based System Implementation

Initially, managing turn transitions between the two players posed several challenges. Inputs intended for one player were sometimes processed during the other player's turn, leading to overlapping actions. To resolve this, the codebase was restructured into modular functions, input states were explicitly reset between turns, and strict condition checks were added to ensure that inputs were only accepted during the appropriate player's turn. This resulted in a more reliable and bug free turn based flow.

6.2 Reset Animation and Audio Synchronization

To enhance immersion, I wanted the orb spawning animation to appear as if the wizard was casting a spell, with synchronized visual and sound effects. However, other concurrent animations such as the introductory "Fight" text rendering conflicted with handling this within the `reset_spawn_animation` function. The solution was to trigger the corresponding sound effect within the `Player.update()` method when a specific animation frame (frame 1) was reached, ensuring correct timing. Additionally, maintaining background music continuity when navigating to and from the pause menu was another challenge. The `pygame.mixer.music.get_pos()` function only provides the time elapsed since the last `play()` or `unpause()` call, making it unsuitable for tracking actual playback position. This was addressed by manually tracking music positions using custom variables and synchronizing them on pause/resume.

6.3 Resolution Scaling Implementation

The game was originally developed at a fixed resolution of 960×540. Later, I wanted to add support for resolution switching without degrading visual quality. Instead of scaling the final display surface which could cause pixelation I opted to implement internal resolution scaling by applying a `scale_factor` to all rendered elements. This meant updating every dimension and coordinate calculation throughout the code to account for this factor. While not the most efficient approach, it preserved sprite clarity across different screen sizes and resolutions.

6.4 Adaptive Input Preference Buttons

I aimed to design a flexible button input system that dynamically adapts to the player's most recent input method either keyboard or mouse. The button action needed to called correctly based on the input method. Several bugs arose while trying to manage conflicting inputs and maintain smooth transitions. Eventually, the system was made reliable by using input flags to track the last input event and decoding actions based on the current active preference.

7 Customizations

A list of all the special customization implemented in the game:

- Custom Theme for the Game.
- Destructible Block Sprites
- Parallax Background of the Main Menu.
- Dynamic Elements and Animations throughout the Game.
- Music and Sound Effects.
- Leaderboard.
- Resolution and Volume Options.
- Pause Menu.
- Customized Fonts.
- Responsive Buttons.

8 Conclusion and Future Work

Overall, the project laid a solid foundation for a fun and functional 2D game and presents many exciting directions for continued development and refinement.
While the core gameplay and systems function as intended, there are several areas for potential improvement and future development:

- **Enhanced Block Physics:** Introduce dynamic physics based movement and collision response for blocks, such as falling due to gravity or bouncing slightly upon impact. Rotational physics could also be added to increase visual realism. (An early attempt at this feature was made but later removed due to instability.)
- **Expanded Spell System and Power Ups:** Introduce a broader range of spells and abilities, each with distinct visual effects, strategic use cases, and cooldown timers, to diversify gameplay and allow for more tactical choices.
- **Procedural or Custom Tower Generation:** Replace static, hardcoded tower layouts with procedurally generated structures or offer players the ability to build and save their own towers, improving replayability and personalization.
- **User Interface and User Experience Enhancements:** Improve the visual design of menus and in game interfaces, including smoother transitions, animations, and responsive feedback for player actions to make the game feel more polished.
- **Code Optimization and Scalability:** Refactor core systems, particularly the scaling and rendering logic, to improve performance across different resolutions and devices while making the codebase easier to maintain and extend.
- **Single Player AI Mode:** Implement an AI controlled player to allow solo gameplay. The AI could vary in difficulty and behavior to simulate different types of opponents.

- **Online Multiplayer Support:** Extend the game to support networked multiplayer so players can compete remotely, possibly with matchmaking and custom lobbies.
- **Replay System and Highlights:** Add functionality to replay matches or highlight key events such as impressive shots or game winning moves.

9 References

[1] Pygame-ce Documentation. https://pyga.me/docs/
Referred throughout development for understanding surfaces, events, and syntax help.

[2] Pygame Tutorial. https://youtu.be/AY9MnQ4x3zk?si=2S1b_BqR-7Euu1wS
Used as a foundation for initializing the game, handling input events, and drawing objects on the screen.

[3] Game States Management Tutorial. https://youtu.be/b_DkQrJxpck?si=Cc5vK_oyTMJd-fHB
Used as a base for creating separate game states like loading screen, menu, pause menu, game world, etc.

[4] Itch Game Assets. https://itch.io/game-assets
Sprites used for characters and backgrounds were sourced from the free game assets section on Itch.io.

[5] Google Fonts. https://fonts.google.com
Fonts for the game interface were obtained from the wide selection of free to use fonts on Google Fonts.

[6] Pixabay. https://pixabay.com
Sound effects for the game were sourced from Pixabay, which offers a variety of free to use sound assets.

Additional Tools Used

ChatGPT *Assisted with providing some new ideas, troubleshooting code and offering suggestions during development.*

VSCode Autocomplete *Used for efficient code completion, error handling, and speeding up the development process with code suggestions.*

GIMP *Used for designing and editing the game's custom assets such as backgrounds and other visual elements.*

Audacity *Used to edit custom sound effects for the game, providing high quality audio output.*

Aseprite *Used for creating and editing pixel art assets, such as character sprites and game tiles.*