Angry Birds - 2 Player: A Pygame Challenge

Rishi Vardhan Majji
24B0969

April 29, 2025

Contents

1 Introduction

This report details the development of a 2-player, turn-based game inspired by Angry Birds, created using the Pygame library. The objective is for players to use a variety of bird projectiles launched from slingshots to destroy their opponent's fortress structure. The game incorporates physics simulation for projectile motion and distinct properties for different block and bird types.

2 Modules Used

The following external Python libraries were used in this project:

- **pygame-ce (pygame Community Edition):** The core library used for game development, handling graphics rendering, event management (keyboard, mouse), sound playback, sprite management, and window creation.
- **numpy:** Used for numerical operations, particularly array manipulation (e.g., in Game/Game.py for block positioning logic using np.zeros, np.array, np.random.choice).
- **random:** Utilized for generating random elements, such as initial bird placement (Game/birds.py), block type selection, and block positioning (Game/Game.py).
- **math:** Employed for mathematical calculations, including trigonometry (sine, cosine, atan2) for projectile angle and velocity calculations (Game/slingshot.py, Game/projectile.py) and potentially for wind effects (Game/Game.py).
- **sys:** Used for system-specific parameters and functions, primarily for exiting the game cleanly (pygame.quit(), sys.exit()) in main.py and Screen_mode/game_setup.py.
- **os (specifically os.path):** Used for path manipulation, primarily os.path.join to create platform-independent paths for loading assets (images, sounds) in my_utilities.py, Game/slingshot.py, Game/projectile.py, etc.

3 Directory Structure

The project is organized into the following directories and files:

```
.
|-- main.py             # Main script to run the game (entry point
|-- Game/               # Core gameplay logic module/package
|   |-- birds.py        # Defines the Bird class
|   |-- blocks.py       # Defines the Block class
|   |-- projectile.py   # Defines the Projectile class
|   |-- slingshot.py    # Defines the Slingshot class
|   |-- Game.py         # Contains the main RunGame function and helpers
|
|-- Screen_mode/        # UI Screens logic module/package
|   |-- start_game.py   # Handles the initial start screen [cite: 73]
|   |-- game_setup.py   # Handles the player name/settings screen [cite: 55]
|
|-- Assets/             # Folder for all game assets
|   |-- Backgrounds/    # Background images, floor, slingshot images
|   |-- Birds/          # Bird animation/state images (.webp)
|   |-- Blocks/         # Block type images (normal, damaged) (.webp, .png)
|   |-- Fonts/          # Font files (e.g., angrybirds-regular.ttf)
|   |-- Logos/          # Game icon/logo images
|   |-- Music/          # Sound effects (.mp3) and background music
|
|-- my_utilities.py     # Utility functions (e.g., image loading, text writing)
|-- values.py           # Stores constant values (screen size, physics factors, etc.)
|-- report.pdf          # This compiled report file
```

- / (Root Directory)
  - `main.py`: The main entry point for the game. Initializes Pygame, sets up the screen, and starts the game flow by calling start_game.
  - `my_utilities.py`: Contains helper functions used across different modules, such as loading and scaling images (background_load, block_load, bird_load), rendering text (text_writer), and setting up the floor graphic (set_floor). Also defines shared fonts.
  - `values.py`: Stores global constants and configuration values used throughout the game, including screen dimensions, floor level, physics constants (gravity, collision factors), asset sizes, color definitions, and game settings options (difficulty, wind).
  - **Assets/** (Directory)
    - **Backgrounds/**: Contains background images, floor textures, and slingshot images.
    - **Blocks/**: Contains images for different block types (wood, stone, glass, hollow variants) and their damaged states.
    - **Birds/**: Contains images for different bird types (Red, Blue, Chuck, Bomb) in various states (default, angry, air, corpse, etc.).
    - **Logos/**: Contains game logos or icons.
    - **Music/**: Contains sound effect files (.mp3) for birds, blocks, slingshot actions, etc.
    - **Fonts/**: (Implied by font loading) Contains font files like angrybirds-regular.ttf.
  - **Game/** (Directory)
    - `birds.py`: Defines the Bird class, handling bird initialization, state management (idle, loaded, pull, air, happy), animations, gravity effects, and interaction logic (getting loaded onto the slingshot).
    - `blocks.py`: Defines the Block class, managing block properties (type, health, mass), image loading (normal, damaged), sound effects, physics (velocity, gravity, friction), collision handling with other blocks and boundaries, and damage application.
    - `projectile.py`: Defines the Projectile class, handling the physics of launched birds (velocity, gravity, wind), collision detection with blocks, damage calculation, state changes (flying to corpse), sound playback, and boundary interactions.
    - `slingshot.py`: Defines the Slingshot class, managing the slingshot state (idle, loaded, pulling), loading birds, handling mouse input for pulling and releasing, calculating launch trajectory (angle, momentum), drawing the slingshot and elastic bands, and playing associated sounds.
    - `Game.py`: Contains the main game logic function RunGame. Initializes game elements (slingshots, birds, fortresses), manages game state (aim, projectile, turnover, gameover), handles player turns, processes game events, updates and draws all game objects, calculates scores, implements game settings (difficulty, wind), and checks for win/loss conditions.
      Also contains functions that initialise fortresses and birds
  - **Screen_mode/** (Directory)
    - `start_game.py`: Manages the initial start screen, displaying the background and a "START" button. Transitions to the game setup screen upon clicking START.
    - `game_setup.py`: Defines the get_game_data function, which presents the game setup UI. Allows players to enter their names and select game settings (Difficulty, Wind Speed, Direction, Type). Handles input validation and returns the chosen settings to start the game.

4 Running Instructions

4.1 Prerequisites

Ensure you have Python installed on your system. You also need the following libraries:

- Pygame Community Edition (pygame-ce)

- NumPy (numpy)

You can typically install them using pip:

```
pip install pygame-ce numpy
```

4.2 Execution

1. Navigate to the root directory of the project in your terminal or command prompt.// 2. Run the main script using Python:

```
python main.py
```

4.3 Controls

- **Start Screen:** Click the "START" button to proceed to the game setup.
- **Game Setup Screen:**
  - Click inside the input boxes to type player names. Press Enter or Tab to switch between name boxes.
  - Click on the desired option buttons for each game setting (Difficulty, Wind, etc.).
  - Click the "START GAME" button once all settings are selected to begin the match.
- **Gameplay:**
  - **Aiming:** Click and hold the left mouse button on an available bird near your slingshot to load it. Drag the mouse away from the slingshot anchor point to aim. The pull distance affects the launch force.
  - **Launching:** Release the left mouse button to launch the bird.
  - **Turns:** The game automatically switches turns after a projectile comes to rest or goes off-screen. The active player's name is typically highlighted.
  - **Score:** The scores are displayed below the respective player's name
  - **Time:**The time left for the players to play is shown above
- **Exiting:** Close the game window or potentially press the ESC key during gameplay.

Note that Red does equal damage to all the blocks (0.75 times it's momentum) while Blue does more damage to glass blocks(0.9 times it's momentum) and less to others(0.6 times it's momentum) and the same for Chuck(does more damage to wooden blocks) and Bomb(does more damage to stone blocks)

5 Implemented Features

5.1 Basic Features

- **Game Interface:**
  - A start screen (start_game.py).
  - A game setup screen (game_setup.py) allowing players to enter names and choose game parameters.
  - The main gameplay screen displaying fortresses, slingshots, birds, scores, and game status.
- **2-Player Gameplay:**
  - Turn-based system where players alternate launching birds (Game.py).
  - Each player controls their own slingshot (left and right sides of the screen) (slingshot.py).

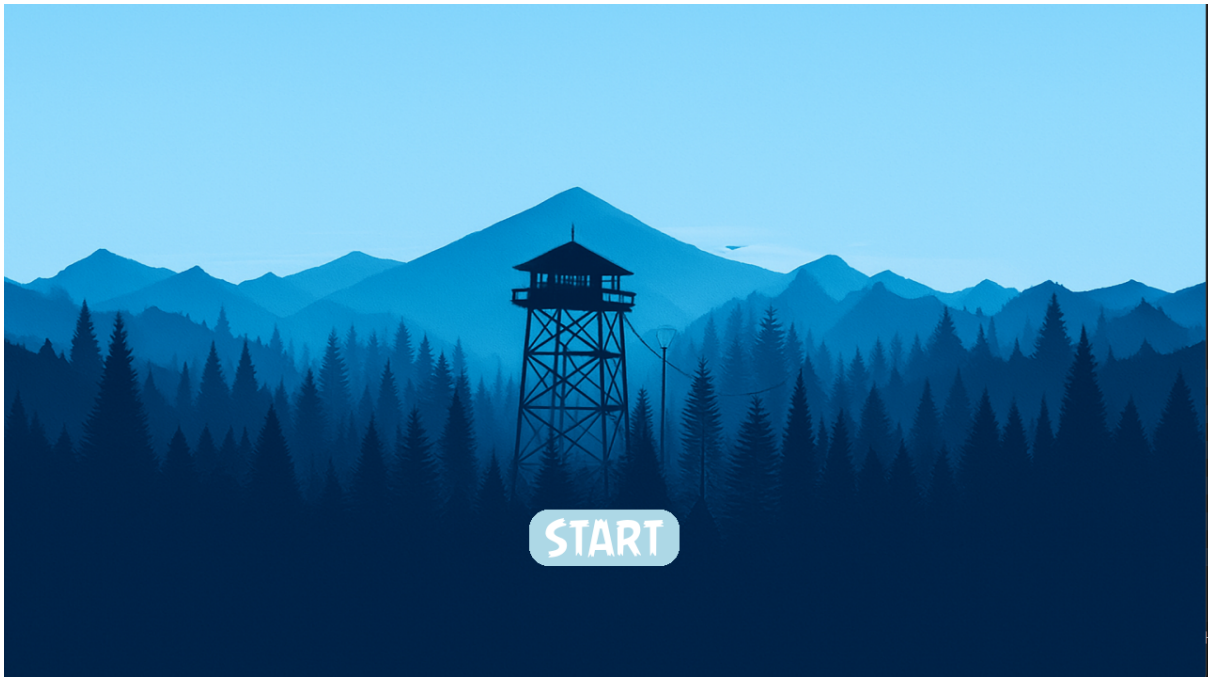

Figure 1: Intro Screen

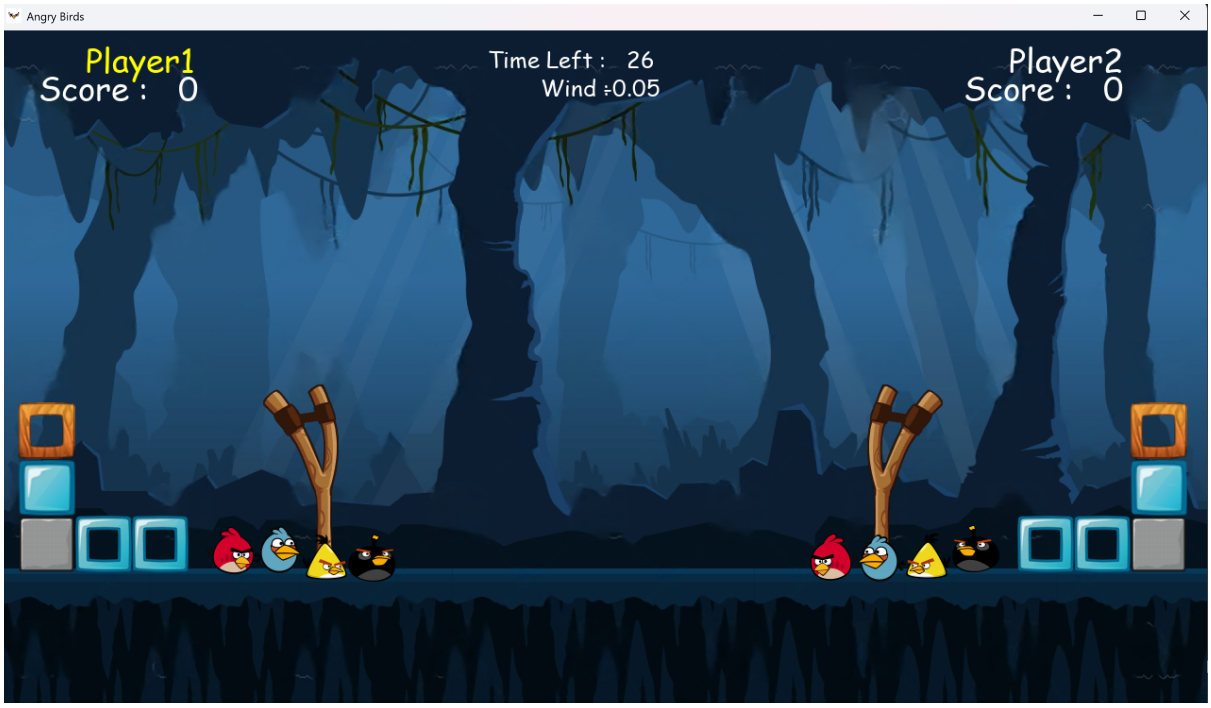

Figure 2: Game Screen

  - Objective: Destroy the opponent's fortress structure as much as possible in given time(Game.py, blocks.py).

- **Projectile Mechanics:**
  - Gravity simulation affecting bird trajectory (projectile.py, values.py).
  - Mouse-based drag-and-pull system for controlling launch angle and force (slingshot.py).
  - Calculation of initial velocity based on pull vector (slingshot.py).
  - Trajectory prediction dots shown during aiming (slingshot.py).

Figure 3: While Pulling

- **Fortress Structures:**
  - Fortresses composed of multiple blocks (blocks.py).
  - Different block types implemented (Wood, Stone, Glass, and their Hollow variants) with distinct properties (health, mass) defined in blocks.py and values.py.
  - Blocks have health and can be damaged or destroyed upon impact (blocks.py).
  - Basic block physics (gravity, friction, block-on-block collisions) implemented (blocks.py).
- **Projectile Types (Birds):**
  - Multiple bird types implemented: Red, Blue, Chuck, Bomb (birds.py, projectile.py).
  - Birds have unique appearances and potentially different effectiveness against block types (damage logic in projectile.py::damage_dealt).
  - Birds transition to a 'corpse' state after significant collision or timeout (projectile.py).
- **Bird Generation:**
  - Replenishing batch-wise: Each player starts with one of each bird type. Once a bird is used, a new one of the same type respawns for that player (Game.py::release_bird, Game.py::bird_init).

Figure 4: Birds

- **Game Over Conditions:**
  - Game ends when one player's fortress is completely destroyed (Game.py).
  - Game ends after a time limit (e.g., 200 seconds), with the winner determined by score (Game.py).
  - Draw condition implemented if both fortresses are destroyed simultaneously or scores are equal at timeout (Game.py).
  - Game over screen displays the winner or draw status and offers options to "Play Again" or start a "New Game" (Game.py).
- **Scoring System:**
  - Points awarded based on damage dealt to opponent's blocks (Game.py within projectile collision handling).
  - Score displayed for each player during the game (Game.py).
- **Sound Effects:**
  - Sounds implemented for various actions: bird selection, slingshot pull/release, projectile flight, block hits/breaks, bird corpse (birds.py, blocks.py, projectile.py, slingshot.py).

5.2 Advanced Features / Customizations

- **Game Difficulty Levels:**
  - Options for "Easy", "Medium", "Hard" difficulty available in the setup screen (game_setup.py).
  - Difficulty affects the number of blocks in the fortresses (Game.py::data).
- **Wind Simulation:**
  - Options for "Low", "Medium", "High" wind speed (game_setup.py).
  - Options for wind direction ("Opposite to Player", "In Direction of Player") (game_setup.py).
  - Options for wind type ("Constant", "Variable" - sinusoidal variation over time) (game_setup.py).
  - Wind force affects projectile trajectory (Game.py::data, applied in projectile.py).
  - Current wind value displayed on screen (Game.py).
- **Block Damage Visualization:**
  - Blocks change appearance when damaged (e.g., switch to a 'damaged' image when health drops below 50%) (blocks.py).
- **Enhanced Physics/Interactions:**
  - Blocks exhibit basic physics including falling under gravity, friction, and restitution (bouncing) on collisions with the floor and other blocks (blocks.py).
  - Projectiles bounce off walls and the floor (projectile.py).
  - Collision physics implemented between projectiles and blocks, transferring momentum (projectile.py::collide)
  - Amount of damage done depends on momentum of bird/projectile
  - Interactive response on hovering over birds while selecting
  - Bird expressions based on their state to give an interactive feel
- **Improved UI/UX:**
  - Visual feedback during slingshot pull (elastic bands drawn) (slingshot.py).
  - Trajectory prediction dots (slingshot.py).
  - Blinking cursor and visual focus indication in input fields (game_setup.py).
  - Hover effects on buttons (game_setup.py).

6 Project Journey

6.1 Key Learnings

- Pygame Concepts
- Code modularization
- Working with assets

6.2 Challenges Faced

- Implementing realistic collisions
- Debugging collision issues
- Heavy UI
- Low frame rates

Figure 5: On Hover

6.3 Solutions and Approaches

- Took help of AI to get rid of collisions that get into loops(vibrate)
- Used print statements to debug
- Took help of AI to know where my code is not doing good(found that using collide_mask on everything drastically reduces frame rate

7 Bibliography / References

1. **Pygame Community Edition Documentation:** https://pyga.me/docs/
2. **GeeksforGeeks:** https://www.geeksforgeeks.org/pygame-tutorial/
3. **ChatGPT(used for small clarifications and to find better ways to do few things):** https://chatgpt.com/
4. **NumPy Documentation:** https://numpy.org/doc/stable/
5. **Youtube:** https://youtu.be/AY9MnQ4x3zk?si=xQ7_fe1emX-UUbib
6. Images from: https://angrybirds.fandom.com/wiki/Angry_Birds_Wiki/gallery