Angry Birds 1v1: Python-Based Game Simulation

Final Technical Report

Introduction

This project is a two-player competitive game inspired by Angry Birds, developed using Python and Pygame CE. The game features physics-based projectile motion, destructible structures, real-time score tracking, and dynamic visual feedback.

Game Mechanics

- **Block Health Initialization.** Each wooden, stone, or ice block is instantiated with a health value of 100 points.
- **Damage & Scoring.** Player score increases by the exact amount of damage dealt to opponent blocks. All damage values are cumulative.
- **Turn Structure.** Players alternate launches in a strict, turn-based sequence. Each turn allows one bird launch.
- **Victory Condition.** When a player's total accrued damage reaches or exceeds 1000 points, the match terminates. The background music is halted and the victory soundtrack is triggered.

Key Features

- **Bird Types and Abilities** – Four unique birds with click-activated powers.
- **Block Mechanics and Damage Handling** – Material-sensitive health and damage model.
- **Score Tracking and Win Condition** – Points equal damage dealt; 1000 points to win.
- **User Interface and Navigation** – Main menu, instructions, pause, and victory screens.
- **Procedural Tower Generation** – Level-dependent block layouts.
- **Physics and Game Loop** – Custom gravity, collision, and real-time frame rate display.

Dynamic Block Textures

Each block swaps its source PNG to a damaged variant once its health falls below 50%.

Figure 1: Normal vs. damaged wood, stone, and ice textures.

- When a block's health drops below 50%, its image switches from `wood.png`, `stone.png`, or `ice.png` to the corresponding `damaged_*`.png variant (Fig. 1).

Audio Feedback

- **Background Music:**
  - `angry-birds.ogg` loops continuously during play.
  - On victory, the music stops and `end.mp3` plays once.
- **Launch and Ability Sounds:**
  - `launch.mp3` plays on every bird release (`launch_sound.play()`).
  - Splitting, boosting, and detonating reuse `launch_sound` for consistency.

Class Overview

Bird

Handles projectile motion, special abilities, rendering, and collision.

Figure 2: Sprite variants for Red, Blue, Yellow and Black birds.

```
class Bird:
    def __init__(...): ...
    def handle_event(self, event): # click-to-boost, split, detonate
    def update(self):              # apply gravity, move, bounce
    def draw(self):                # render bird/explosion
```

Listing 1: Key Methods in `Bird`

Block

Manages health, texture swapping, damage logic and drawing.

```
class Block:
    def __init__(...): ...
    def update_image(self): # swap to damaged_ variant if health<50%
    def hit(self, bird):    # compute damage, reduce health, bounce
   bird
    def draw(self):         # blit image + health bar
```

Listing 2: Key Methods in `Block`

Sling

Draws the slingshot base and the elastic lines while dragging.

```
class Sling:
    def __init__(...): ...
    def draw(self):              # render sling base
    def draw_sling_line(self, bird): # draw elastic bands
```

Listing 3: Key Methods in `Sling`

Player

Initializes name, sling, randomized tower and manages bird queue and score.

```
class Player:
    def __init__(self, sling_pos, tower_pos, name): ...
    def next_bird(self): # cycle through bird_defs
```

Listing 4: Key Methods in `Player`

Game1v1

Orchestrates turns, UI states, input routing, scoring and win detection.

```
class Game1v1:
    def __init__(self): ...
    def draw_buttons(self): ...
    def draw_score_bars(self): ...
    def draw_victory_screen(self): ...
    def run(self): # main loop: events, update/draw, check score>=1000
```

Listing 5: Key Methods in `Game1v1`

Navigation

Describes menu flows and on-screen controls.

```
def show_menu(): ...
def show_instructions(): ...
```

Listing 6: `show`$_{m}$*enu*`show`$_{i}$*nstructions*

Running the Game

How to launch the application from the command line:

```
if __name__ == "__main__":
    show_menu()
    Game1v1().run()
```

Listing 7: Entry Point

Level One: 2×5 Tower

- Layout: two columns of five blocks each, offset per player side.
- Health Scoring: blocks start at 100 HP; damage dealt adds to score.
- Turns Victory: alternate bird launches until one player reaches 1000 points.

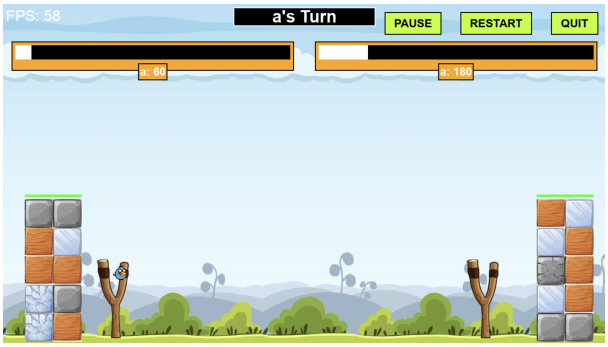

Figure 3: Level One – standard 2×5 tower configuration.

Figure 4: Level Two – centered pyramid (base width=5, tapering to apex).

Level Two: Pyramid Structure

- Layout: base row of N blocks, then N-1, ... up to one, centered.
- Procedural Generation: offsets computed per row for centering.
- Health, Scoring Victory: identical to Level One mechanics.

Core Functional Components

1. Bird Types and Abilities

```
class Bird:
    def handle_event(self, event):
        # Yellow Bird: speed boost
        if self.launched and event.type == pygame.MOUSEBUTTONDOWN and
   self.type == 'yellow' and not self.ability_used:
            self.vx *= 3; self.vy *= 3; self.ability_used = True

        # Blue Bird: split into three
        if self.launched and event.type == pygame.MOUSEBUTTONDOWN and
   self.type == 'blue' and not self.ability_used:
            self.ability_used = True
            bird_above = Bird(...); bird_below = Bird(...)
            # position and launch splits
            bird_above.launched = bird_below.launched = True
            self.split_birds = [bird_above, bird_below]
```

```
        # Black Bird: explode on click
        if self.launched and event.type == pygame.MOUSEBUTTONDOWN and
    self.type == 'black' and not self.ability_used:
            self.ability_used = True; self.exploded = True; self.
    explosion_time = time.time()
            return
```

Listing 8: Bird Ability Handling

2. Block Mechanics and Damage Handling

```
class Block:
    def hit(self, bird):
        dmg = bird.mass
        # Double damage on material advantage
        if (bird.type=='black' and self.type=='stone') \
           or (bird.type=='blue' and self.type=='ice') \
           or (bird.type=='yellow' and self.type=='wood'):
            dmg *= 2
        self.health -= min(self.health, dmg)
        # Bounce bird on impact
        if abs(dx)>abs(dy): bird.vx = -bird.vx * 0.7
        else: bird.vy = -bird.vy * 0.7
        return dmg
```

Listing 9: Block Collision and Damage

3. Score Tracking and Win Condition

```
# Inside Game1v1 game loop
if b.rect.colliderect(blk.rect):
    dmg = blk.hit(b)
    player.score += dmg
    # Check for win
    if player.score >= player.winning_score:
        self.game_over = True; self.winner = self.turn
```

Listing 10: Updating Score and Checking Victory

4. User Interface and Navigation

```
def show_menu():
    while True:
        for event in pygame.event.get():
            if event.type==pygame.MOUSEBUTTONDOWN and PLAY_BUTTON_RECT.
    collidepoint(event.pos):
                return
        surface.blit(menu_background,(0,0))
        # Draw title and buttons
        surface.blit(title, title_rect)
        surface.blit(play_button_img, PLAY_BUTTON_RECT.topleft)
        pygame.display.flip()

```

```
def show_instructions():
    back = pygame.Rect(...)
    while True:
        for event in pygame.event.get():
            if event.type==pygame.MOUSEBUTTONDOWN and back.collidepoint
    (event.pos): return
        # Render instructions overlay
        surface.blit(menu_background,(0,0))
```

Listing 11: Main Menu and Instructions Screens

5. Procedural Tower Generation

```
class Player:
    def __init__(self, sling_pos, tower_pos, name="Player"):
        bt = random.sample(['wood']*4 + ['ice']*3 + ['stone']*3, len(
    tower_pos))
        self.blocks = [
            Block(cx+dx-200, cy+dy+30, btype, block_spritesheet)
            for (dx,dy), btype in zip(tower_pos, bt)
        ]
```

Listing 12: Randomized Tower Layout

6. Physics and Game Loop

```
class Bird:
    def update(self):
        if not self.launched or self.exploded: return
        self.vy += self.gravity
        self.x += self.vx; self.y += self.vy
        # Ground collision
        if self.y + self.rect.height > HEIGHT:
            self.y = HEIGHT - self.rect.height; self.vy *= -0.3
        self.rect.topleft = (self.x, self.y)

# Main loop snippet
while True:
    for event in pygame.event.get(): handle_events()
    surface.blit(background,(0,0))
    for p in self.players: p.sling.draw(); for b in p.blocks: b.draw()
    # Launch and update birds
    player.current.update(); player.current.draw()
    pygame.display.flip()
    clock.tick(60)
```

Listing 13: Motion Update and Rendering

Resources and References

- Pygame CE Documentation
- *Angry Birds 2 - Official Gameplay Trailer* (YouTube)

- Estevão Fon: `https://github.com/estevaofon/angry-birds-python`
- Marblexu: `https://github.com/marblexu/PythonAngryBirds`
- Pygame CE vs Pygame: `https://www.youtube.com/watch?v=dj2N7dOD4DM`