2D Fighting Game Code Analysis Report

July 21, 2025

1 Overview

This report analyzes a 2D fighting game implemented in Python using the Raylib library (`raylibpy`). The game features a player-controlled character and a CPU opponent with a "Nemesis System" that adapts the CPU's strategy based on the player's actions. Key features include player controls, CPU AI, health and damage systems, visual effects, and a round-based match structure.

2 Code Structure

The code is organized into the following components:

- **Imports and Constants**: Uses `raylibpy`, `random`, `time`, and `os`. Constants define game parameters like screen size, movement speeds, and animation timings.
- **Classes**: `Player` for base character attributes and `Cpu` (inherits from `Player`) for AI-specific behavior.
- **Key Functions**:
  - `log_player_move`: Logs player actions to `player_moves.txt`.
  - `reset_move_log`: Clears the move log file.
  - `analyze_moves_on_loss`: Analyzes the last 100 moves to identify the player's dominant action.
  - `update_cpu_strategy`: Adjusts CPU behavior based on dominant actions.
  - `get_texture_for_state`: Returns textures based on character state.
  - `reset_round`: Resets states for a new round.
  - `main`: Manages the game loop, input, AI, physics, collisions, and rendering.
- **Main Game Loop**: Initializes the window, audio, and textures, processes input, updates AI, and renders the game state.

3 Key Features

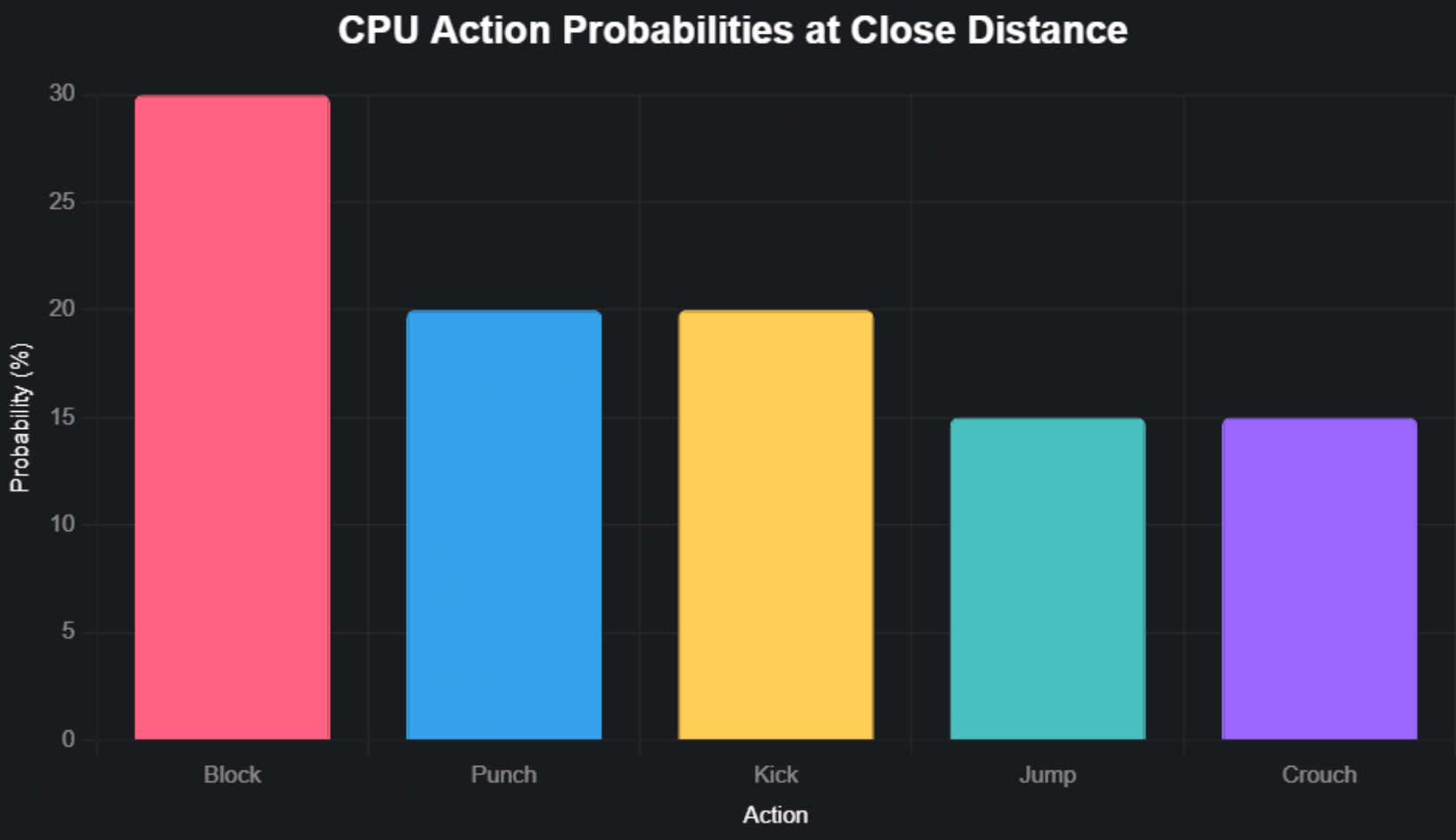
- **Player Controls**: Movement (`A`, `D`, `W`, `S`), attacks (`K` for punch, `L` for kick), and blocking (`SPACE`).
- **CPU AI**: Uses a state machine with actions (idle, move, punch, kick, block, crouch, jump) and adapts via the Nemesis System.
- **Game Mechanics**: Health bars, combo counters, screen shake, particle effects, and rectangle-based collision detection.
- **Audio and Visuals**: Background music, hit sounds, and state-based textures.
- **Nemesis System**: Analyzes player moves to adjust CPU strategy (e.g., increasing block probability for frequent punches).

4 Strengths

- Functional gameplay with core fighting mechanics.
- Innovative Nemesis System for adaptive AI.
- Visual feedback via screen shake, particles, and damage flash.
- Audio integration for immersion.
- Modular code with reusable functions.
- Clear match structure (best-of-three rounds).

5 Potential Issues

1. **Hardcoded File Paths**: Absolute paths (e.g., `D:\programs\c++\games\game_5`) reduce portability.
2. **Error Handling**: Minimal handling for file I/O and asset loading, risking crashes.
3. **CPU AI Limitations**: Random decision-making and simplistic Nemesis System updates.
4. **Game Balance**: CPU advantages (higher damage, health regeneration) may feel unfair.
5. **Performance Concerns**: High memory usage from texture loading and frequent file I/O.
6. **Code Duplication**: Repeated logic for player and CPU handling.
7. **Gameplay Issues**: Simplistic jumping, no pause menu, and limited input buffering.
8. **Audio Management**: Incorrect use of `load_music_stream` for hit sounds.
9. **Missing Features**: Commented-out throw/ball mechanics, no difficulty settings.


CPU Action Probabilities at Close Distance
Probability (%)
30
25
20
15
10
5
0
Block
Punch
Kick
Jump
Crouch
Action

6 Suggestions for Improvement

1. **Fix Hardcoded Paths**: Use relative paths or a configuration file.
2. **Improve Error Handling**: Add try-catch for file I/O and display errors on-screen.
3. **Enhance CPU AI**: Use Markov chains or reduce randomness for smarter decisions.
4. **Balance Gameplay**: Equalize damage or add player health regeneration.
5. **Optimize Performance**: Load textures on-demand and buffer move logs.
6. **Reduce Code Duplication**: Unify player/CPU logic in shared functions.
7. **Improve Gameplay**: Add smooth jumping physics, pause menu, and input buffering.
8. **Fix Audio**: Use `load_sound` for hit effects and add volume controls.
9. **Add Features**: Implement throw/ball mechanics, difficulty levels, and tutorials.
10. **Polish UI/UX**: Add countdown timers, animated win screens, and control hints.

7 Chart Analysis

A bar chart was proposed to visualize CPU action probabilities at close distance (< 150 units), with default values: block (30%), punch (20%), kick (20%), jump (15%), crouch (15%). The Nemesis System adjusts these probabilities (e.g., increasing block probability by 50 for frequent punches). Due to LaTeX limitations, the chart is not rendered here but can be implemented in Chart.js for web-based visualization.

8 Conclusion

The 2D fighting game provides a solid foundation with an innovative Nemesis System, but issues like hardcoded paths, limited AI, and unbalanced mechanics require attention. Implementing the suggested improvements will enhance portability, gameplay polish, and player engagement, making the game more robust and enjoyable.