Pong Game Software Design Document

Table of Contents

1. Introduction

This document describes the software design for a modern interpretation of the classic Pong game, implemented in MicroPython for a custom hardware platform. The game features a single-player mode against an AI opponent, power-ups, particle effects, and an audio system, providing an engaging and dynamic gaming experience.

2. System Architecture

The Pong game system is composed of several interconnected components, each responsible for specific aspects of the game's functionality. Below is a high-level overview of the system architecture:

```
graph TD
    A[Main Game Loop] --> B[Pong Game Logic]
    B --> C[Paddle Management]
    B --> D[Ball Physics]
    B --> E[Collision Detection]
    B --> F[Scoring System]
    B --> G[Power-Up System]
    B --> H[Particle System]
    B --> I[AI Opponent]
    B --> J[Audio Engine]
    B --> K[User Interface]
    L[Hardware Inputs] --> A
    A --> M[Display Output]
```

The main game loop orchestrates the overall flow of the game, managing state transitions and coordinating updates to all game components. It interfaces with

hardware inputs to gather user actions and sends rendering commands to the display output.

3. Class Descriptions

3.1 Pong

The `Pong` class serves as the central controller for the game, managing game states, score tracking, and coordinating updates to all game objects.

```
classDiagram
    class Pong {
        +paddle1: Paddle
        +paddle2: Paddle
        +balls: List[Ball]
        +score1: int
        +score2: int
        +running: bool
        +debug: bool
        +particle_system: ParticleSystem
        +power_ups: List[PowerUp]
        +game_state: str
        +audio_engine: AudioEngine
        +update(event: str)
        +draw(lcd: LCD)
        +reset_game()
        +update_ai()
        +apply_power_up(paddle: Paddle, power_up: PowerUp)
    }
```

3.2 Paddle

The `Paddle` class represents a player's paddle, handling movement, power-up effects, and rendering.

```
classDiagram
    class Paddle {
        +x: float
        +y: float
        +width: int
        +height: int
        +velocity: float
        +acceleration: float
        +max_speed: float
        +friction: float
        +color: int
        +power_up_timer: int
```

```
        +power_up_type: str
        +move(direction: str)
        +draw(lcd: LCD)
        +apply_power_up(power_up_type: str)
        +update()
    }
```

3.3 Ball

The `Ball` class manages the ball's position, velocity, and interactions with paddles and walls.

```
classDiagram
    class Ball {
        +x: float
        +y: float
        +radius: int
        +vx: float
        +vy: float
        +max_speed: float
        +color: int
        +controlled: bool
        +controlled_by: Paddle
        +move(paddles: List[Paddle])
        +bounce(paddle_velocity: float)
        +draw(lcd: LCD)
    }
```

3.4 PowerUp

The `PowerUp` class represents power-up items that appear during gameplay, providing various effects when collected.

```
classDiagram
    class PowerUp {
        +x: float
        +y: float
        +radius: int
        +vx: float
        +vy: float
        +type: str
        +color: int
        +letter: str
        +move()
        +draw(lcd: LCD)
    }
```

3.5 ParticleSystem

The `ParticleSystem` class manages visual effects, creating and updating particles for various game events.

```
classDiagram
    class ParticleSystem {
        +particles: List[Particle]
        +max_particles: int
        +add_particle(x: float, y: float, vx: float, vy: float, color: int, lifetime: int)
        +update()
        +draw(lcd: LCD)
    }

    class Particle {
        +x: float
        +y: float
        +vx: float
        +vy: float
        +color: int
        +lifetime: int
        +update()
        +draw(lcd: LCD)
    }

    ParticleSystem --> Particle
```

3.6 AudioEngine

The `AudioEngine` class manages sound generation and playback for various game events.

```
classDiagram
    class AudioEngine {
        +sound_generators: List[SoundGenerator]
        +add_sound_generator(generator: SoundGenerator)
        +update(dt: float)
        +remove_inactive_generators()
        +play_paddle_hit()
        +play_wall_bounce()
        +play_power_up_collect()
        +play_goal()
    }

    class SoundGenerator {
        +pwm: PWM
        +volume: int
```

```
        +attack: float
        +decay: float
        +sustain: float
        +release: float
        +start_freq: int
        +min_freq: int
        +set_frequency(freq: int)
        +set_volume(vol: int)
        +note_on(timeout: float)
        +note_off()
        +update(dt: float)
    }

    AudioEngine --> SoundGenerator
```

4. Game States

The game transitions between different states, each with its own update and draw logic:

```
stateDiagram-v2
    [*] --> Welcome
    Welcome --> Playing: Any button press
    Playing --> Paused: Center button press
    Paused --> Playing: A button press
    Paused --> Welcome: B button press
    Playing --> Goal: Ball enters goal
    Goal --> Playing: Animation complete
```

1. **Welcome**: Displays game instructions and waits for user input to start the game.
2. **Playing**: The main gameplay state where the game logic is updated and rendered.
3. **Paused**: Halts gameplay and displays a pause menu with options to resume or return to the welcome screen.
4. **Goal**: Triggered when a goal is scored, displaying an animation before resuming play.

5. User Interface

The user interface is primarily handled by the `draw` methods of various classes, rendering game elements on the LCD screen. Key UI components include:

1. Paddles and balls
2. Score displays (using seven-segment display simulation)
3. Power-up indicators
4. Particle effects

5. Debug information (when enabled)
6. Welcome screen with scrolling instructions
7. Pause menu
8. Goal animation

6. Audio System

The audio system uses PWM (Pulse Width Modulation) to generate sound effects for various game events. It supports multiple sound generators, allowing for concurrent audio playback.

```
graph TD
    A[AudioEngine] --> B[SoundGenerator]
    B --> C[SquareWave]
    B --> D[SawtoothWave]
    B --> E[SineWave]
    B --> F[NoiseGenerator]
    A --> G[play_paddle_hit]
    A --> H[play_wall_bounce]
    A --> I[play_power_up_collect]
    A --> J[play_goal]
```

Each sound generator uses an ADSR (Attack, Decay, Sustain, Release) envelope for shaping the sound, providing more dynamic and interesting audio feedback.

7. Power-Up System

The power-up system adds variety and strategic depth to the gameplay. Power-ups appear randomly on the screen and can be collected by paddles. Types of power-ups include:

1. Grow: Increases paddle size
2. Shrink: Decreases paddle size
3. Magnet: Attracts or repels the ball
4. Control: Allows the paddle to temporarily "catch" the ball
5. Speed: Increases ball speed
6. Multiball: Adds an additional ball to the game

Power-ups are managed by the `Pong` class, which handles their creation, movement, collision detection, and application of effects.

8. Particle System

The particle system enhances visual feedback by creating small, animated particles for various game events:

1. Paddle hits
2. Wall bounces

3. Power-up collections
4. Goal celebrations

Particles are managed by the `ParticleSystem` class, which handles their creation, updating, and rendering.

9. AI Component

The AI opponent is implemented within the `Pong` class's `update_ai` method. It uses a simple tracking algorithm to move the AI-controlled paddle towards the ball's position. The AI difficulty can be adjusted, affecting the speed and responsiveness of the AI paddle.

```
graph TD
    A[update_ai] --> B{Select target ball}
    B --> C[Calculate target Y position]
    C --> D{AI difficulty}
    D --> |Easy| E[Apply 0.5x speed factor]
    D --> |Medium| F[Apply 0.75x speed factor]
    D --> |Hard| G[Apply 1.0x speed factor]
    E --> H[Move paddle towards target]
    F --> H
    G --> H
```

10. Hardware Integration

The game interfaces with hardware components for input and output:

```
graph TD
    A[Hardware Inputs] --> B[Button A]
    A --> C[Button B]
    A --> D[Joystick Up]
    A --> E[Joystick Down]
    A --> F[Joystick Left]
    A --> G[Joystick Right]
    A --> H[Joystick Center]
    I[Hardware Outputs] --> J[LCD Display]
    I --> K[PWM for Audio]
```

Input is polled in the main game loop and passed to the `Pong` class's `update` method. The LCD display is updated at the end of each game loop iteration.

11. Performance Considerations

To maintain smooth gameplay on the limited hardware:

1. The game uses a frame rate of approximately 100 FPS (10ms sleep between frames).

2. Particle effects are limited to a maximum number of particles.
3. The audio system removes inactive sound generators to conserve resources.
4. Debug information can be toggled on/off to reduce rendering overhead when not needed.

12. Future Enhancements

Potential areas for future improvement include:

1. Networked multiplayer support
2. Additional power-up types and effects
3. Multiple AI difficulty levels with more sophisticated behaviors
4. High score tracking and persistent storage
5. Customizable paddle and ball appearances
6. Background music with volume control
7. Menu system for game options and customization

This modular design allows for easy expansion and modification of game features, providing a solid foundation for future development.