Guide to Building an AI Player and Refining In-Game Mechanics

This guide will help you:

1. Build an AI player for your Pong game.

2. Refine in-game mechanics like scoring, collision detection, and player disconnections.

---

1. Building an AI Player

Overview

The AI player will simulate a human opponent by controlling a paddle. It will:

Track the ball’s position and predict its movement.

Adjust the paddle position to follow the ball.

Implementation

Step 1: Add AI Logic

1. Update your game\_state to include the AI paddle:

game\_state = {

players: ['player1', 'AI'],

ball: { x: 50, y: 50, dx: 1, dy: 1 },

paddles: { player1: 40, ai: 40 },

score: { player1: 0, ai: 0 }

}

2. Define a function to simulate AI behavior:

def update\_ai\_paddle

ai\_position = game\_state[:paddles][:ai]

ball\_position = game\_state[:ball][:y]

if ai\_position < ball\_position

game\_state[:paddles][:ai] += 2 # Move AI paddle down

elsif ai\_position > ball\_position

game\_state[:paddles][:ai] -= 2 # Move AI paddle up

end

# Ensure paddle stays within bounds

game\_state[:paddles][:ai] = [[game\_state[:paddles][:ai], 0].max, 100].min

end

---

Step 2: Integrate AI Paddle Updates

1. Update the game loop to call update\_ai\_paddle periodically:

Thread.new do

loop do

sleep 0.05

# Update ball position

game\_state[:ball][:x] += game\_state[:ball][:dx]

game\_state[:ball][:y] += game\_state[:ball][:dy]

# Check for collisions

if game\_state[:ball][:y] <= 0 || game\_state[:ball][:y] >= 100

game\_state[:ball][:dy] \*= -1

end

# Update AI paddle position

update\_ai\_paddle

# Broadcast the updated game state to all clients

clients.each { |client| client.send(game\_state.to\_json) }

end

end

---

Step 3: AI Paddle Collision

Update the ball’s dx direction when it hits the AI paddle:

if game\_state[:ball][:x] >= 95 &&

(game\_state[:ball][:y] - game\_state[:paddles][:ai]).abs <= 10

game\_state[:ball][:dx] \*= -1

end

---

Step 4: Test the AI

1. Start the backend:

ruby app.rb

2. Connect to the game as player1 using WebSocket.

3. Watch the AI paddle follow the ball in real-time.

---

2. Refining In-Game Mechanics

Scoring System

Step 1: Add Scoring Logic

Update the game loop to track when the ball crosses the screen:

if game\_state[:ball][:x] <= 0

game\_state[:score][:ai] += 1

reset\_ball

elsif game\_state[:ball][:x] >= 100

game\_state[:score][:player1] += 1

reset\_ball

end

Step 2: Reset Ball Position

Add a reset\_ball method:

def reset\_ball

game\_state[:ball] = { x: 50, y: 50, dx: [-1, 1].sample, dy: [-1, 1].sample }

end

Step 3: Display Score

Send the updated score to all clients in the WebSocket broadcast:

clients.each { |client| client.send(game\_state.to\_json) }

---

Collision Detection

Ensure proper ball collisions with paddles and screen edges:

1. Ball collides with player paddle:

if game\_state[:ball][:x] <= 5 &&

(game\_state[:ball][:y] - game\_state[:paddles][:player1]).abs <= 10

game\_state[:ball][:dx] \*= -1

end

2. Ball collides with AI paddle (already covered above).

3. Ball collides with top or bottom edges:

if game\_state[:ball][:y] <= 0 || game\_state[:ball][:y] >= 100

game\_state[:ball][:dy] \*= -1

end

---

Player Disconnections

Handle Disconnections

1. Remove the player’s WebSocket from the clients list:

ws.on :close do |\_event|

clients.delete(ws)

puts "Player disconnected"

end

2. Notify remaining players:

clients.each { |client| client.send({ message: "Player disconnected" }.to\_json) }

---

3. Testing In-Game Mechanics

1. Open two browser clients:

One as player1.

The other observing AI gameplay.

2. Test scenarios:

Paddle collisions.

Ball scoring and resetting.

Disconnection handling.

---

4. Optional Improvements

1. Power-Ups:

Add random power-ups on the field (e.g., increase paddle speed or ball size).

Example:

if game\_state[:ball][:x] == 50 && game\_state[:ball][:y] == 50

game\_state[:paddles][:player1] += 5

end

2. Multiplayer with Remote Players:

Extend the WebSocket logic to manage more than two players.

Implement custom game modes (e.g., 4-player Pong on a square board).

3. Performance Optimization:

Use EventMachine for asynchronous WebSocket handling.

Reduce the frequency of game state broadcasts if performance becomes an issue.

---

Next Steps

Would you like to:

1. Implement a multiplayer mode (e.g., 3+ players)?

2. Add power-ups and custom game modes?

3. Optimize performance for production deployment?

Let me know your focus, and I’ll provide tailored instructions!