

Flappy Bird AI Agent – Dokumentacija

Arman Ossi Loko, IB250209

Contents

Uvod.....	2
Cilj projekta.....	2
Tehnologije korištene	2
Pregled Projekta.....	2
Struktura projekta	2
Funkcionalnosti.....	2
Razvoj sa AI Asistentom	3
Prompt 1: Kreiranje osnovnog projekta	3
Prompt 2: Pобоljšanje performansi AI modela	23
Prompt 3: Konverzija u Node.js aplikaciju	28
Prompt 4: Pобоljšanje korisničkog interfejsa	35
Prompt 5: Pokretanje igre sa SPACE tipkom	41
Prompt 6: Popravka AI Treninga.....	46
Komponente AI Sistema	48
Neuralna Mreža - Detaljno Objašnjenje.....	48
Struktura Naše Neuralne Mreže	48
Ulazni Podaci.....	48
Sigmoid Aktivacijska Funkcija	48
Genetski Algoritam - Detaljno Objašnjenje.....	48
Ciklus Evolucije.....	49
Kako AI Agent Uči Igrati Flappy Bird.....	49
Kako pokrenuti	49

Uvod

Ovaj projekat predstavlja implementaciju popularne igre **Flappy Bird** sa integrisanim AI agentom koji koristi **neuralnu mrežu** i **genetski algoritam** za učenje kako da igra. Projekat je u potpunosti razvijen uz pomoć AI asistenta (Cursor), a svrha je demonstrirati kako se umjetna inteligencija može koristiti za automatsko učenje igranja video igara bez korištenja eksternih AI modela.

Cilj projekta

Glavni ciljevi ovog projekta su:

- Kreirati funkcionalni klon Flappy Bird igre
- Implementirati AI agenta koji može učiti kroz generacije
- Omogućiti korisnicima da igraju sami ili gledaju kako AI uči
- Demonstrirati koncept genetskog algoritma i neuralnih mreža

Tehnologije korištene

- **JavaScript (ES6+)** - Glavni programski jezik
- **HTML5 Canvas** - Za renderovanje igre
- **Node.js + Express** - Server za serviranje aplikacije
- **Neuralna mreža** - Implementirana od nule (bez eksternih biblioteka)
- **Genetski algoritam** - Za evoluciju AI agenata

Pregled Projekta

Struktura projekta

- **index.html** - Glavna HTML stranica koja sadrži strukturu korisničkog interfejsa
- **styles.css** - CSS stilovi za izgled aplikacije
- **game.js** - Flappy Bird engine koji upravlja logikom igre
- **neural-network.js** - Implementacija neuralne mreže od nule
- **ai.js** - AI agent sa genetskim algoritmom
- **main.js** - Glavni kontroler aplikacije koji povezuje sve komponente
- **server.js** - Express server za serviranje aplikacije
- **data/** - Direktorij za sačuvane AI modele
- **chat/** - Historija razgovora sa AI asistentom

Funkcionalnosti

1. **Ručno igranje** - Igrač može igrati Flappy Bird koristeći SPACE ili klik miša
2. **AI Trening** - 50 AI agenata istovremeno uče kroz genetski algoritam
3. **AI Play Mode** - Najbolji trenirani AI demonstrira svoje sposobnosti
4. **Save/Load** - Mogućnost čuvanja i učitavanja treniranih modela
5. **Fast Mode** - Ubrzano treniranje za brže rezultate

Razvoj sa AI Asistentom

Cijeli projekat je razvijen kroz AI interakciju u Cursoru. U nastavku su prikazani svi promptovi (zahtjevi) koji su postavljeni i kompletan kod koji je AI generirao.

Prompt 1: Kreiranje osnovnog projekta

Korisnikov zahtjev:

"In this workspace, I would like to generate the following:

- 1. a Flappy Bird clone with procedurally generated levels*
- 2. a (separate) AI agent that will be capable of learning and adapting to procedurally generated levels in the Flappy Bird clone.*

The end result would be that I am able to open the Flappy Bird clone and play it on my own or that I can let the AI agent to play the game and learn as it plays. Please note that I want to avoid using any external AI models for this."

AI Odgovor:

AI je kreirao kompletnu strukturu projekta sa svim potrebnim datotekama.

Kompletna datoteka: `index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Flappy Bird - AI Learning</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <div class="game-header">
      <h1>Flappy Bird AI</h1>
      <div class="controls">
        <button id="startBtn">Start Game</button>
        <button id="aiBtn">Start AI Learning</button>
        <button id="stopBtn">Stop</button>
        <label>
          <input type="checkbox" id="fastMode"> Fast Mode
(AI only)
        </label>
      </div>
      <div class="stats">
        <div>Score: <span id="score">0</span></div>
```



```
        margin-bottom: 15px;
    }

    .game-header h1 {
        text-align: center;
        color: #333;
        margin-bottom: 15px;
        font-size: 2em;
    }

    .controls {
        display: flex;
        gap: 10px;
        justify-content: center;
        flex-wrap: wrap;
        margin-bottom: 15px;
    }

    .controls button {
        padding: 10px 20px;
        font-size: 14px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        background: #667eea;
        color: white;
        font-weight: bold;
        transition: background 0.3s;
    }

    .controls button:hover {
        background: #5568d3;
    }

    .controls button:active {
        transform: scale(0.95);
    }

    .controls button:disabled {
        background: #ccc;
        cursor: not-allowed;
    }

    .controls label {
        display: flex;
        align-items: center;
        gap: 5px;
        cursor: pointer;
    }

    .stats {
        display: flex;
```

```

    gap: 20px;
    justify-content: center;
    flex-wrap: wrap;
    font-size: 14px;
    color: #666;
}

.stats div {
    padding: 5px 10px;
    background: #f5f5f5;
    border-radius: 5px;
}

#gameCanvas {
    display: block;
    margin: 0 auto;
    border: 3px solid #667eea;
    border-radius: 10px;
    background: #87CEEB;
    cursor: pointer;
}

#instructions {
    text-align: center;
    margin-top: 15px;
    color: #666;
    font-size: 14px;
    line-height: 1.6;
}

#instructions p {
    margin: 5px 0;
}

```

Kompletna datoteka: *neural-network.js*

```

// Simple Neural Network implementation from scratch
class NeuralNetwork {
    constructor(inputNodes, hiddenNodes, outputNodes) {
        this.inputNodes = inputNodes;
        this.hiddenNodes = hiddenNodes;
        this.outputNodes = outputNodes;

        // Initialize weights with random values between -1 and 1
        this.weights_ih = this.createMatrix(hiddenNodes, inputNodes);
        this.weights_ho = this.createMatrix(outputNodes, hiddenNodes);

        // Initialize biases
        this.bias_h = this.createMatrix(hiddenNodes, 1);
        this.bias_o = this.createMatrix(outputNodes, 1);

        this.randomize(this.weights_ih);
    }
}

```

```

        this.randomize(this.weights_ho);
        this.randomize(this.bias_h);
        this.randomize(this.bias_o);

        this.learningRate = 0.1;
    }

    createMatrix(rows, cols) {
        return Array(rows).fill().map(() => Array(cols).fill(0));
    }

    randomize(matrix) {
        for (let i = 0; i < matrix.length; i++) {
            for (let j = 0; j < matrix[i].length; j++) {
                matrix[i][j] = Math.random() * 2 - 1; // Random
between -1 and 1
            }
        }
    }

    sigmoid(x) {
        return 1 / (1 + Math.exp(-x));
    }

    dsigmoid(y) {
        return y * (1 - y);
    }

    feedforward(inputArray) {
        // Convert input array to matrix
        let inputs = inputArray.map(x => [x]);

        // Hidden layer
        let hidden = this.createMatrix(this.hiddenNodes, 1);
        for (let i = 0; i < this.hiddenNodes; i++) {
            let sum = 0;
            for (let j = 0; j < this.inputNodes; j++) {
                sum += this.weights_ih[i][j] * inputs[j][0];
            }
            sum += this.bias_h[i][0];
            hidden[i][0] = this.sigmoid(sum);
        }

        // Output layer
        let outputs = this.createMatrix(this.outputNodes, 1);
        for (let i = 0; i < this.outputNodes; i++) {
            let sum = 0;
            for (let j = 0; j < this.hiddenNodes; j++) {
                sum += this.weights_ho[i][j] * hidden[j][0];
            }
            sum += this.bias_o[i][0];
            outputs[i][0] = this.sigmoid(sum);
        }
    }

```

```

    }

    return outputs.map(x => x[0]);
}

copy() {
    let nn = new NeuralNetwork(this.inputNodes, this.hiddenNodes,
this.outputNodes);
    nn.weights_ih = this.weights_ih.map(row => [...row]);
    nn.weights_ho = this.weights_ho.map(row => [...row]);
    nn.bias_h = this.bias_h.map(row => [...row]);
    nn.bias_o = this.bias_o.map(row => [...row]);
    return nn;
}

mutate(rate) {
    const mutateValue = (val) => {
        if (Math.random() < rate) {
            return val + (Math.random() * 2 - 1) * 0.5;
        }
        return val;
    };

    this.weights_ih = this.weights_ih.map(row =>
row.map(mutateValue));
    this.weights_ho = this.weights_ho.map(row =>
row.map(mutateValue));
    this.bias_h = this.bias_h.map(row => row.map(mutateValue));
    this.bias_o = this.bias_o.map(row => row.map(mutateValue));
}

// Genetic algorithm crossover
static crossover(parent1, parent2) {
    let child = new NeuralNetwork(
        parent1.inputNodes,
        parent1.hiddenNodes,
        parent1.outputNodes
    );

    // Crossover weights
    for (let i = 0; i < child.weights_ih.length; i++) {
        for (let j = 0; j < child.weights_ih[i].length; j++) {
            child.weights_ih[i][j] = Math.random() < 0.5
                ? parent1.weights_ih[i][j]
                : parent2.weights_ih[i][j];
        }
    }

    for (let i = 0; i < child.weights_ho.length; i++) {
        for (let j = 0; j < child.weights_ho[i].length; j++) {
            child.weights_ho[i][j] = Math.random() < 0.5
                ? parent1.weights_ho[i][j]

```



```

        : parent2.weights_ho[i][j];
    }
}

// Crossover biases
for (let i = 0; i < child.bias_h.length; i++) {
    child.bias_h[i][0] = Math.random() < 0.5
        ? parent1.bias_h[i][0]
        : parent2.bias_h[i][0];
}

for (let i = 0; i < child.bias_o.length; i++) {
    child.bias_o[i][0] = Math.random() < 0.5
        ? parent1.bias_o[i][0]
        : parent2.bias_o[i][0];
}

return child;
}
}

```

Kompletna datoteka: *game.js*

```

// Flappy Bird Game Engine
class Game {
    constructor(canvas) {
        this.canvas = canvas;
        this.ctx = canvas.getContext('2d');
        this.width = 1000;
        this.height = 700;
        canvas.width = this.width;
        canvas.height = this.height;

        this.bird = {
            x: 100,
            y: this.height / 2,
            width: 30,
            height: 30,
            velocity: 0,
            gravity: 0.6,
            jumpStrength: -12,
            color: '#FFD700',
            rotation: 0,
            wingAngle: 0,
            wingDirection: 1
        };

        this.pipes = [];
        this.pipeWidth = 60;
        this.pipeGap = 200;
        this.pipeSpeed = 3;
        this.pipeSpacing = 300; // Distance between pipes
    }
}

```

```

        // Ground animation
        this.groundX = 0;
        this.groundHeight = 112;

        this.score = 0;
        this.highScore = parseInt(localStorage.getItem('highScore') ||
'0');
        this.gameOver = false;
        this.gameStarted = false;
        this.frameCount = 0;

        // For AI input
        this.nextPipe = null;

        this.setupEventListeners();
        this.generateInitialPipes();
    }

    setupEventListeners() {
        this.canvas.addEventListener('click', () => {
            if (!this.gameStarted && !this.gameOver) {
                this.flap();
            } else if (this.gameStarted && !this.gameOver) {
                this.flap();
            }
        });

        document.addEventListener('keydown', (e) => {
            if (e.code === 'Space') {
                if (!this.gameStarted && !this.gameOver) {
                    this.start();
                } else if (this.gameStarted && !this.gameOver) {
                    this.flap();
                }
                e.preventDefault();
            }
        });
    }

    generateInitialPipes() {
        this.pipes = [];
        for (let i = 0; i < 5; i++) {
            this.addPipe(this.width + i * this.pipeSpacing);
        }
    }

    addPipe(x) {
        const minHeight = 50;
        const playableHeight = this.height - this.groundHeight;
        const maxHeight = playableHeight - this.pipeGap - minHeight;
        const topHeight = Math.random() * (maxHeight - minHeight) +

```

```

minHeight;

    this.pipes.push({
        x: x,
        topHeight: topHeight,
        bottomY: topHeight + this.pipeGap,
        passed: false
    });
}

flap() {
    this.bird.velocity = this.bird.jumpStrength;
}

update() {
    if (!this.gameStarted || this.gameOver) return;

    this.frameCount++;

    // Use reduced gravity for first 150 frames (~2.5 sec at
60fps)
    const reducedGravityFrames = 150;
    const gravityMultiplier = this.frameCount <
reducedGravityFrames
        ? 0.3 + (this.frameCount / reducedGravityFrames) * 0.7
        : 1.0;

    // Update bird physics
    this.bird.velocity += this.bird.gravity * gravityMultiplier;
    this.bird.y += this.bird.velocity;

    // Boundary check
    if (this.bird.y < 0) {
        this.bird.y = 0;
        this.bird.velocity = 0;
    }
    if (this.bird.y + this.bird.height > this.height -
this.groundHeight) {
        this.gameOver = true;
        this.bird.y = this.height - this.groundHeight -
this.bird.height;
    }

    // Update pipes
    for (let i = this.pipes.length - 1; i >= 0; i--) {
        const pipe = this.pipes[i];
        pipe.x -= this.pipeSpeed;

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + this.pipeWidth < this.bird.x)
{
            pipe.passed = true;

```

```

        this.score++;
        if (this.score > this.highScore) {
            this.highScore = this.score;
            localStorage.setItem('highScore',
this.highScore.toString());
        }
    }

    // Remove pipes that are off screen
    if (pipe.x + this.pipeWidth < 0) {
        this.pipes.splice(i, 1);
        // Add new pipe
        const lastPipe = this.pipes[this.pipes.length - 1];
        if (lastPipe) {
            this.addPipe(lastPipe.x + this.pipeSpacing);
        } else {
            this.addPipe(this.width + this.pipeSpacing);
        }
    }

    // Collision detection
    if (this.checkCollision(pipe)) {
        this.gameOver = true;
    }
}

// Find next pipe for AI
this.nextPipe = this.pipes.find(pipe => pipe.x +
this.pipeWidth >= this.bird.x) || null;
}

checkCollision(pipe) {
    const birdLeft = this.bird.x;
    const birdRight = this.bird.x + this.bird.width;
    const birdTop = this.bird.y;
    const birdBottom = this.bird.y + this.bird.height;

    const pipeLeft = pipe.x;
    const pipeRight = pipe.x + this.pipeWidth;

    // Check if bird is within pipe's x range
    if (birdRight > pipeLeft && birdLeft < pipeRight) {
        // Check if bird hits top or bottom pipe
        if (birdTop < pipe.topHeight || birdBottom > pipe.bottomY)
{
            return true;
        }
    }

    return false;
}

```

```

getAIInputs() {
  // Input features for AI:
  // 1. Bird Y position (normalized)
  // 2. Bird velocity (normalized)
  // 3. Next pipe X distance (normalized)
  // 4. Next pipe top height (normalized)
  // 5. Next pipe gap bottom (normalized)

  const playableHeight = this.height - this.groundHeight;

  if (!this.nextPipe) {
    // If no pipe, return default values
    return [
      this.bird.y / playableHeight,
      this.bird.velocity / 20,
      1,
      0.5,
      0.5
    ];
  }

  const pipe = this.nextPipe;
  const distanceToPipe = (pipe.x - this.bird.x) / this.width;
  const normalizedTop = pipe.topHeight / playableHeight;
  const normalizedBottom = pipe.bottomY / playableHeight;

  return [
    this.bird.y / playableHeight,
    this.bird.velocity / 20,
    distanceToPipe,
    normalizedTop,
    normalizedBottom
  ];
}

draw() {
  // Clear canvas
  this.ctx.fillStyle = '#87CEEB';
  this.ctx.fillRect(0, 0, this.width, this.height);

  // Draw ground
  this.ctx.fillStyle = '#8B4513';
  this.ctx.fillRect(0, this.height - 20, this.width, 20);

  // Draw grass on ground
  this.ctx.fillStyle = '#228B22';
  this.ctx.fillRect(0, this.height - 20, this.width, 5);

  // Draw pipes
  this.ctx.fillStyle = '#228B22';
  for (const pipe of this.pipes) {
    // Top pipe
  }
}

```

```

        this.ctx.fillRect(pipe.x, 0, this.pipeWidth,
pipe.topHeight);
        // Bottom pipe
        this.ctx.fillRect(pipe.x, pipe.bottomY, this.pipeWidth,
this.height - pipe.bottomY);

        // Pipe caps
        this.ctx.fillRect(pipe.x - 5, pipe.topHeight - 20,
this.pipeWidth + 10, 20);
        this.ctx.fillRect(pipe.x - 5, pipe.bottomY, this.pipeWidth
+ 10, 20);
    }

    // Draw bird
    this.ctx.fillStyle = this.bird.color;
    this.ctx.beginPath();
    this.ctx.ellipse(
        this.bird.x + this.bird.width / 2,
        this.bird.y + this.bird.height / 2,
        this.bird.width / 2,
        this.bird.height / 2,
        0,
        0,
        2 * Math.PI
    );
    this.ctx.fill();

    // Draw bird eye
    this.ctx.fillStyle = 'black';
    this.ctx.beginPath();
    this.ctx.arc(
        this.bird.x + this.bird.width * 0.6,
        this.bird.y + this.bird.height * 0.4,
        3,
        0,
        2 * Math.PI
    );
    this.ctx.fill();

    // Draw game over message
    if (this.gameOver) {
        this.ctx.fillStyle = 'rgba(0, 0, 0, 0.5)';
        this.ctx.fillRect(0, 0, this.width, this.height);

        this.ctx.fillStyle = 'white';
        this.ctx.font = 'bold 48px Arial';
        this.ctx.textAlign = 'center';
        this.ctx.fillText('Game Over', this.width / 2, this.height
/ 2 - 50);

        this.ctx.font = '24px Arial';
        this.ctx.fillText(`Score: ${this.score}`, this.width / 2,

```

```

this.height / 2);
    } else if (!this.gameStarted) {
        this.ctx.fillStyle = 'rgba(0, 0, 0, 0.3)';
        this.ctx.fillRect(0, 0, this.width, this.height);

        this.ctx.fillStyle = 'white';
        this.ctx.font = 'bold 32px Arial';
        this.ctx.textAlign = 'center';
        this.ctx.fillText('Click to Start', this.width / 2,
this.height / 2);
    }
}

reset() {
    this.bird.y = this.height / 2;
    this.bird.velocity = 0;
    this.bird.rotation = 0;
    this.score = 0;
    this.gameOver = false;
    this.gameStarted = false;
    this.frameCount = 0;
    this.groundX = 0;
    this.generateInitialPipes();
    this.nextPipe = null;
}

start() {
    this.gameStarted = true;
    this.gameOver = false;
}
}

```

Kompletna datoteka: **ai.js**

```

// AI Agent with Genetic Algorithm
class AIAgent {
    constructor(game, populationSize = 50) {
        this.game = game;
        this.populationSize = populationSize;
        this.population = [];
        this.generation = 1;
        this.bestScore = 0;
        this.bestBird = null;

        // Create initial population
        for (let i = 0; i < this.populationSize; i++) {
            this.population.push({
                brain: new NeuralNetwork(5, 8, 1), // 5 inputs, 8
hidden, 1 output
                fitness: 0,
                score: 0,

```

```

        alive: true,
        game: null
    });
}

this.isRunning = false;
this.fastMode = false;
}

createGameInstance() {
    const canvas = document.createElement('canvas');
    canvas.width = this.game.width;
    canvas.height = this.game.height;
    const gameInstance = new Game(canvas);
    gameInstance.start();
    return gameInstance;
}

start() {
    this.isRunning = true;
    this.generation = 1;

    // Initialize games for each bird
    for (let bird of this.population) {
        bird.game = this.createGameInstance();
        bird.alive = true;
        bird.score = 0;
        bird.fitness = 0;
    }

    this.updateLoop();
}

stop() {
    this.isRunning = false;
}

updateLoop() {
    if (!this.isRunning) return;

    let allDead = true;
    let aliveCount = 0;
    let maxScore = 0;

    // Update all birds
    for (let bird of this.population) {
        if (bird.alive) {
            allDead = false;
            aliveCount++;

            // Get AI decision
            const inputs = bird.game.getAIInputs();

```



```

        const output = bird.brain.feedforward(inputs);

        // If output > 0.5, flap
        if (output[0] > 0.5) {
            bird.game.flap();
        }

        // Update game
        bird.game.update();

        if (bird.game.gameOver) {
            bird.alive = false;
            bird.fitness = this.calculateFitness(bird);
            bird.score = bird.game.score;

            if (bird.score > this.bestScore) {
                this.bestScore = bird.score;
                this.bestBird = bird.brain.copy();
            }

            if (bird.score > maxScore) {
                maxScore = bird.score;
            }
        } else {
            bird.score = bird.game.score;
            if (bird.score > maxScore) {
                maxScore = bird.score;
            }

            // Update fitness continuously
            bird.fitness = this.calculateFitness(bird);
        }

        // Update visual game for first alive bird
        if (aliveCount === 1 && bird.alive) {
            this.syncGameToVisual(bird.game);
        }
    }

    // Update stats
    this.updateStats(aliveCount, maxScore);

    // If all dead, create next generation
    if (allDead) {
        this.evolve();

        // Initialize next generation
        for (let bird of this.population) {
            bird.game = this.createGameInstance();
            bird.alive = true;
            bird.score = 0;
        }
    }
}

```

```

        bird.fitness = 0;
    }
}

// Draw game
this.game.draw();

// Schedule next frame
if (this.isRunning) {
    if (this.fastMode) {
        // Fast mode: run immediately without waiting
        setTimeout(() => this.updateLoop(), 0);
    } else {
        requestAnimationFrame(() => this.updateLoop());
    }
}
}

calculateFitness(bird) {
    // Fitness based on:
    // 1. Survival time (frameCount) - MOST IMPORTANT for early
learning
    // 2. Score (pipes passed)
    // 3. How close to center of gap (small bonus)

    let fitness = 0;

    // Primary: Survival time - crucial for early generations
    // Birds that survive longer get higher fitness even with
score 0
    if (bird.game) {
        fitness += bird.game.frameCount;
    }

    // Secondary: Score multiplier - passing pipes is very
valuable
    fitness += bird.score * 1000;

    // Tertiary: Gap centering bonus (only if approaching a pipe)
    if (bird.game && bird.game.nextPipe) {
        const pipe = bird.game.nextPipe;
        const gapCenter = (pipe.topHeight + pipe.bottomY) / 2;
        const birdCenter = bird.game.bird.y +
bird.game.bird.height / 2;
        const distanceFromCenter = Math.abs(birdCenter -
gapCenter);
        const maxDistance = bird.game.pipeGap / 2;

        // Small bonus for being centered (max 50 points)
        if (distanceFromCenter < maxDistance) {
            fitness += (1 - distanceFromCenter / maxDistance) *

```

```

50;
    }
}

return Math.max(0, fitness);
}

evolve() {
  // Sort by fitness
  this.population.sort((a, b) => b.fitness - a.fitness);

  // Keep top 10% (elite)
  const eliteSize = Math.floor(this.populationSize * 0.1);
  const newPopulation = [];

  // Add elite to new population
  for (let i = 0; i < eliteSize; i++) {
    newPopulation.push({
      brain: this.population[i].brain.copy(),
      fitness: 0,
      score: 0,
      alive: true,
      game: null
    });
  }

  // Generate rest of population through crossover and mutation
  while (newPopulation.length < this.populationSize) {
    // Select two parents (tournament selection)
    const parent1 = this.selectParent();
    const parent2 = this.selectParent();

    // Create child through crossover
    let child = NeuralNetwork.crossover(parent1.brain,
parent2.brain);

    // Mutate child
    child.mutate(0.2); // 20% mutation rate

    newPopulation.push({
      brain: child,
      fitness: 0,
      score: 0,
      alive: true,
      game: null
    });
  }

  this.population = newPopulation;
  this.generation++;
}

```

```

selectParent() {
  // Tournament selection: pick random birds and return the best
one
  const tournamentSize = 5;
  let best = null;
  let bestFitness = -1;

  for (let i = 0; i < tournamentSize; i++) {
    const randomIndex = Math.floor(Math.random() *
this.population.length);
    const candidate = this.population[randomIndex];
    if (candidate.fitness > bestFitness) {
      best = candidate;
      bestFitness = candidate.fitness;
    }
  }

  return best || this.population[0];
}

syncGameToVisual(sourceGame) {
  // Copy state from source game to visual game
  this.game.bird.x = sourceGame.bird.x;
  this.game.bird.y = sourceGame.bird.y;
  this.game.bird.velocity = sourceGame.bird.velocity;
  this.game.score = sourceGame.score;
  this.game.gameOver = sourceGame.gameOver;
  this.game.gameStarted = sourceGame.gameStarted;

  // Copy pipes
  this.game.pipes = [];
  for (let pipe of sourceGame.pipes) {
    this.game.pipes.push({
      x: pipe.x,
      topHeight: pipe.topHeight,
      bottomY: pipe.bottomY,
      passed: pipe.passed
    });
  }

  this.game.nextPipe = sourceGame.nextPipe;
}

updateStats(aliveCount, maxScore) {
  document.getElementById('alive').textContent = aliveCount;
  document.getElementById('generation').textContent =
this.generation;
  document.getElementById('bestScore').textContent =
this.bestScore;
  document.getElementById('score').textContent = maxScore;
}

```

```
    setFastMode(enabled) {  
        this.fastMode = enabled;  
    }  
}
```

Kompletna datoteka: *main.js*

```
// Main application controller  
let game;  
let aiAgent;  
let gameLoopRunning = false;  
let currentMode = null; // 'human' or 'ai'  
  
function init() {  
    const canvas = document.getElementById('gameCanvas');  
    game = new Game(canvas);  
    aiAgent = new AIAgent(game);  
  
    // Update high score display  
    document.getElementById('highScore').textContent = game.highScore;  
  
    // Setup button handlers  
    document.getElementById('startBtn').addEventListener('click',  
startHumanGame);  
    document.getElementById('aiBtn').addEventListener('click',  
startAIGame);  
    document.getElementById('stopBtn').addEventListener('click',  
stopGame);  
    document.getElementById('fastMode').addEventListener('change', (e)  
=> {  
        if (aiAgent) {  
            aiAgent.setFastMode(e.target.checked);  
        }  
    });  
  
    // Start game loop  
    gameLoop();  
}  
  
function startHumanGame() {  
    stopGame();  
    currentMode = 'human';  
    game.reset();  
    game.start();  
    gameLoopRunning = true;  
  
    document.getElementById('aiStats').style.display = 'none';  
    document.getElementById('instructions').style.display = 'block';  
}  
  
function startAIGame() {  
    stopGame();
```

```

    currentMode = 'ai';
    game.reset();
    game.start();
    gameLoopRunning = true;
    aiAgent.start();

    document.getElementById('aiStats').style.display = 'block';
    document.getElementById('instructions').style.display = 'none';
}

function stopGame() {
    gameLoopRunning = false;
    if (aiAgent) {
        aiAgent.stop();
    }
    if (currentMode === 'human') {
        game.gameOver = true;
    }
}

function gameLoop() {
    if (gameLoopRunning && currentMode === 'human') {
        game.update();
    }

    game.draw();

    // Update score display
    document.getElementById('score').textContent = game.score;
    document.getElementById('highScore').textContent = game.highScore;

    requestAnimationFrame(gameLoop);
}

// Initialize when page loads
window.addEventListener('load', init);

```

Prompt 2: Poboljšanje performansi AI modela

Korisnikov zahtjev:

"Please analyze @neural-network.js and @ai.js and tell me what I can improve here. Currently when I train a model, no matter how many generations, when I click 'AI play', the bird often dies within the first few pipes while in training, it reaches scores of 700 for example."

AI Odgovor:

AI je identificirao nekoliko kritičnih problema:

1. **Best Bird Selection Problem** - `saveToFile()` funkcija je tražila "trenutno najbolju pticu" iz populacije sa pogrešnom logikom
2. **Training vs Play Mode Physics Mismatch** - Kompleksna sinkronizacija između instanci igre
3. **State Initialization Timing Issues** - Problemi s inicijalizacijom stanja

AI je implementirao ispravke direktno u `ai.js`:

Ažurirana `ai.js` sa ispravkama

```
// AI Agent with Genetic Algorithm
class AIAgent {
  constructor(game, populationSize = 50) {
    this.game = game;
    this.populationSize = populationSize;
    this.population = [];
    this.generation = 1;
    this.bestScore = 0;
    this.bestBird = null;

    // File-based loading is now manual via loadFromFile()

    // If no saved data, create initial population
    if (this.population.length === 0) {
      for (let i = 0; i < this.populationSize; i++) {
        this.population.push({
          brain: new NeuralNetwork(5, 8, 1),
          fitness: 0,
          score: 0,
          alive: true,
          game: null
        });
      }
    }

    this.isRunning = false;
    this.fastMode = false;
    this.playMode = false;
    this.playBird = null;
    this.playLoopId = null;
    this.playLoopRunning = false;
```

```

        this.playFrameCount = 0;
    }

    // Start play mode - AI just plays without learning
    startPlayMode() {
        if (!this.bestBird && this.population.length === 0) {
            alert('No trained AI found! Please train the AI first.');
```

return false;

```
        }

        this.stop();

        this.game.reset();
        this.game.start();

        setTimeout(() => {
            this.isRunning = true;
            this.playMode = true;

            const brainToUse = this.bestBird ||
this.population[0].brain;
            this.playBird = {
                brain: brainToUse.copy(),
                game: this.createGameInstance(),
                score: 0,
                alive: true
            };

            this.playBird.game.reset();
            this.playBird.game.start();

            this.playLoopRunning = true;
            this.playLoop();
        }, 10);

        return true;
    }

    playLoop() {
        if (!this.isRunning || !this.playMode ||
!this.playLoopRunning) {
            this.playLoopRunning = false;
            return;
        }

        this.playFrameCount++;

        if (this.playBird && this.playBird.alive &&
!this.playBird.game.gameOver) {
            const inputs = this.playBird.game.getAIInputs();
            const output = this.playBird.brain.feedforward(inputs);
```



```

        if (output[0] > 0.5) {
            this.playBird.game.flap();
        }

        this.playBird.game.update();

        const isStuckAtTop = this.playFrameCount > 1 &&
            this.playBird.game.bird.y === 0 &&
            this.playBird.game.bird.velocity ===
0;

        if (!isStuckAtTop) {
            this.syncGameToVisual(this.playBird.game);
        } else {
            this.playBird.game.reset();
            this.playBird.game.start();
            this.playFrameCount = 0;
        }

        this.playBird.score = this.playBird.game.score;
        document.getElementById('score').textContent =
this.playBird.score;

        if (this.playBird.game.gameOver) {
            this.playBird.alive = false;
        }
    }

    this.game.draw();

    if (this.isRunning && this.playMode && this.playLoopRunning) {
        this.playLoopId = requestAnimationFrame(() =>
this.playLoop());
    }
}

// Save AI knowledge to file - FIXED VERSION
saveToFile() {
    try {
        // Always use the bestBird that actually achieved the
bestScore
        if (!this.bestBird) {
            console.error('No trained bird available to save');
            alert('No trained AI found! Train the AI first before
saving.');
```

saving.');

```

            return false;
        }

        if (this.bestScore === 0) {
            console.warn('Best score is 0 - AI has not scored
yet');
```

yet');

```

            if (!confirm('The AI has not scored any points yet.
```

```

Save anyway?')) {
    return false;
}

    console.log(`Saving AI with bestScore: ${this.bestScore},
generation: ${this.generation}`);

    const saveData = {
        generation: this.generation,
        bestScore: this.bestScore,
        bestBird: this.bestBird.serialize(),
        population: this.population.map(bird => ({
            brain: bird.brain.serialize(),
            score: bird.score,
            fitness: bird.fitness
        })),
        savedAt: new Date().toISOString(),
        version: '2.0'
    };

    const jsonString = JSON.stringify(saveData, null, 2);
    const blob = new Blob([jsonString], { type:
'application/json' });
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = `flappy-bird-ai-gen${this.generation}-
score${this.bestScore}.json`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    URL.revokeObjectURL(url);

    return true;
} catch (error) {
    console.error('Error saving:', error);
    return false;
}

}

// Load AI knowledge from file
loadFromFile(fileContent) {
    try {
        let data;
        if (typeof fileContent === 'string') {
            data = JSON.parse(fileContent);
        } else {
            data = fileContent;
        }

        this.generation = data.generation || 1;

```

```

        this.bestScore = data.bestScore || 0;

        if (data.population && data.population.length > 0) {
            this.population = data.population.map(saved => {
                const brainData = typeof saved.brain === 'string'
                    ? JSON.parse(saved.brain)
                    : saved.brain;
                return {
                    brain: NeuralNetwork.deserialize(brainData),
                    fitness: 0,
                    score: 0,
                    alive: true,
                    game: null
                };
            });
        }

        if (data.bestBird) {
            const bestBirdData = typeof data.bestBird === 'string'
                ? JSON.parse(data.bestBird)
                : data.bestBird;
            this.bestBird =
NeuralNetwork.deserialize(bestBirdData);
        }

        return true;
    } catch (error) {
        console.error('Error loading:', error);
        return false;
    }
}

// ... rest of the AIAgent class methods remain the same
}

```

Prompt 3: Konverzija u Node.js aplikaciju

Korisnikov zahtjev:

"Please turn this workspace / Flappy Bird clone into a NodeJS application. Make sure that upon loading, the page automatically imports the latest saved AI agent data from the data/ directory (recursively find the latest one created)."

AI Odgovor:

AI je kreirao Express server i ažurirao `main.js`.

Nova datoteka: `server.js`

```
const express = require('express');
const path = require('path');
const fs = require('fs').promises;

const app = express();
const PORT = process.env.PORT || 3000;

// Serve static files from the current directory
app.use(express.static(__dirname));

// API endpoint to get the latest AI data file
app.get('/api/latest-ai-data', async (req, res) => {
  try {
    const dataDir = path.join(__dirname, 'data');

    // Check if data directory exists
    try {
      await fs.access(dataDir);
    } catch (error) {
      return res.json({ success: false, message: 'No AI data found yet' });
    }

    // Read all files in data directory recursively
    const files = await getAllJsonFiles(dataDir);

    if (files.length === 0) {
      return res.json({ success: false, message: 'No AI data files found' });
    }

    // Get file stats to find the latest one
    const filesWithStats = await Promise.all(
      files.map(async (file) => {
        const stats = await fs.stat(file);
        return {
          path: file,
          mtime: stats.mtime,

```

```

        name: path.basename(file)
    });
    })
};

// Sort by modification time (newest first)
filesWithStats.sort((a, b) => b.mtime - a.mtime);

const latestFile = filesWithStats[0];

// Read the file content
const fileContent = await fs.readFile(latestFile.path, 'utf-
8');

const aiData = JSON.parse(fileContent);

res.json({
    success: true,
    data: aiData,
    filename: latestFile.name,
    savedAt: aiData.savedAt || latestFile.mtime.toISOString()
});

} catch (error) {
    console.error('Error loading AI data:', error);
    res.status(500).json({
        success: false,
        message: 'Error loading AI data: ' + error.message
    });
}
});

// Recursive function to get all JSON files in a directory
async function getAllJsonFiles(dir) {
    const files = [];

    try {
        const entries = await fs.readdir(dir, { withFileTypes: true
});

        for (const entry of entries) {
            const fullPath = path.join(dir, entry.name);

            if (entry.isDirectory()) {
                const subFiles = await getAllJsonFiles(fullPath);
                files.push(...subFiles);
            } else if (entry.isFile() && entry.name.endsWith('.json'))
{
                files.push(fullPath);
            }
        }
    } catch (error) {
        console.warn(`Could not read directory ${dir}:`,

```

```

error.message);
    }

    return files;
}

// Serve index.html for the root path
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'index.html'));
});

// Start the server
app.listen(PORT, () => {
    console.log(`🐦 Flappy Bird AI Server is running!`);
    console.log(`💰 Server: http://localhost:${PORT}`);
    console.log(`📁 The AI will automatically load the latest saved data on startup`);
    console.log(`\nPress Ctrl+C to stop the server`);
});

```

Ažurirana **main.js** sa auto-load funkcijom

```

// Main application controller
let game;
let aiAgent;
let gameLoopRunning = false;
let currentMode = null; // 'human', 'ai', or 'ai-play'

function init() {
    const canvas = document.getElementById('gameCanvas');
    game = new Game(canvas);
    aiAgent = new AIAgent(game);

    // Auto-load latest AI data from server
    loadLatestAIData();

    // Update high score display
    document.getElementById('highScore').textContent = game.highScore;

    // Setup button handlers
    document.getElementById('startBtn').addEventListener('click', startHumanGame);
    document.getElementById('aiBtn').addEventListener('click', startAIGame);
    document.getElementById('aiPlayBtn').addEventListener('click', startAIPlay);
    document.getElementById('stopBtn').addEventListener('click', stopGame);
    document.getElementById('saveBtn').addEventListener('click', () => {
        if (aiAgent) {

```

```

        aiAgent.saveToFile();
    }
});

// File input for loading
const fileInput = document.getElementById('fileInput');
fileInput.addEventListener('change', (e) => {
    const file = e.target.files[0];
    if (!file) return;

    const reader = new FileReader();
    reader.onload = (event) => {
        try {
            const fileContent = event.target.result;
            if (aiAgent) {
                if (aiAgent.loadFromFile(fileContent)) {
                    alert(`Successfully loaded AI from generation
${aiAgent.generation}`);
                }
            }
        } catch (error) {
            alert('Error reading file: ' + error.message);
        }
    };
    reader.readAsText(file);
    fileInput.value = '';
});

document.getElementById('loadBtn').addEventListener('click', () => {
    fileInput.click();
});

document.getElementById('clearBtn').addEventListener('click', ()
=> {
    if (aiAgent && confirm('Are you sure you want to reset the
AI?')) {
        aiAgent.clearSavedData();
    }
});

document.getElementById('fastMode').addEventListener('change', (e)
=> {
    if (aiAgent) {
        aiAgent.setFastMode(e.target.checked);
    }
});

// Start game loop
gameLoop();
}

```

```

// Function to automatically load the latest AI data from server
async function loadLatestAIData() {
  try {
    console.log('🔍 Looking for the latest AI data...');
    const response = await fetch('/api/latest-ai-data');
    const result = await response.json();

    if (result.success && result.data) {
      console.log('✅ Found AI data: ${result.filename}');
      console.log('📊 Generation: ${result.data.generation},
Best Score: ${result.data.bestScore}');

      if (aiAgent && aiAgent.loadFromFile(result.data)) {
        console.log('🤖 Successfully loaded AI agent!');

        const saveStatus =
document.getElementById('saveStatus');
        if (saveStatus) {
          saveStatus.textContent = `Auto-loaded: Gen
${result.data.generation}`;
          saveStatus.style.color = '#10b981';
        }

        showNotification(`AI Loaded: Generation
${result.data.generation}`, 'success');
      }
    } else {
      console.log('❌ No saved AI data found. Starting fresh!');
      showNotification('No saved AI data found. Train a new
AI!', 'info');
    }
  } catch (error) {
    console.warn('Could not auto-load AI data:', error.message);
  }
}

// Function to show a temporary notification
function showNotification(message, type = 'info') {
  const notification = document.createElement('div');
  notification.textContent = message;
  notification.style.cssText = `
    position: fixed;
    top: 20px;
    right: 20px;
    padding: 16px 24px;
    background: ${type === 'success' ? '#10b981' : type ===
'error' ? '#ef4444' : '#3b82f6'};
    color: white;
    border-radius: 8px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
  `;

```



```

        font-size: 14px;
        font-weight: 500;
        z-index: 1000;
        animation: slideIn 0.3s ease-out;
        max-width: 400px;
    `;

    document.body.appendChild(notification);

    setTimeout(() => {
        notification.style.animation = 'slideOut 0.3s ease-in';
        setTimeout(() => {
            document.body.removeChild(notification);
        }, 300);
    }, 5000);
}

function startHumanGame() {
    stopGame();
    currentMode = 'human';
    game.reset();
    game.start();
    gameLoopRunning = true;

    document.getElementById('aiStats').style.display = 'none';
    document.getElementById('instructions').style.display = 'block';
}

function startAIGame() {
    stopGame();
    currentMode = 'ai';
    game.reset();
    game.start();
    gameLoopRunning = true;

    document.getElementById('aiStats').style.display = 'block';
    document.getElementById('instructions').style.display = 'none';

    aiAgent.start();
}

function startAIPlay() {
    stopGame();
    currentMode = 'ai-play';
    game.reset();
    game.start();
    gameLoopRunning = true;

    document.getElementById('aiStats').style.display = 'none';
    document.getElementById('instructions').style.display = 'none';

    if (aiAgent) {

```

```

        if (!aiAgent.startPlayMode()) {
            currentMode = null;
            gameLoopRunning = false;
            document.getElementById('instructions').style.display =
'block';
        }
    }
}

function stopGame() {
    gameLoopRunning = false;
    if (aiAgent) {
        aiAgent.stop();
    }
    if (currentMode === 'human') {
        game.gameOver = true;
    }
}

function gameLoop() {
    if (gameLoopRunning && currentMode === 'human') {
        game.update();
        game.draw();
        document.getElementById('score').textContent = game.score;
        document.getElementById('highScore').textContent =
game.highScore;
    } else if (currentMode !== 'ai' && currentMode !== 'ai-play') {
        game.draw();
    }

    document.getElementById('highScore').textContent = game.highScore;

    requestAnimationFrame(gameLoop);
}

window.addEventListener('load', init);

```

Prompt 4: Poboljšanje korisničkog interfejsa

Korisnikov zahtjev:

"Please rework the UI of this Flappy Bird clone a bit. Do not touch the game window or its size. Currently, the UI feels cluttered."

AI Odgovor:

AI je kompletno reorganizirao UI.

Ažurirani [index.html](#)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Flappy Bird - AI Learning</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <div class="game-header">
      <h1>🐦 Flappy Bird AI</h1>

      <div class="controls-wrapper">
        <div class="control-group">
          <button id="startBtn">🎮 Play</button>
          <button id="aiBtn">🤖 Train AI</button>
          <button id="aiPlayBtn">👁 Watch AI</button>
          <button id="stopBtn">Stop</button>
        </div>
        <div class="control-group secondary">
          <button id="saveBtn">💾 Save</button>
          <button id="loadBtn">📁 Load</button>
          <button id="clearBtn">🗑 Reset</button>
          <input type="file" id="fileInput" accept=".json"
style="display: none;">
          <label class="fast-mode-toggle">
            <input type="checkbox" id="fastMode">
            <span>⚡ Fast Mode</span>
          </label>
        </div>
      </div>

      <div class="stats-container">
        <div class="stat-card">
          <span class="stat-label">Score</span>
          <span class="stat-value" id="score">0</span>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

```

        <div class="stat-card">
            <span class="stat-label">Best</span>
            <span class="stat-value" id="highScore">0</span>
        </div>
        <div id="aiStats" class="ai-stats-group"
style="display: none;">
            <div class="stat-card">
                <span class="stat-label">Gen</span>
                <span class="stat-value"
id="generation">1</span>
            </div>
            <div class="stat-card">
                <span class="stat-label">AI Best</span>
                <span class="stat-value"
id="bestScore">0</span>
            </div>
            <div class="stat-card">
                <span class="stat-label">Alive</span>
                <span class="stat-value" id="alive">0</span>
            </div>
        </div>
        <div id="saveStatus" class="save-status"></div>
    </div>

    <canvas id="gameCanvas"></canvas>

    <div id="instructions">
        <p>Press <strong>SPACE</strong> or <strong>CLICK</strong>
to flap •
        Train AI to learn automatically • Save/Load your best
models</p>
    </div>
</div>
<script src="neural-network.js"></script>
<script src="game.js"></script>
<script src="ai.js"></script>
<script src="main.js"></script>
</body>
</html>

```

Ažurirani *styles.css*

```

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
}

```

```
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: 30px 20px;
}

.container {
    background: white;
    border-radius: 20px;
    padding: 30px;
    box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
    max-width: 1100px;
    width: 100%;
}

.game-header {
    margin-bottom: 20px;
}

.game-header h1 {
    text-align: center;
    color: #333;
    margin-bottom: 25px;
    font-size: 2.5em;
    font-weight: 700;
    letter-spacing: -0.5px;
}

.controls-wrapper {
    display: flex;
    flex-direction: column;
    gap: 12px;
    margin-bottom: 20px;
}

.control-group {
    display: flex;
    gap: 10px;
    justify-content: center;
    flex-wrap: wrap;
}

.control-group.secondary {
    align-items: center;
}

.control-group button {
    padding: 12px 20px;
    font-size: 14px;
    border: none;
    border-radius: 8px;
```

```
    cursor: pointer;
    background: #667eea;
    color: white;
    font-weight: 600;
    transition: all 0.2s ease;
    box-shadow: 0 2px 8px rgba(102, 126, 234, 0.3);
}

.control-group.secondary button {
    padding: 10px 18px;
    font-size: 13px;
    background: #9ca3af;
    box-shadow: 0 2px 6px rgba(156, 163, 175, 0.3);
}

.control-group button:hover {
    background: #5568d3;
    transform: translateY(-2px);
    box-shadow: 0 4px 12px rgba(102, 126, 234, 0.4);
}

.control-group.secondary button:hover {
    background: #6b7280;
}

.fast-mode-toggle {
    display: flex;
    align-items: center;
    gap: 6px;
    cursor: pointer;
    font-size: 13px;
    font-weight: 600;
    color: #4b5563;
    padding: 8px 14px;
    background: #f3f4f6;
    border-radius: 8px;
    transition: all 0.2s ease;
}

.fast-mode-toggle:hover {
    background: #e5e7eb;
}

.stats-container {
    display: flex;
    gap: 12px;
    justify-content: center;
    flex-wrap: wrap;
    align-items: center;
}

.stat-card {
```

```
display: flex;
flex-direction: column;
align-items: center;
gap: 4px;
padding: 10px 18px;
background: linear-gradient(135deg, #f9fafb 0%, #f3f4f6 100%);
border-radius: 10px;
box-shadow: 0 2px 6px rgba(0, 0, 0, 0.06);
min-width: 80px;
}

.stat-label {
font-size: 11px;
font-weight: 600;
color: #6b7280;
text-transform: uppercase;
letter-spacing: 0.5px;
}

.stat-value {
font-size: 20px;
font-weight: 700;
color: #667eea;
line-height: 1;
}

.ai-stats-group {
display: flex;
gap: 12px;
padding-left: 12px;
border-left: 3px solid #e5e7eb;
}

.save-status {
text-align: center;
font-size: 13px;
margin-top: 12px;
color: #10b981;
min-height: 18px;
font-weight: 500;
}

#gameCanvas {
display: block;
margin: 0 auto;
border: 4px solid #5CBE5C;
border-radius: 15px;
background: #4EC0CA;
cursor: pointer;
box-shadow: 0 8px 24px rgba(92, 190, 92, 0.3);
}
```

```
#instructions {
  text-align: center;
  margin-top: 20px;
  padding: 12px 20px;
  background: #f9fafb;
  border-radius: 8px;
  color: #6b7280;
  font-size: 13px;
  line-height: 1.6;
}

#instructions strong {
  color: #374151;
  font-weight: 600;
}

@media (max-width: 768px) {
  body {
    padding: 15px;
  }

  .container {
    padding: 15px;
  }

  .game-header h1 {
    font-size: 1.5em;
  }

  .control-group button {
    padding: 10px 16px;
    font-size: 13px;
  }
}
```


Prompt 5: Pokretanje igre sa SPACE tipkom

Korisnikov zahtjev:

"Please analyze this workspace and make it possible to start playing the game by just pressing SPACE."

AI Odgovor:

AI je dodao globalni SPACE handler, zatim ga ažurirao da ne ometa flap funkcionalnost.

Finalna verzija `main.js` sa SPACE handlerom

```
// Main application controller
let game;
let aiAgent;
let gameLoopRunning = false;
let currentMode = null; // 'human', 'ai', or 'ai-play'

function init() {
  const canvas = document.getElementById('gameCanvas');
  game = new Game(canvas);
  aiAgent = new AIAgent(game);

  // Auto-load latest AI data from server
  loadLatestAIData();

  // Update high score display
  document.getElementById('highScore').textContent = game.highScore;

  // Setup button handlers
  document.getElementById('startBtn').addEventListener('click',
startHumanGame);
  document.getElementById('aiBtn').addEventListener('click',
startAIGame);
  document.getElementById('aiPlayBtn').addEventListener('click',
startAIPlay);
  document.getElementById('stopBtn').addEventListener('click',
stopGame);
  document.getElementById('saveBtn').addEventListener('click', () =>
{
    if (aiAgent) {
      aiAgent.saveToFile();
    }
  }));

  // File input for loading
  const fileInput = document.getElementById('fileInput');
  fileInput.addEventListener('change', (e) => {
    const file = e.target.files[0];
    if (!file) return;

    const reader = new FileReader();
    reader.onload = (event) => {
```

```

        try {
            const fileContent = event.target.result;
            if (aiAgent) {
                if (aiAgent.loadFromFile(fileContent)) {
                    alert(`Successfully loaded AI from generation
${aiAgent.generation}`);
                }
            }
        } catch (error) {
            alert('Error reading file: ' + error.message);
        }
    };
    reader.readAsText(file);
    fileInput.value = '';
});

document.getElementById('loadBtn').addEventListener('click', () =>
{
    fileInput.click();
});

document.getElementById('clearBtn').addEventListener('click', ()
=> {
    if (aiAgent && confirm('Are you sure you want to reset the
AI?')) {
        aiAgent.clearSavedData();
    }
});

document.getElementById('fastMode').addEventListener('change', (e)
=> {
    if (aiAgent) {
        aiAgent.setFastMode(e.target.checked);
    }
});

// Global SPACE key handler to start game if not running
// This handler only starts/restarts the game, never interferes
with flapping
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        // Only handle starting/restarting if:
        // 1. No game mode is active (currentMode === null)
        // 2. Human game is over (needs restart)
        // Otherwise, let the game's own handler take care of
everything
        const shouldStart = currentMode === null || (currentMode
=== 'human' && game.gameOver);

        if (shouldStart) {
            e.preventDefault();
            startHumanGame();
        }
    }
});

```

```

        }
        // If game is running normally, do nothing - let game.js
handler handle flapping
    }
    }, true); // Use capture phase to ensure this runs first

    // Start game loop
    gameLoop();
}

function startHumanGame() {
    stopGame();
    currentMode = 'human';
    game.reset();
    game.start();
    gameLoopRunning = true;

    document.getElementById('aiStats').style.display = 'none';
    document.getElementById('instructions').style.display = 'block';
}

function startAIGame() {
    stopGame();
    currentMode = 'ai';
    game.reset();
    game.start();
    gameLoopRunning = true;

    document.getElementById('aiStats').style.display = 'block';
    document.getElementById('instructions').style.display = 'none';

    aiAgent.start();
}

function startAIPlay() {
    stopGame();
    currentMode = 'ai-play';
    game.reset();
    game.start();
    gameLoopRunning = true;

    document.getElementById('aiStats').style.display = 'none';
    document.getElementById('instructions').style.display = 'none';

    if (aiAgent) {
        if (!aiAgent.startPlayMode()) {
            currentMode = null;
            gameLoopRunning = false;
            document.getElementById('instructions').style.display =
'block';
        }
    }
}

```

```

}

function stopGame() {
  gameLoopRunning = false;
  if (aiAgent) {
    aiAgent.stop();
  }
  if (currentMode === 'human') {
    game.gameOver = true;
  }
}

function gameLoop() {
  if (gameLoopRunning && currentMode === 'human') {
    game.update();
    game.draw();
    document.getElementById('score').textContent = game.score;
    document.getElementById('highScore').textContent =
game.highScore;
  } else if (currentMode !== 'ai' && currentMode !== 'ai-play') {
    game.draw();
  }

  document.getElementById('highScore').textContent = game.highScore;

  requestAnimationFrame(gameLoop);
}

// Function to automatically load the latest AI data from server
async function loadLatestAIData() {
  try {
    console.log('🔍 Looking for the latest AI data...');
    const response = await fetch('/api/latest-ai-data');
    const result = await response.json();

    if (result.success && result.data) {
      console.log(`✅ Found AI data: ${result.filename}`);

      if (aiAgent && aiAgent.loadFromFile(result.data)) {
        console.log(`🤖 Successfully loaded AI agent!`);

        const saveStatus =
document.getElementById('saveStatus');
        if (saveStatus) {
          saveStatus.textContent = `Auto-loaded: Gen
${result.data.generation}`;
          saveStatus.style.color = '#10b981';
        }

        showNotification(`AI Loaded: Generation
${result.data.generation}`, 'success');

```

```

    }
    } else {
        console.log('❗ No saved AI data found. Starting fresh!');
        showNotification('No saved AI data found. Train a new
AI!', 'info');
    }
    } catch (error) {
        console.warn('Could not auto-load AI data:', error.message);
    }
}

function showNotification(message, type = 'info') {
    const notification = document.createElement('div');
    notification.textContent = message;
    notification.style.cssText = `
        position: fixed;
        top: 20px;
        right: 20px;
        padding: 16px 24px;
        background: ${type === 'success' ? '#10b981' : type ===
'error' ? '#ef4444' : '#3b82f6'};
        color: white;
        border-radius: 8px;
        box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
        font-size: 14px;
        font-weight: 500;
        z-index: 1000;
        max-width: 400px;
    `;

    document.body.appendChild(notification);

    setTimeout(() => {
        document.body.removeChild(notification);
    }, 5000);
}

window.addEventListener('load', init);

```

Prompt 6: Popravka AI Treninga

Korisnikov zahtjev:

"Please analyze this @neural-network.js and @ai.js and tell me what is wrong here. When I run the train mode, the agent never reaches more than score 6 for some reason."

AI Odgovor:

AI je identificirao tri kritična problema koji su onemogućavali uspješan trening:

Problem 1: Fitness funkcija nije imala metriku preživljavanja

Prije: Kada su sve ptice umirale sa skorom 0, sve su imale sličan fitness (samo gap bonus 0-50). Genetski algoritam nije mogao razlikovati bolje od lošijih ponašanja.

Poslije: Fitness sada koristi frameCount kao primarnu metriku:

```
fitness = frameCount + (score * 1000) + gap_bonus
```

Ptica koja preživi 300 frejma dobija fitness ~325, dok ona koja umre nakon 60 frejma dobija ~85. Ova 4x razlika omogućava selekciju.

Problem 2: Gravitacija je koristila realno vrijeme umjesto game frame

Prije: Korišteno *Date.now()* za kalkulaciju gravitacije. U brzom modu (5x ažuriranja po frejmu), period smanjene gravitacije je trajao 5x duže u game vremenu, stvarajući nekonzistentnu fiziku.

Poslije: Koristi *frameCount* za kalkulaciju gravitacije. Fizika je sada konzistentna bez obzira na mod.

Problem 3: Nije postojao brojač za frames

Prije: Nije postojao način za praćenje koliko dugo je ptica preživjela.

Poslije: game.frameCount prati stvarni napredak igre.

Ažurirani *game.js* sa frameCount

```
update() {
  if (!this.gameStarted || this.gameOver) return;

  // Increment frame counter for fitness calculation
  this.frameCount++;

  // Use reduced gravity for first 150 frames (~2.5 sec at 60fps)
  const reducedGravityFrames = 150;
  const gravityMultiplier = this.frameCount < reducedGravityFrames
    ? 0.3 + (this.frameCount / reducedGravityFrames) * 0.7
    : 1.0;

  // Update bird physics with adjusted gravity
  this.bird.velocity += this.bird.gravity * gravityMultiplier;
  this.bird.y += this.bird.velocity;

  // ... rest of update logic
```

```

}

reset() {
  // ... other resets
  this.frameCount = 0;
}

```

Ažurirani *ai.js* calculateFitness

```

calculateFitness(bird) {
  let fitness = 0;

  // Primary: Survival time - crucial for early generations
  if (bird.game) {
    fitness += bird.game.frameCount;
  }

  // Secondary: Score multiplier - passing pipes is very valuable
  fitness += bird.score * 1000;

  // Tertiary: Gap centering bonus
  if (bird.game && bird.game.nextPipe) {
    const pipe = bird.game.nextPipe;
    const gapCenter = (pipe.topHeight + pipe.bottomY) / 2;
    const birdCenter = bird.game.bird.y + bird.game.bird.height /
2;

    const distanceFromCenter = Math.abs(birdCenter - gapCenter);
    const maxDistance = bird.game.pipeGap / 2;

    if (distanceFromCenter < maxDistance) {
      fitness += (1 - distanceFromCenter / maxDistance) * 50;
    }
  }

  return Math.max(0, fitness);
}

```

Komponente AI Sistema

Naš AI agent ima tri ključne komponente:

- **Okruženje (Environment)** je sama igra Flappy Bird. Okruženje pruža informacije agentu o trenutnom stanju igre - gdje se ptica nalazi, gdje su cijevi, kolika je brzina padanja. Također, okruženje prima akcije od agenta (skok ili ne) i ažurira stanje igre na osnovu te akcije.
- **Neuralna mreža** predstavlja "mozak" agenta. Ona prima informacije iz okruženja, obrađuje ih kroz svoje slojeve, i donosi odluku koju akciju poduzeti.
- **Genetski algoritam** je mehanizam učenja. On evoluirao populaciju agenata kroz generacije. Agenti koji bolje performiraju imaju veću šansu da prenesu svoje "gene" (težine neuralne mreže) na sljedeću generaciju.

Neuralna Mreža - Detaljno Objašnjenje

Struktura Naše Neuralne Mreže

Naša neuralna mreža ima tri sloja:

- **Ulazni sloj** - 5 neurona (informacije o stanju igre)
- **Skriiveni sloj** - 8 neurona (obrada informacija)
- **Izlazni sloj** - 1 neuron (odluka: skočiti ili ne)

Ulazni Podaci

Pet informacija koje ptica "vidi":

1. **Pozicija ptice (Y)** - normalizirana vertikalna pozicija
2. **Brzina padanja** - koliko brzo ptica pada ili raste
3. **Udaljenost do sljedeće cijevi** - koliko je daleko cijev
4. **Visina gornje cijevi** - gdje završava gornja cijev
5. **Pozicija donje cijevi** - gdje počinje donja cijev

Sigmoid Aktivacijska Funkcija

Pretvara bilo koji broj u vrijednost između 0 i 1:

Izlaz > 0.5 = SKOČI

Izlaz ≤ 0.5 = NE SKAČI

Genetski Algoritam - Detaljno Objašnjenje

Ključni koncepti:

1. **Populacija** - 50 AI agenata koji istovremeno igraju
2. **Fitness** - mjera koliko je ptica dobra (bodovi + bonus za poziciju)
3. **Selekcija** - turnirska selekcija za odabir roditelja
4. **Crossover** - kombiniranje gena dva roditelja
5. **Mutacija** - male nasumične promjene (20% šansa)
6. **Elitizam** - top 10% automatski prelazi u novu generaciju

Ciklus Evolucije

1. Kreiranje početne populacije s nasumičnim težinama
2. Sve ptice igraju dok ne umru
3. Izračunavanje fitnessa za svaku pticu
4. Selekcija najboljih za roditelje
5. Crossover i mutacija za kreiranje nove generacije
6. Ponavljanje procesa

Kako AI Agent Uči Igrati Flappy Bird

Kroz Generacije

- **Generacija 1:** Haotično ponašanje, većina ptica umire na prvoj cijevi
- **Generacija 5:** Bolja kontrola visine, neke prolaze 2-3 cijevi
- **Generacija 20:** Osnovna strategija naučena, 20-50 cijevi
- **Generacija 50+:** Mogu igrati "beskonačno", stotine cijevi bez greške

Kako pokrenuti

1. Instalirati Node.js
2. U terminalu pokrenuti:

```
npm install  
npm run start
```

3. Otvoriti browser na <http://localhost:3000/>
4. Kliknuti "Train AI" i gledati kako AI uči!

Dokumentacija kreirana: Decembar 2025

Projekat razvijen uz pomoć Cursor AI asistenta