

Create a browser-based flappy bird clone

Exported on 11/12/2025 at 11:30:19 MST from Cursor (2.0.69)

User

I want you to create a game that is exactly like flappy bird and runs in the browser, use the space key to flap

Cursor

Creating a browser-based Flappy Bird game. Checking the project directory, then adding the necessary files.

Creating the Flappy Bird game files:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Flappy Bird</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Flappy Bird</h1>
    <canvas id="gameCanvas" width="400" height="600"></
canvas>
    <div class="instructions">
      <p>Press <strong>SPACE</strong> to flap and avoid
the pipes!</p>
      <p id="score">Score: 0</p>
      <p id="gameOver" class="hidden">Game Over! Press
SPACE to restart</p>
    </div>
  </div>
  <script src="game.js"></script>
</body>
</html>
```

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

body {
  font-family: 'Arial', sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.container {
  text-align: center;
  background: white;
  padding: 20px;
  border-radius: 15px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

h1 {
  color: #333;
  margin-bottom: 15px;
  font-size: 2.5em;
}

#gameCanvas {
  border: 3px solid #333;
  border-radius: 10px;
  background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB
70%, #90EE90 70%, #90EE90 100%);
  display: block;
  margin: 0 auto;
}

.instructions {
```

```
    margin-top: 15px;
    color: #555;
}
```

```
.instructions p {
    margin: 8px 0;
    font-size: 1.1em;
}
```

```
#score {
    font-size: 1.5em;
    font-weight: bold;
    color: #667eea;
}
```

```
#gameOver {
    font-size: 1.3em;
    font-weight: bold;
    color: #e74c3c;
    margin-top: 10px;
}
```

```
.hidden {
    display: none;
}
```

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
```

```
// Game state
let gameState = 'playing'; // 'playing' or 'gameOver'
let score = 0;
```

```
// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
```

```

    velocity: 0,
    gravity: 0.5,
    jumpStrength: -8,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
const pipeWidth = 60;
const pipeGap = 150;
const pipeSpeed = 3;
let pipeSpawnTimer = 0;
const pipeSpawnInterval = 120; // frames

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

```

```

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {

```

```

        bird.y = canvas.height - bird.height;
        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes
        pipeSpawnTimer;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();
            pipeSpawnTimer = 0;
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score;
                scoreElement.textContent = `Score: ${score}`;
            }
        });

        // Remove off-screen pipes
        while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
        {
            pipes.shift();
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState != 'playing') return;

```

```

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Reset game
function resetGame() {
    gameState = 'playing';
    score = 0;
    scoreElement.textContent = `Score: ${score}`;
    gameOverElement.classList.add('hidden');

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    pipes.length = 0;
    pipeSpawnTimer = 0;
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {

```

```
        resetGame();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Game loop
function gameLoop() {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    checkCollisions();

    drawPipes();
    drawBird();

    requestAnimationFrame(gameLoop);
}
```



```
// Start game
createPipe();
gameLoop();
```

Created a Flappy Bird game that runs in the browser. Here's what I've built:

Files Created:

1. **index.html** - Main HTML file with canvas and game UI
2. **style.css** - Styling
3. **game.js** - Game logic

Features:

- **Bird physics:** Gravity and jumping mechanics
- **Pipe obstacles:** Randomly generated pipes that move from right to left
- **Space key control:** Press SPACE to make the bird flap
- **Score tracking:** Score increases when you pass pipes
- **Collision detection:** Game ends when the bird hits pipes or boundaries
- **Game over & restart:** Press SPACE after game over to restart
- **Touch/click support:** Also works with mouse clicks and touch on mobile devices

How to Play:

1. Open index.html in your web browser
2. Press **SPACE** (or click/tap) to make the bird flap upward
3. Avoid the green pipes
4. Try to get the highest score possible!

The game includes a simple bird design with an eye and beak, and a sky/ground background. Open index.html in your browser to start playing.

User

I need the bird to move slower up and down and the gravity be less

Cursor

Adjusting bird physics: reducing gravity and jump strength to slow vertical movement.

Reducing gravity and jump strength to slow vertical movement:

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');

// Game state
let gameState = 'playing'; // 'playing' or 'gameOver'
let score = 0;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
const pipeWidth = 60;
const pipeGap = 150;
const pipeSpeed = 3;
let pipeSpawnTimer = 0;
const pipeSpawnInterval = 120; // frames

// Initialize first pipe
function createPipe() {
  const minHeight = 50;
  const maxHeight = canvas.height - pipeGap - minHeight;
```

```

    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

```

```

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes
        pipeSpawnTimer;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();

```

```

        pipeSpawnTimer = 0;
    }

    // Move pipes
    pipes.forEach(pipe => {
        pipe.x -= pipeSpeed;

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;
        }
    });

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

```

```
// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Reset game
function resetGame() {
    gameState = 'playing';
    score = 0;
    scoreElement.textContent = `Score: ${score}`;
    gameOverElement.classList.add('hidden');

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    pipes.length = 0;
    pipeSpawnTimer = 0;
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
```

```

        jump();
    });

    // Game loop
    function gameLoop() {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();

        requestAnimationFrame(gameLoop);
    }

    // Start game
    createPipe();
    gameLoop();

```

Adjusted the bird physics:

- **Gravity:** reduced from 0.5 to 0.25 (50% less)
- **Jump strength:** reduced from -8 to -5 (slower upward movement)

The bird now moves slower both up and down. Refresh your browser to see the changes. The movement should feel more controlled and gentle.

User

add some levels, where in the initial screen the user selects the level, and they have to complete the previous level to unlock the next

Cursor

Adding a level system with an initial level selection screen and unlock progression. Reviewing the current files:

Adding a level system with a level selection screen and unlock progression:

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

    <!-- Level Selection Screen -->
    <div id="levelSelection" class="level-selection">
      <h2>Select Level</h2>
      <div id="levelButtons" class="level-buttons"></div>
      <p class="level-instructions">Complete a level to
unlock the next one!</p>
    </div>

    <!-- Game Screen -->
    <div id="gameScreen" class="hidden">
      <canvas id="gameCanvas" width="400" height="600"></
canvas>

      <div class="instructions">
        <p>Press <strong>SPACE</strong> to flap and
```



```

avoid the pipes!</p>
    <p id="levelInfo">Level: 1</p>
    <p id="score">Score: 0</p>
    <p id="targetScore" class="hidden">Target: 0</
p>
    <p id="gameOver" class="hidden">Game Over!
Press SPACE to restart</p>
    <p id="levelComplete" class="hidden">Level
Complete! Press SPACE to continue</p>
    <button id="backToMenu" class="back-
button">Back to Menu</button>
    </div>
  </div>
</div>
<script src="game.js"></script>
</body>
</html>

```

```

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

body {
  font-family: 'Arial', sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.container {
  text-align: center;
  background: white;
  padding: 20px;
  border-radius: 15px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

```

```
}
```

```
h1 {  
    color: #333;  
    margin-bottom: 15px;  
    font-size: 2.5em;  
}
```

```
#gameCanvas {  
    border: 3px solid #333;  
    border-radius: 10px;  
    background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB  
70%, #90EE90 70%, #90EE90 100%);  
    display: block;  
    margin: 0 auto;  
}
```

```
.instructions {  
    margin-top: 15px;  
    color: #555;  
}
```

```
.instructions p {  
    margin: 8px 0;  
    font-size: 1.1em;  
}
```

```
#score {  
    font-size: 1.5em;  
    font-weight: bold;  
    color: #667eea;  
}
```

```
#gameOver {  
    font-size: 1.3em;  
    font-weight: bold;  
    color: #e74c3c;  
    margin-top: 10px;  
}
```

```
.hidden {  
    display: none;  
}
```

```
/* Level Selection Styles */
.level-selection {
    padding: 20px;
}

.level-selection h2 {
    color: #333;
    margin-bottom: 20px;
    font-size: 2em;
}

.level-buttons {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px,
1fr));
    gap: 15px;
    margin-bottom: 20px;
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}

.level-button {
    padding: 20px;
    font-size: 1.5em;
    font-weight: bold;
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    color: white;
    cursor: pointer;
    transition: all 0.3s;
}

.level-button:hover:not(.locked) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}

.level-button.locked {
    background: #ccc;
    color: #888;
}
```

```
    cursor: not-allowed;
    opacity: 0.6;
}

.level-button.completed {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d
100%);
}

.level-instructions {
    color: #666;
    font-size: 1em;
    margin-top: 15px;
}

#levelInfo {
    font-size: 1.3em;
    font-weight: bold;
    color: #764ba2;
}

#targetScore {
    font-size: 1.2em;
    color: #e67e22;
}

#levelComplete {
    font-size: 1.3em;
    font-weight: bold;
    color: #27ae60;
    margin-top: 10px;
}

.back-button {
    margin-top: 15px;
    padding: 10px 20px;
    font-size: 1em;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
}
```

```
.back-button:hover {  
    background: #5568d3;  
}
```

Rewriting the game logic to add the level system:

```
// Canvas setup  
const canvas = document.getElementById('gameCanvas');  
const ctx = canvas.getContext('2d');  
const scoreElement = document.getElementById('score');  
const gameOverElement = document.getElementById('gameOver');  
const levelCompleteElement =  
document.getElementById('levelComplete');  
const levelInfoElement = document.getElementById('levelInfo');  
const targetScoreElement =  
document.getElementById('targetScore');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');  
  
// Level definitions  
const levels = [  
    {  
        name: "Level 1",  
        pipeSpeed: 2,  
        pipeGap: 180,  
        pipeSpawnInterval: 140,  
        targetScore: 5,  
        description: "Easy – Get 5 points"  
    },  
    {  
        name: "Level 2",  
        pipeSpeed: 2.5,  
        pipeGap: 160,
```

```

        pipeSpawnInterval: 130,
        targetScore: 10,
        description: "Medium – Get 10 points"
    },
    {
        name: "Level 3",
        pipeSpeed: 3,
        pipeGap: 150,
        pipeSpawnInterval: 120,
        targetScore: 15,
        description: "Hard – Get 15 points"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        targetScore: 20,
        description: "Very Hard – Get 20 points"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        targetScore: 25,
        description: "Expert – Get 25 points"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        targetScore: 30,
        description: "Master – Get 30 points"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

```

```

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let targetScore = 0;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

```

```

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

```



```

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    targetScore = currentLevel.targetScore;

    // Reset game state
    gameState = 'playing';
    score = 0;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    targetScoreElement.textContent = `Target: ${targetScore}`;
    targetScoreElement.classList.remove('hidden');
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

    pipes.push({

```

```

        x: canvas.width,
        topHeight: topHeight,
        bottomY: topHeight + pipeGap,
        passed: false
    });
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

```

```

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes
        pipeSpawnTimer;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();
            pipeSpawnTimer = 0;
        }

        // Move pipes
        pipes.forEach(pipe => {

```

```

    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;

        // Check if level is complete
        if (score >= targetScore) {
            completeLevel();
        }
    }
});

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

```

```
// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}
```

```

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}

```

```

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

Added a level system to your Flappy Bird game. Here's what's included:

Features Added:

1. **Level Selection Screen:** Appears first when you open the game, showing all available levels.
2. **6 Levels** with increasing difficulty:
 - **Level 1:** Easy (5 points target, slower pipes, larger gaps)
 - **Level 2:** Medium (10 points)
 - **Level 3:** Hard (15 points)
 - **Level 4:** Very Hard (20 points)
 - **Level 5:** Expert (25 points)

- **Level 6:** Master (30 points)

3. **Unlock System:**

- Level 1 is unlocked by default
- Complete a level to unlock the next one
- Progress is saved in localStorage (persists after closing the browser)

4. **Level Completion:**

- Each level has a target score
- When you reach the target, the level completes
- Completed levels are marked with a green button
- The next level unlocks automatically

5. **Visual Indicators:**

- Locked levels are grayed out
- Completed levels have a green background
- Current level and target score are displayed during gameplay

6. **Navigation:**

- "Back to Menu" button to return to level selection
- After completing a level, press SPACE to return to the menu

The game now starts with the level selection screen. Complete each level to unlock the next one. Your progress is saved automatically.

User

Add an end to each level based on the increasing difficulty of each level

Cursor

Adding a finite end to each level: each level requires passing a set number of pipes that increases with difficulty. Reviewing the current code:

Adding a finite end to each level: each level requires passing a set number of pipes that increases with difficulty. Updating the code:

Updating the code to add a finite end to each level:


```

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

        <!-- Level Selection Screen -->
        <div id="levelSelection" class="level-selection">
            <h2>Select Level</h2>
            <div id="levelButtons" class="level-buttons"></div>
            <p class="level-instructions">Complete a level to
unlock the next one!</p>
        </div>

        <!-- Game Screen -->
        <div id="gameScreen" class="hidden">
            <canvas id="gameCanvas" width="400" height="600"></
canvas>

            <div class="instructions">
                <p>Press <strong>SPACE</strong> to flap and
avoid the pipes!</p>
                <p id="levelInfo">Level: 1</p>
                <p id="progress">Progress: 0 / 0</p>
                <p id="score">Score: 0</p>
                <p id="gameOver" class="hidden">Game Over!
Press SPACE to restart</p>
                <p id="levelComplete" class="hidden">Level
Complete! Press SPACE to continue</p>
                <button id="backToMenu" class="back-
button">Back to Menu</button>
            </div>
        </div>
    </div>
    <script src="game.js"></script>
</body>

```

```
</html>
```

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const targetScoreElement =
document.getElementById('targetScore');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
```

```

        pipeGap: 150,
        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let targetScore = 0;

```

```

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
  const completed = getCompletedLevels();
  if (!completed.includes(levelIndex)) {
    completed.push(levelIndex);
    localStorage.setItem('completedLevels',
JSON.stringify(completed));
  }
}

```

```

    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;

```

```

pipeGap = currentLevel.pipeGap;
pipeSpawnInterval = currentLevel.pipeSpawnInterval;
targetScore = currentLevel.targetScore;

// Reset game state
gameState = 'playing';
score = 0;
scoreElement.textContent = `Score: ${score}`;
levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
targetScoreElement.textContent = `Target: ${targetScore}`;
targetScoreElement.classList.remove('hidden');
gameOverElement.classList.add('hidden');
levelCompleteElement.classList.add('hidden');

// Reset bird
bird.y = canvas.height / 2;
bird.velocity = 0;

// Reset pipes
pipes.length = 0;
pipeSpawnTimer = 0;
createPipe();

// Show game screen, hide level selection
levelSelectionElement.classList.add('hidden');
gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
  const minHeight = 50;
  const maxHeight = canvas.height - pipeGap - minHeight;
  const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw bird

```

```

function drawBird() {
  ctx.fillStyle = bird.color;
  ctx.beginPath();
  ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,
bird.width/2, 0, Math.PI * 2);
  ctx.fill();

  // Draw eye
  ctx.fillStyle = 'white';
  ctx.beginPath();
  ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
  ctx.fill();

  ctx.fillStyle = 'black';
  ctx.beginPath();
  ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
  ctx.fill();

  // Draw beak
  ctx.fillStyle = '#FF8C00';
  ctx.beginPath();
  ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
  ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
  ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
  ctx.closePath();
  ctx.fill();
}

// Draw pipes
function drawPipes() {
  ctx.fillStyle = '#228B22';

  pipes.forEach(pipe => {
    // Top pipe
    ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
    ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

    // Bottom pipe
    ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,

```

```

canvas.height - pipe.bottomY);
    ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

```

// Update bird

```

function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

```

// Update pipes

```

function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes
        pipeSpawnTimer;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();
            pipeSpawnTimer = 0;
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score;
                scoreElement.textContent = `Score: ${score}`;
            }
        });
    }
}

```



```

        // Check if level is complete
        if (score >= targetScore) {
            completeLevel();
        }
    }
});

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

```

```

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels

```

```

}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
    }
}

```

```

        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
    {
        name: "Level 1",
        pipeSpeed: 2,
        pipeGap: 180,

```

```
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy – Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium – Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,  
    description: "Hard – Pass 12 pipes"  
  },  
  {  
    name: "Level 4",  
    pipeSpeed: 3.5,  
    pipeGap: 140,  
    pipeSpawnInterval: 110,  
    totalPipes: 18,  
    description: "Very Hard – Pass 18 pipes"  
  },  
  {  
    name: "Level 5",  
    pipeSpeed: 4,  
    pipeGap: 130,  
    pipeSpawnInterval: 100,  
    totalPipes: 25,  
    description: "Expert – Pass 25 pipes"  
  },  
  {  
    name: "Level 6",  
    pipeSpeed: 4.5,  
    pipeGap: 120,  
    pipeSpawnInterval: 90,  
    totalPipes: 35,  
    description: "Master – Pass 35 pipes"  
  }  
}
```

```

];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let targetScore = 0;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
  }
}

```

```

        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {

```

```

        button.addEventListener('click', () =>
startLevel(index));
    }

    levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    targetScore = currentLevel.targetScore;

    // Reset game state
    gameState = 'playing';
    score = 0;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    targetScoreElement.textContent = `Target: ${targetScore}`;
    targetScoreElement.classList.remove('hidden');
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe

```



```

function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();

```

```

    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes
        pipeSpawnTimer;
    }
}

```

```

    if (pipeSpawnTimer >= pipeSpawnInterval) {
        createPipe();
        pipeSpawnTimer = 0;
    }

    // Move pipes
    pipes.forEach(pipe => {
        pipe.x -= pipeSpeed;

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;

            // Check if level is complete
            if (score >= targetScore) {
                completeLevel();
            }
        }
    });

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&

```

```

        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;
    }
}

```

```

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

```

```

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');

```

```
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard – Pass 18 pipes"
  },
  {
    name: "Level 5",
```

```

        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

```



```

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

```

```

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    targetScore = currentLevel.targetScore;

    // Reset game state
    gameState = 'playing';
    score = 0;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    targetScoreElement.textContent = `Target: ${targetScore}`;
    targetScoreElement.classList.remove('hidden');
}

```

```

gameOverElement.classList.add('hidden');
levelCompleteElement.classList.add('hidden');

// Reset bird
bird.y = canvas.height / 2;
bird.velocity = 0;

// Reset pipes
pipes.length = 0;
pipeSpawnTimer = 0;
createPipe();

// Show game screen, hide level selection
levelSelectionElement.classList.add('hidden');
gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -

```

```

5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;
    }
}

```

```

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
        bird.y = canvas.height - bird.height;
        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes
        pipeSpawnTimer++;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();
            pipeSpawnTimer = 0;
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score++;
                scoreElement.textContent = `Score: ${score}`;

                // Check if level is complete
                if (score >= targetScore) {
                    completeLevel();
                }
            }
        });

        // Remove off-screen pipes
        while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

```

```

    }
  }
}

// Collision detection
function checkCollisions() {
  if (gameState !== 'playing') return;

  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
      gameOver();
    }

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
      gameOver();
    }
  });
}

// Game over
function gameOver() {
  if (gameState === 'playing') {
    gameState = 'gameOver';
    gameOverElement.classList.remove('hidden');
  }
}

// Complete level
function completeLevel() {
  if (gameState === 'playing') {
    gameState = 'levelComplete';
    levelCompleteElement.classList.remove('hidden');
    gameOverElement.classList.add('hidden');

    // Mark level as completed
    markLevelCompleted(currentLevelIndex);

    // Unlock next level
  }
}

```

```

        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

```

```

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }
}

```



```

        requestAnimationFrame(gameLoop);
    }

    // Initialize game
    initLevelSelection();
    gameLoop();

    // Canvas setup
    const canvas = document.getElementById('gameCanvas');
    const ctx = canvas.getContext('2d');
    const scoreElement = document.getElementById('score');
    const gameOverElement = document.getElementById('gameOver');
    const levelCompleteElement =
    document.getElementById('levelComplete');
    const levelInfoElement = document.getElementById('levelInfo');
    const progressElement = document.getElementById('progress');
    const levelSelectionElement =
    document.getElementById('levelSelection');
    const gameScreenElement =
    document.getElementById('gameScreen');
    const levelButtonsElement =
    document.getElementById('levelButtons');
    const backToMenuButton = document.getElementById('backToMenu');

    // Level definitions
    const levels = [
        {
            name: "Level 1",
            pipeSpeed: 2,
            pipeGap: 180,
            pipeSpawnInterval: 140,
            totalPipes: 5,
            description: "Easy - Pass 5 pipes"
        },
        {
            name: "Level 2",
            pipeSpeed: 2.5,
            pipeGap: 160,
            pipeSpawnInterval: 130,
            totalPipes: 8,
            description: "Medium - Pass 8 pipes"
        },
    ],

```

```

    {
      name: "Level 3",
      pipeSpeed: 3,
      pipeGap: 150,
      pipeSpawnInterval: 120,
      totalPipes: 12,
      description: "Hard – Pass 12 pipes"
    },
    {
      name: "Level 4",
      pipeSpeed: 3.5,
      pipeGap: 140,
      pipeSpawnInterval: 110,
      totalPipes: 18,
      description: "Very Hard – Pass 18 pipes"
    },
    {
      name: "Level 5",
      pipeSpeed: 4,
      pipeGap: 130,
      pipeSpawnInterval: 100,
      totalPipes: 25,
      description: "Expert – Pass 25 pipes"
    },
    {
      name: "Level 6",
      pipeSpeed: 4.5,
      pipeGap: 120,
      pipeSpawnInterval: 90,
      totalPipes: 35,
      description: "Master – Pass 35 pipes"
    }
  ];

  // Game state
  let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
  'levelComplete'
  let score = 0;
  let currentLevel = null;
  let currentLevelIndex = 0;

  // Level settings (will be set based on selected level)
  let pipeWidth = 60;
  let pipeGap = 150;

```

```

let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {

```

```

        const completed = getCompletedLevels();
        if (!completed.includes(levelIndex)) {
            completed.push(levelIndex);
            localStorage.setItem('completedLevels',
JSON.stringify(completed));
        }
    }

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {

```

```

currentLevelIndex = levelIndex;
currentLevel = levels[levelIndex];

// Set level-specific settings
pipeSpeed = currentLevel.pipeSpeed;
pipeGap = currentLevel.pipeGap;
pipeSpawnInterval = currentLevel.pipeSpawnInterval;
totalPipes = currentLevel.totalPipes;

// Reset game state
gameState = 'playing';
score = 0;
pipesSpawned = 0;
finishLineX = null;
scoreElement.textContent = `Score: ${score}`;
levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
gameOverElement.classList.add('hidden');
levelCompleteElement.classList.add('hidden');

// Reset bird
bird.y = canvas.height / 2;
bird.velocity = 0;

// Reset pipes
pipes.length = 0;
pipeSpawnTimer = 0;
createPipe();
pipesSpawned = 1;

// Show game screen, hide level selection
levelSelectionElement.classList.add('hidden');
gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
  const minHeight = 50;
  const maxHeight = canvas.height - pipeGap - minHeight;
  const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

  pipes.push({

```

```

        x: canvas.width,
        topHeight: topHeight,
        bottomY: topHeight + pipeGap,
        passed: false
    });
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

```

```

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes
        pipeSpawnTimer++;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();
            pipeSpawnTimer = 0;
        }

        // Move pipes
        pipes.forEach(pipe => {

```

```

    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;

        // Check if level is complete
        if (score >= targetScore) {
            completeLevel();
        }
    }
});

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

```



```

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}

```

```

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}

```

```

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
```

```
const backButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard - Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert - Pass 25 pipes"
  },
]
```

```

    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {

```

```

        const unlocked = localStorage.getItem('unlockedLevels');
        return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
    }

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
    });
}

```

```

        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

```

```

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';

```



```

    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
    });
}

```

```

        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes
        pipeSpawnTimer;
        if (pipeSpawnTimer >= pipeSpawnInterval) {
            createPipe();
            pipeSpawnTimer = 0;
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe

```

```

        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;

            // Check if level is complete
            if (score >= targetScore) {
                completeLevel();
            }
        }
    });

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {

```

```

        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';

```

```

        levelSelectionElement.classList.remove('hidden');
        gameScreenElement.classList.add('hidden');
        initLevelSelection(); // Refresh to show newly unlocked
levels
    }

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);

```

```

        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [

```

```
{
  name: "Level 1",
  pipeSpeed: 2,
  pipeGap: 180,
  pipeSpawnInterval: 140,
  totalPipes: 5,
  description: "Easy – Pass 5 pipes"
},
{
  name: "Level 2",
  pipeSpeed: 2.5,
  pipeGap: 160,
  pipeSpawnInterval: 130,
  totalPipes: 8,
  description: "Medium – Pass 8 pipes"
},
{
  name: "Level 3",
  pipeSpeed: 3,
  pipeGap: 150,
  pipeSpawnInterval: 120,
  totalPipes: 12,
  description: "Hard – Pass 12 pipes"
},
{
  name: "Level 4",
  pipeSpeed: 3.5,
  pipeGap: 140,
  pipeSpawnInterval: 110,
  totalPipes: 18,
  description: "Very Hard – Pass 18 pipes"
},
{
  name: "Level 5",
  pipeSpeed: 4,
  pipeGap: 130,
  pipeSpawnInterval: 100,
  totalPipes: 25,
  description: "Expert – Pass 25 pipes"
},
{
  name: "Level 6",
  pipeSpeed: 4.5,
  pipeGap: 120,
```

```

        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

```



```

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {

```

```

        button.classList.add('locked');
    } else if (isLevelCompleted(index)) {
        button.classList.add('completed');
    }

    if (isLevelUnlocked(index)) {
        button.addEventListener('click', () =>
startLevel(index));
    }

    levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;

```

```

    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

```

```

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);
    });
}

```

```

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;

                // If this is the last pipe, set finish line
position
                if (pipesSpawned = totalPipes) {
                    // Set finish line after the last pipe
passes
                    finishLineX = canvas.width + 200;
                }
            }
        }
    }
}

```

```

    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line
if (finishLineX != null) {
    finishLineX -= pipeSpeed;

    // Check if bird reached finish line
    if (bird.x >= finishLineX && score >= totalPipes) {
        completeLevel();
    }
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

}

// Collision detection
function checkCollisions() {
    if (gameState != 'playing') return;

```

```

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState === 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {

```

```

    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        scoreElement.textContent = `Score: ${score}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

```



```

// Mouse/touch input
canvas.addEventListener('click',_jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click',_returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  }
]

```

```

    },
    {
      name: "Level 4",
      pipeSpeed: 3.5,
      pipeGap: 140,
      pipeSpawnInterval: 110,
      totalPipes: 18,
      description: "Very Hard - Pass 18 pipes"
    },
    {
      name: "Level 5",
      pipeSpeed: 4,
      pipeGap: 130,
      pipeSpawnInterval: 100,
      totalPipes: 25,
      description: "Expert - Pass 25 pipes"
    },
    {
      name: "Level 6",
      pipeSpeed: 4.5,
      pipeGap: 120,
      pipeSpawnInterval: 90,
      totalPipes: 35,
      description: "Master - Pass 35 pipes"
    }
  ];

  // Game state
  let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
  'levelComplete'
  let score = 0;
  let currentLevel = null;
  let currentLevelIndex = 0;

  // Level settings (will be set based on selected level)
  let pipeWidth = 60;
  let pipeGap = 150;
  let pipeSpeed = 3;
  let pipeSpawnInterval = 120;
  let totalPipes = 0;
  let pipesSpawned = 0;
  let finishLineX = null;

  // Bird properties

```

```

const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
  levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
    JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
  const completed = getCompletedLevels();
  if (!completed.includes(levelIndex)) {
    completed.push(levelIndex);
    localStorage.setItem('completedLevels',
    JSON.stringify(completed));
  }
}

```

```

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;

```

```

    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

```

```

// Draw finish line
function drawFinishLine() {
    if (finishLineX == null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak

```

```

    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

```



```

    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;

                // If this is the last pipe, set finish line
                position
                if (pipesSpawned === totalPipes) {
                    // Set finish line after the last pipe
                    passes
                    finishLineX = canvas.width + 200;
                }
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score;
                scoreElement.textContent = `Score: ${score}`;
                progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

                // Check if all pipes are passed
                if (score >= totalPipes) {
                    completeLevel();
                }
            }
        });

        // Move finish line

```

```

    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;

        // Check if bird reached finish line
        if (bird.x >= finishLineX && score >= totalPipes) {
            completeLevel();
        }
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

```

```

}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu

```

```

function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background

```

```

        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard – Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert – Pass 25 pipes"
  },
  {
```

```

        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');

```

```

        return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
    }

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
    });
}

```



```

        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

```

```

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
}

```

```

    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth

```

```

+ 10, 20);

    // Bottom pipe
    ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
    ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;

                // If this is the last pipe, set finish line
position
                if (pipesSpawned = totalPipes) {
                    // Set finish line after the last pipe
passes

```

```

        finishLineX = canvas.width + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line
if (finishLineX != null) {
    finishLineX -= pipeSpeed;

    // Check if bird reached finish line
    if (bird.x >= finishLineX && score >= totalPipes) {
        completeLevel();
    }
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

}

// Collision detection
function checkCollisions() {

```

```

    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState === 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

```

```

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

```

```

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', _jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }
}

```



```
    requestAnimationFrame(gameLoop);  
}
```

```
// Initialize game  
initLevelSelection();  
gameLoop();
```

Fixing finish line logic: it should appear after all pipes are passed, not when the last pipe spawns.

```
// Canvas setup  
const canvas = document.getElementById('gameCanvas');  
const ctx = canvas.getContext('2d');  
const scoreElement = document.getElementById('score');  
const gameOverElement = document.getElementById('gameOver');  
const levelCompleteElement =  
document.getElementById('levelComplete');  
const levelInfoElement = document.getElementById('levelInfo');  
const progressElement = document.getElementById('progress');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions  
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy - Pass 5 pipes"  
  },  
  {  
    name: "Level 2",
```

```

        pipeSpeed: 2.5,
        pipeGap: 160,
        pipeSpawnInterval: 130,
        totalPipes: 8,
        description: "Medium – Pass 8 pipes"
    },
    {
        name: "Level 3",
        pipeSpeed: 3,
        pipeGap: 150,
        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;

```

```

let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
  levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
    JSON.stringify(unlocked));
  }
}

```

```

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }
    }
}

```

```

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
```

```
const minHeight = 50;  
const maxHeight = canvas.height - pipeGap - minHeight;  
const topHeight = Math.random() * (maxHeight - minHeight) +  
minHeight;
```

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

```
// Draw finish line
```

```
function drawFinishLine() {  
  if (finishLineX = null) return;  
  
  ctx.strokeStyle = '#FFD700';  
  ctx.lineWidth = 5;  
  ctx.setLineDash([10, 10]);  
  ctx.beginPath();  
  ctx.moveTo(finishLineX, 0);  
  ctx.lineTo(finishLineX, canvas.height);  
  ctx.stroke();  
  ctx.setLineDash([]);
```

```
  // Draw "FINISH" text  
  ctx.fillStyle = '#FFD700';  
  ctx.font = 'bold 24px Arial';  
  ctx.textAlign = 'center';  
  ctx.fillText('FINISH', finishLineX, canvas.height / 2);  
  ctx.textAlign = 'left';  
}
```

```
// Draw bird
```

```
function drawBird() {  
  ctx.fillStyle = bird.color;  
  ctx.beginPath();  
  ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,  
bird.width/2, 0, Math.PI * 2);  
  ctx.fill();
```

```
  // Draw eye  
  ctx.fillStyle = 'white';
```

```

    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {

```

```

    bird.velocity += bird.gravity;
    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
        bird.y = canvas.height - bird.height;
        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX === null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = 0;
            pipes.forEach(pipe => {
                if (pipe.x + pipeWidth > rightmostX) {
                    rightmostX = pipe.x + pipeWidth;
                }
            });
            finishLineX = rightmostX + 150;
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

```



```

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;
            progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

            // Check if all pipes are passed
            if (score >= totalPipes) {
                completeLevel();
            }
        }
    });

    // Move finish line
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;

        // Check if bird reached finish line
        if (bird.x >= finishLineX && score >= totalPipes) {
            completeLevel();
        }
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }
    });
}

```

```

    }

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
        gameOver();
    }
});
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
    }
}

```

```

        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

```

```
// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();
```

Updating the CSS to style the progress indicator:

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

body {
  font-family: 'Arial', sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.container {
  text-align: center;
  background: white;
  padding: 20px;
  border-radius: 15px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

h1 {
  color: #333;
  margin-bottom: 15px;
  font-size: 2.5em;
}

#gameCanvas {
  border: 3px solid #333;
  border-radius: 10px;
  background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB
70%, #90EE90 70%, #90EE90 100%);
  display: block;
```

```
        margin: 0 auto;
    }

    .instructions {
        margin-top: 15px;
        color: #555;
    }

    .instructions p {
        margin: 8px 0;
        font-size: 1.1em;
    }

    #score {
        font-size: 1.5em;
        font-weight: bold;
        color: #667eea;
    }

    #gameOver {
        font-size: 1.3em;
        font-weight: bold;
        color: #e74c3c;
        margin-top: 10px;
    }

    .hidden {
        display: none;
    }

    /* Level Selection Styles */
    .level-selection {
        padding: 20px;
    }

    .level-selection h2 {
        color: #333;
        margin-bottom: 20px;
        font-size: 2em;
    }

    .level-buttons {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(120px,
```

```
1fr));
    gap: 15px;
    margin-bottom: 20px;
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}

.level-button {
    padding: 20px;
    font-size: 1.5em;
    font-weight: bold;
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    color: white;
    cursor: pointer;
    transition: all 0.3s;
}

.level-button:hover:not(.locked) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}

.level-button.locked {
    background: #ccc;
    color: #888;
    cursor: not-allowed;
    opacity: 0.6;
}

.level-button.completed {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d
100%);
}

.level-instructions {
    color: #666;
    font-size: 1em;
    margin-top: 15px;
}
```

```
#levelInfo {
    font-size: 1.3em;
    font-weight: bold;
    color: #764ba2;
}

#progress {
    font-size: 1.2em;
    font-weight: bold;
    color: #e67e22;
}

#levelComplete {
    font-size: 1.3em;
    font-weight: bold;
    color: #27ae60;
    margin-top: 10px;
}

.back-button {
    margin-top: 15px;
    padding: 10px 20px;
    font-size: 1em;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
}

.back-button:hover {
    background: #5568d3;
}
```

Refining finish line logic to handle edge cases:

```
// Canvas setup
```



```
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
```

```

        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,

```

```

    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');

```

```

        return completed ? JSON.parse(completed) : [];
    }

    function isLevelCompleted(levelIndex) {
        return getCompletedLevels().includes(levelIndex);
    }

    // Initialize level selection screen
    function initLevelSelection() {
        levelButtonsElement.innerHTML = '';

        levels.forEach((level, index) => {
            const button = document.createElement('button');
            button.className = 'level-button';
            button.textContent = index + 1;
            button.title = level.description;

            if (!isLevelUnlocked(index)) {
                button.classList.add('locked');
            } else if (isLevelCompleted(index)) {
                button.classList.add('completed');
            }

            if (isLevelUnlocked(index)) {
                button.addEventListener('click', () =>
startLevel(index));
            }

            levelButtonsElement.appendChild(button);
        });
    }

    // Start a level
    function startLevel(levelIndex) {
        currentLevelIndex = levelIndex;
        currentLevel = levels[levelIndex];

        // Set level-specific settings
        pipeSpeed = currentLevel.pipeSpeed;
        pipeGap = currentLevel.pipeGap;
        pipeSpawnInterval = currentLevel.pipeSpawnInterval;
        totalPipes = currentLevel.totalPipes;

        // Reset game state
    }

```

```

    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
{totalPipes}}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

```

```

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2

```

```

- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes

```

```

function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score;
                scoreElement.textContent = `Score: ${score}`;
                progressElement.textContent = `Progress: $
                {score} / ${totalPipes}`;
            }
        });
    }
}

```



```

        // Check if all pipes are passed
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line
if (finishLineX != null) {
    finishLineX -= pipeSpeed;

    // Check if bird reached finish line
    if (bird.x >= finishLineX && score >= totalPipes) {
        completeLevel();
    }
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState != 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
};

```

```

}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;
    }
}

```

```

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', _jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

```

```

// Game loop
function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    checkCollisions();

    drawPipes();
    drawFinishLine();
    drawBird();
  }

  requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

Completing the level immediately when all pipes are passed, with the finish line as a visual indicator:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');

```

```
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backButton = document.getElementById('backToMenu');
```

```
// Level definitions
```

```
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
```

```

        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard - Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert - Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,

```

```

    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

```

```

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;

```



```

    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';

```

```

    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2

```

```

+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {

```

```

// Spawn new pipes until we reach the total
if (pipesSpawned < totalPipes) {
    pipeSpawnTimer;
    if (pipeSpawnTimer >= pipeSpawnInterval) {
        createPipe();
        pipesSpawned;
        pipeSpawnTimer = 0;
    }
}

// Show finish line when all pipes are passed
if (score >= totalPipes && finishLineX = null) {
    // Find the rightmost pipe position and place
    finish line after it
    let rightmostX = bird.x + 200; // Default position
    ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
    bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
        level immediately

```

```

        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX ! null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
    }
}

```

```

        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

```

```

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}

```

```

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

Changes Made:

1. **Finite pipe count per level** (increasing with difficulty):
 - Level 1: 5 pipes
 - Level 2: 8 pipes
 - Level 3: 12 pipes
 - Level 4: 18 pipes
 - Level 5: 25 pipes
 - Level 6: 35 pipes
2. **Pipe spawning stops** once the level's total is reached.

3. **Progress indicator:** Shows "Progress: X / Y" to track how many pipes you've passed.
4. **Finish line:** A gold dashed line with "FINISH" appears after all pipes are passed (visual indicator).
5. **Level completion:** The level completes immediately when you pass all required pipes.

Each level now has a clear end goal. The number of pipes increases with difficulty, making later levels longer and more challenging. Refresh your browser to see the changes.

User

add a coins feature that you have to pick up as you progress through the game the coins stay registered even between levels

Cursor

Adding a coin system: coins appear during gameplay, can be collected, and persist between levels. Reviewing the current code:

Adding the coin system:

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

    <!-- Level Selection Screen -->
```

```

        <div id="levelSelection" class="level-selection">
            <h2>Select Level</h2>
            <div id="coinDisplay" class="coin-display">💰
Coins: <span id="totalCoins">0</span></div>
            <div id="levelButtons" class="level-buttons"></div>
            <p class="level-instructions">Complete a level to
unlock the next one!</p>
        </div>

        <!-- Game Screen -->
        <div id="gameScreen" class="hidden">
            <canvas id="gameCanvas" width="400" height="600"></
canvas>

            <div class="instructions">
                <p>Press <strong>SPACE</strong> to flap and
avoid the pipes!</p>
                <p id="levelInfo">Level: 1</p>
                <p id="progress">Progress: 0 / 0</p>
                <p id="score">Score: 0</p>
                <p id="gameOver" class="hidden">Game Over!
Press SPACE to restart</p>
                <p id="levelComplete" class="hidden">Level
Complete! Press SPACE to continue</p>
                <button id="backToMenu" class="back-
button">Back to Menu</button>
            </div>
        </div>
    </div>
    <script src="game.js"></script>
</body>
</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</title>
    <link rel="stylesheet" href="style.css">
</head>

```

```

<body>
  <div class="container">
    <h1>Flappy Bird</h1>

    <!-- Level Selection Screen -->
    <div id="levelSelection" class="level-selection">
      <h2>Select Level</h2>
      <div id="coinDisplay" class="coin-display">💰
Coins: <span id="totalCoins">0</span></div>
      <div id="levelButtons" class="level-buttons"></div>
      <p class="level-instructions">Complete a level to
unlock the next one!</p>
    </div>

    <!-- Game Screen -->
    <div id="gameScreen" class="hidden">
      <canvas id="gameCanvas" width="400" height="600"></
canvas>
      <div class="instructions">
        <p>Press <strong>SPACE</strong> to flap and
avoid the pipes!</p>
        <p id="levelInfo">Level: 1</p>
        <p id="progress">Progress: 0 / 0</p>
        <p id="score">Score: 0</p>
        <p id="coins">💰 Coins: 0</p>
        <p id="gameOver" class="hidden">Game Over!
Press SPACE to restart</p>
        <p id="levelComplete" class="hidden">Level
Complete! Press SPACE to continue</p>
        <button id="backToMenu" class="back-
button">Back to Menu</button>
      </div>
    </div>
  </div>
  <script src="game.js"></script>
</body>
</html>

```

Adding coin functionality to the game:

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
```

```

        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;

```

```

let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
  const completed = getCompletedLevels();
  if (!completed.includes(levelIndex)) {
    completed.push(levelIndex);
    localStorage.setItem('completedLevels',

```

```

JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings

```

```

    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

```



```

    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();
}

```

```

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;

```

```

        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

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

```

```

        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

```

```

    }
  });
}

// Game over
function gameOver() {
  if (gameState = 'playing') {
    gameState = 'gameOver';
    gameOverElement.classList.remove('hidden');
  }
}

// Complete level
function completeLevel() {
  if (gameState = 'playing') {
    gameState = 'levelComplete';
    levelCompleteElement.classList.remove('hidden');
    gameOverElement.classList.add('hidden');

    // Mark level as completed
    markLevelCompleted(currentLevelIndex);

    // Unlock next level
    if (currentLevelIndex + 1 < levels.length) {
      unlockLevel(currentLevelIndex + 1);
    }
  }
}

// Reset game
function resetGame() {
  if (gameState = 'gameOver' || gameState = 'levelComplete')
  {
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    bird.y = canvas.height / 2;
  }
}

```

```

        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', _jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

```

```

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');

```

```
const levelCompleteElement =  
document.getElementById('levelComplete');  
const levelInfoElement = document.getElementById('levelInfo');  
const progressElement = document.getElementById('progress');  
const coinsElement = document.getElementById('coins');  
const totalCoinsElement =  
document.getElementById('totalCoins');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
```

```
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy – Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium – Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,  
    description: "Hard – Pass 12 pipes"  
  },  
  {  
    name: "Level 4",  
    pipeSpeed: 3.5,
```



```

        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard - Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert - Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,

```

```

    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

```

```

    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;

```

```

    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

```

```
}
```

```
// Draw finish line
```

```
function drawFinishLine() {
```

```
    if (finishLineX = null) return;
```

```
    ctx.strokeStyle = '#FFD700';
```

```
    ctx.lineWidth = 5;
```

```
    ctx.setLineDash([10, 10]);
```

```
    ctx.beginPath();
```

```
    ctx.moveTo(finishLineX, 0);
```

```
    ctx.lineTo(finishLineX, canvas.height);
```

```
    ctx.stroke();
```

```
    ctx.setLineDash([]);
```

```
    // Draw "FINISH" text
```

```
    ctx.fillStyle = '#FFD700';
```

```
    ctx.font = 'bold 24px Arial';
```

```
    ctx.textAlign = 'center';
```

```
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
```

```
    ctx.textAlign = 'left';
```

```
}
```

```
// Draw bird
```

```
function drawBird() {
```

```
    ctx.fillStyle = bird.color;
```

```
    ctx.beginPath();
```

```
    ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,  
bird.width/2, 0, Math.PI * 2);
```

```
    ctx.fill();
```

```
    // Draw eye
```

```
    ctx.fillStyle = 'white';
```

```
    ctx.beginPath();
```

```
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -  
5, 5, 0, Math.PI * 2);
```

```
    ctx.fill();
```

```
    ctx.fillStyle = 'black';
```

```
    ctx.beginPath();
```

```
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -  
5, 3, 0, Math.PI * 2);
```

```
    ctx.fill();
```

```

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;

```

```

    }
  }
}

```

```

// Update pipes
function updatePipes() {
  if (gameState === 'playing') {
    // Spawn new pipes until we reach the total
    if (pipesSpawned < totalPipes) {
      pipeSpawnTimer;
      if (pipeSpawnTimer >= pipeSpawnInterval) {
        createPipe();
        pipesSpawned++;
        pipeSpawnTimer = 0;
      }
    }

    // Show finish line when all pipes are passed
    if (score >= totalPipes && finishLineX === null) {
      // Find the rightmost pipe position and place
      finish line after it
      let rightmostX = bird.x + 200; // Default position
      ahead of bird
      if (pipes.length > 0) {
        pipes.forEach(pipe => {
          if (pipe.x + pipeWidth > rightmostX) {
            rightmostX = pipe.x + pipeWidth;
          }
        });
        finishLineX = rightmostX + 150;
      } else {
        // If no pipes left, place finish line ahead of
        bird
        finishLineX = bird.x + 200;
      }
    }

    // Move pipes
    pipes.forEach(pipe => {
      pipe.x -= pipeSpeed;

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

```

```

        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX ! null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    })
}

```



```

    });
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;
    }
}

```

```

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button

```

```

backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    checkCollisions();

    drawPipes();
    drawFinishLine();
    drawBird();
  }

  requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement = document.getElementById('levelComplete');

```

```
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
```

```

        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,

```

```

    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

```

```

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {

```

```

        button.classList.add('locked');
    } else if (isLevelCompleted(index)) {
        button.classList.add('completed');
    }

    if (isLevelUnlocked(index)) {
        button.addEventListener('click', () =>
startLevel(index));
    }

    levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;

```



```

    createPipe();
    pipesSpawned = 1;

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

```

```

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

  // Draw eye
  ctx.fillStyle = 'white';
  ctx.beginPath();
  ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
  ctx.fill();

  ctx.fillStyle = 'black';
  ctx.beginPath();
  ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
  ctx.fill();

  // Draw beak
  ctx.fillStyle = '#FF8C00';
  ctx.beginPath();
  ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
  ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
  ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
  ctx.closePath();
  ctx.fill();
}

// Draw pipes
function drawPipes() {
  ctx.fillStyle = '#228B22';

  pipes.forEach(pipe => {
    // Top pipe
    ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
    ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);
  });
}

```

```

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

```

```

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

```

```

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX === null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position

```

```

ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

```

```
    }  
  }  
}
```

// Collision detection

```
function checkCollisions() {  
  if (gameState !== 'playing') return;  
  
  pipes.forEach(pipe => {  
    // Check collision with top pipe  
    if (bird.x < pipe.x + pipeWidth &&  
      bird.x + bird.width > pipe.x &&  
      bird.y < pipe.topHeight) {  
      gameOver();  
    }  
  
    // Check collision with bottom pipe  
    if (bird.x < pipe.x + pipeWidth &&  
      bird.x + bird.width > pipe.x &&  
      bird.y + bird.height > pipe.bottomY) {  
      gameOver();  
    }  
  });  
}
```

// Game over

```
function gameOver() {  
  if (gameState === 'playing') {  
    gameState = 'gameOver';  
    gameOverElement.classList.remove('hidden');  
  }  
}
```

// Complete level

```
function completeLevel() {  
  if (gameState === 'playing') {  
    gameState = 'levelComplete';  
    levelCompleteElement.classList.remove('hidden');  
    gameOverElement.classList.add('hidden');  
  
    // Mark level as completed  
    markLevelCompleted(currentLevelIndex);  
  
    // Unlock next level  
  }  
}
```

```

        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {

```

```

        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', _jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();
    }
}

```

```

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

Adding coin creation, drawing, spawning, and collision detection:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
    {
        name: "Level 1",
        pipeSpeed: 2,

```



```
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy - Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium - Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,  
    description: "Hard - Pass 12 pipes"  
  },  
  {  
    name: "Level 4",  
    pipeSpeed: 3.5,  
    pipeGap: 140,  
    pipeSpawnInterval: 110,  
    totalPipes: 18,  
    description: "Very Hard - Pass 18 pipes"  
  },  
  {  
    name: "Level 5",  
    pipeSpeed: 4,  
    pipeGap: 130,  
    pipeSpawnInterval: 100,  
    totalPipes: 25,  
    description: "Expert - Pass 25 pipes"  
  },  
  {  
    name: "Level 6",  
    pipeSpeed: 4.5,  
    pipeGap: 120,  
    pipeSpawnInterval: 90,  
    totalPipes: 35,  
    description: "Master - Pass 35 pipes"
```

```

    }
  };

  // Game state
  let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
  'levelComplete'
  let score = 0;
  let currentLevel = null;
  let currentLevelIndex = 0;

  // Level settings (will be set based on selected level)
  let pipeWidth = 60;
  let pipeGap = 150;
  let pipeSpeed = 3;
  let pipeSpawnInterval = 120;
  let totalPipes = 0;
  let pipesSpawned = 0;
  let finishLineX = null;

  // Bird properties
  const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
  };

  // Pipes array
  const pipes = [];
  let pipeSpawnTimer = 0;

  // Coins array
  const coins = [];
  let coinSpawnTimer = 0;
  const coinSpawnInterval = 80; // Spawn coins more frequently
  than pipes

  // LocalStorage functions
  function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');

```

```
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0  
(first level) is always unlocked  
}
```

```
function unlockLevel(levelIndex) {  
    const unlocked = getUnlockedLevels();  
    if (!unlocked.includes(levelIndex) && levelIndex <  
levels.length) {  
        unlocked.push(levelIndex);  
        localStorage.setItem('unlockedLevels',  
JSON.stringify(unlocked));  
    }  
}
```

```
function isLevelUnlocked(levelIndex) {  
    return getUnlockedLevels().includes(levelIndex);  
}
```

```
function markLevelCompleted(levelIndex) {  
    const completed = getCompletedLevels();  
    if (!completed.includes(levelIndex)) {  
        completed.push(levelIndex);  
        localStorage.setItem('completedLevels',  
JSON.stringify(completed));  
    }  
}
```

```
function getCompletedLevels() {  
    const completed = localStorage.getItem('completedLevels');  
    return completed ? JSON.parse(completed) : [];  
}
```

```
function isLevelCompleted(levelIndex) {  
    return getCompletedLevels().includes(levelIndex);  
}
```

```
// Coin functions  
function getCoins() {  
    const coins = localStorage.getItem('totalCoins');  
    return coins ? parseInt(coins) : 0;  
}
```

```
function addCoin() {  
    const currentCoins = getCoins();
```

```

    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

```

```

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;

```

```
const maxHeight = canvas.height - pipeGap - minHeight;  
const topHeight = Math.random() * (maxHeight - minHeight) +  
minHeight;
```

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

```
// Draw finish line  
function drawFinishLine() {  
  if (finishLineX == null) return;  
  
  ctx.strokeStyle = '#FFD700';  
  ctx.lineWidth = 5;  
  ctx.setLineDash([10, 10]);  
  ctx.beginPath();  
  ctx.moveTo(finishLineX, 0);  
  ctx.lineTo(finishLineX, canvas.height);  
  ctx.stroke();  
  ctx.setLineDash([]);  
  
  // Draw "FINISH" text  
  ctx.fillStyle = '#FFD700';  
  ctx.font = 'bold 24px Arial';  
  ctx.textAlign = 'center';  
  ctx.fillText('FINISH', finishLineX, canvas.height / 2);  
  ctx.textAlign = 'left';  
}
```

```
// Draw bird  
function drawBird() {  
  ctx.fillStyle = bird.color;  
  ctx.beginPath();  
  ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,  
bird.width/2, 0, Math.PI * 2);  
  ctx.fill();  
  
  // Draw eye  
  ctx.fillStyle = 'white';  
  ctx.beginPath();
```

```

    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
    }
}

```

```

    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
        bird.y = canvas.height - bird.height;
        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX === null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }
    }
}

```



```

    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX ! null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&

```

```

        bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
    }
}

```

```

        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
}

```

```

});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
```

```

        pipeGap: 150,
        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;

```

```

let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

```

```

}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen

```



```

function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;

```

```

gameOverElement.classList.add('hidden');
levelCompleteElement.classList.add('hidden');

// Reset bird
bird.y = canvas.height / 2;
bird.velocity = 0;

// Reset pipes
pipes.length = 0;
pipeSpawnTimer = 0;
createPipe();
pipesSpawned = 1;

// Reset coins
coins.length = 0;
coinSpawnTimer = 0;

// Update coin display
updateCoinDisplay();

// Show game screen, hide level selection
levelSelectionElement.classList.add('hidden');
gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Spawn coin in a safe area (middle of screen, avoiding
pipes)

```

```

    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
100; // Random around center
    const coinSize = 20;

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';

```

```

    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {

```

```

    bird.velocity += bird.gravity;
    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
        bird.y = canvas.height - bird.height;
        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer++;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX === null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
            }
        }
    }
}

```

```

        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

}

// Collision detection
function checkCollisions() {
    if (gameState != 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&

```

```

        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState === 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState === 'gameOver' || gameState === 'levelComplete')
    {
        gameState = 'playing';
    }
}

```

```

        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

```

```

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

```

```

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

```

```

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
}

```



```

    }
  });

  // Mouse/touch input
  canvas.addEventListener('click', jump);
  canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
  });

  // Back to menu button
  backToMenuButton.addEventListener('click', returnToMenu);

  // Game loop
  function gameLoop() {
    if (gameState !== 'menu') {
      // Clear canvas
      ctx.clearRect(0, 0, canvas.width, canvas.height);

      // Draw background
      const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
      gradient.addColorStop(0, '#87CEEB');
      gradient.addColorStop(0.7, '#87CEEB');
      gradient.addColorStop(0.7, '#90EE90');
      gradient.addColorStop(1, '#90EE90');
      ctx.fillStyle = gradient;
      ctx.fillRect(0, 0, canvas.width, canvas.height);

      // Update and draw
      updateBird();
      updatePipes();
      checkCollisions();

      drawPipes();
      drawFinishLine();
      drawBird();
    }

    requestAnimationFrame(gameLoop);
  }

  // Initialize game
  initLevelSelection();

```

```
gameLoop();
```

```
// Canvas setup  
const canvas = document.getElementById('gameCanvas');  
const ctx = canvas.getContext('2d');  
const scoreElement = document.getElementById('score');  
const gameOverElement = document.getElementById('gameOver');  
const levelCompleteElement =  
document.getElementById('levelComplete');  
const levelInfoElement = document.getElementById('levelInfo');  
const progressElement = document.getElementById('progress');  
const coinsElement = document.getElementById('coins');  
const totalCoinsElement =  
document.getElementById('totalCoins');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backButton = document.getElementById('backToMenu');
```

```
// Level definitions  
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy – Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium – Pass 8 pipes"  
  },  
  {  
    name: "Level 3",
```

```

        pipeSpeed: 3,
        pipeGap: 150,
        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard - Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard - Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert - Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;

```

```

let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {

```

```

    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

```

```

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
}

```

```

    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {

```

```
    // Spawn coin in a safe area (middle of screen, avoiding
    pipes)
    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
    100; // Random around center
    const coinSize = 20;

    coins.push({
      x: canvas.width,
      y: coinY,
      size: coinSize,
      collected: false,
      rotation: 0
    });
}
```

```
// Draw finish line
function drawFinishLine() {
  if (finishLineX = null) return;

  ctx.strokeStyle = '#FFD700';
  ctx.lineWidth = 5;
  ctx.setLineDash([10, 10]);
  ctx.beginPath();
  ctx.moveTo(finishLineX, 0);
  ctx.lineTo(finishLineX, canvas.height);
  ctx.stroke();
  ctx.setLineDash([]);

  // Draw "FINISH" text
  ctx.fillStyle = '#FFD700';
  ctx.font = 'bold 24px Arial';
  ctx.textAlign = 'center';
  ctx.fillText('FINISH', finishLineX, canvas.height / 2);
  ctx.textAlign = 'left';
}
```

```
// Draw bird
function drawBird() {
  ctx.fillStyle = bird.color;
  ctx.beginPath();
  ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,
  bird.width/2, 0, Math.PI * 2);
  ctx.fill();
}
```



```

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins

```

```

function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';
            ctx.textBaseline = 'middle';
            ctx.fillText('$', 0, 0);

            ctx.restore();
        }
    });
}

```

```

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {

```

```

        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }

            // Show finish line when all pipes are passed
            if (score >= totalPipes && finishLineX === null) {
                // Find the rightmost pipe position and place
                finish line after it
                let rightmostX = bird.x + 200; // Default position
                ahead of bird
                if (pipes.length > 0) {
                    pipes.forEach(pipe => {
                        if (pipe.x + pipeWidth > rightmostX) {
                            rightmostX = pipe.x + pipeWidth;
                        }
                    });
                    finishLineX = rightmostX + 150;
                } else {
                    // If no pipes left, place finish line ahead of
                    bird
                    finishLineX = bird.x + 200;
                }
            }

            // Move pipes
            pipes.forEach(pipe => {
                pipe.x -= pipeSpeed;

                // Check if bird passed the pipe

```

```

        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;
            progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

            // Check if all pipes are passed - complete
level immediately
            if (score >= totalPipes) {
                completeLevel();
            }
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
        pipes.shift();
    }
}

// Collision detection
function checkCollisions() {
    if (gameState ! 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {

```

```

        gameOver();
    }
});
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');
    }
}

```

```

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

```

```

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    checkCollisions();

    drawPipes();
    drawFinishLine();
    drawBird();
  }

  requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

Adding coin update logic, collision detection, and integrating it into the game loop:

```

// Canvas setup

```

```
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  }
]
```



```

    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties

```

```
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
  const completed = getCompletedLevels();
```

```

        if (!completed.includes(levelIndex)) {
            completed.push(levelIndex);
            localStorage.setItem('completedLevels',
JSON.stringify(completed));
        }
    }

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {

```

```

        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird

```

```

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Spawn coin in a safe area (middle of screen, avoiding
pipes)
    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
100; // Random around center
    const coinSize = 20;

```

```

        coins.push({
            x: canvas.width,
            y: coinY,
            size: coinSize,
            collected: false,
            rotation: 0
        });
    }

    // Draw finish line
    function drawFinishLine() {
        if (finishLineX = null) return;

        ctx.strokeStyle = '#FFD700';
        ctx.lineWidth = 5;
        ctx.setLineDash([10, 10]);
        ctx.beginPath();
        ctx.moveTo(finishLineX, 0);
        ctx.lineTo(finishLineX, canvas.height);
        ctx.stroke();
        ctx.setLineDash([]);

        // Draw "FINISH" text
        ctx.fillStyle = '#FFD700';
        ctx.font = 'bold 24px Arial';
        ctx.textAlign = 'center';
        ctx.fillText('FINISH', finishLineX, canvas.height / 2);
        ctx.textAlign = 'left';
    }

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

        // Draw eye
        ctx.fillStyle = 'white';
        ctx.beginPath();
        ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
        ctx.fill();
    }

```

```

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);

```

```

        coin.rotation += 0.1;
        ctx.rotate(coin.rotation);

        // Draw coin (golden circle)
        ctx.fillStyle = '#FFD700';
        ctx.beginPath();
        ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
        ctx.fill();

        // Draw coin border
        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

```



```

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score;
                scoreElement.textContent = `Score: ${score}`;
                progressElement.textContent = `Progress: $

```

```

    {score} / ${totalPipes}`;

    // Check if all pipes are passed - complete
    level immediately
    if (score >= totalPipes) {
        completeLevel();
    }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });
    }
}

```

```

        // Remove off-screen or collected coins
        for (let i = coins.length - 1; i >= 0; i--) {
            if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
                coins.splice(i, 1);
            }
        }
    }
}

```

```

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

```

```

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

```

```

// Complete level
function completeLevel() {
    if (gameState === 'playing') {
        gameState = 'levelComplete';
    }
}

```

```

        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels

```

```

}

// Jump function
function jump() {
  if (gameState = 'playing') {
    bird.velocity = bird.jumpStrength;
  } else if (gameState = 'gameOver') {
    resetGame();
  } else if (gameState = 'levelComplete') {
    returnToMenu();
  }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
  if (e.code = 'Space') {
    e.preventDefault();
    jump();
  }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
  e.preventDefault();
  jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
  }
}

```

```

        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [

```

```
{
  name: "Level 1",
  pipeSpeed: 2,
  pipeGap: 180,
  pipeSpawnInterval: 140,
  totalPipes: 5,
  description: "Easy – Pass 5 pipes"
},
{
  name: "Level 2",
  pipeSpeed: 2.5,
  pipeGap: 160,
  pipeSpawnInterval: 130,
  totalPipes: 8,
  description: "Medium – Pass 8 pipes"
},
{
  name: "Level 3",
  pipeSpeed: 3,
  pipeGap: 150,
  pipeSpawnInterval: 120,
  totalPipes: 12,
  description: "Hard – Pass 12 pipes"
},
{
  name: "Level 4",
  pipeSpeed: 3.5,
  pipeGap: 140,
  pipeSpawnInterval: 110,
  totalPipes: 18,
  description: "Very Hard – Pass 18 pipes"
},
{
  name: "Level 5",
  pipeSpeed: 4,
  pipeGap: 130,
  pipeSpawnInterval: 100,
  totalPipes: 25,
  description: "Expert – Pass 25 pipes"
},
{
  name: "Level 6",
  pipeSpeed: 4.5,
  pipeGap: 120,
```

```
        pipeSpawnInterval: 90,  
        totalPipes: 35,  
        description: "Master – Pass 35 pipes"  
    }  
];  
  
// Game state  
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',  
    'levelComplete'  
let score = 0;  
let currentLevel = null;  
let currentLevelIndex = 0;  
  
// Level settings (will be set based on selected level)  
let pipeWidth = 60;  
let pipeGap = 150;  
let pipeSpeed = 3;  
let pipeSpawnInterval = 120;  
let totalPipes = 0;  
let pipesSpawned = 0;  
let finishLineX = null;  
  
// Bird properties  
const bird = {  
    x: 50,  
    y: canvas.height / 2,  
    width: 30,  
    height: 30,  
    velocity: 0,  
    gravity: 0.25,  
    jumpStrength: -5,  
    color: '#FFD700'  
};  
  
// Pipes array  
const pipes = [];  
let pipeSpawnTimer = 0;  
  
// Coins array  
const coins = [];  
let coinSpawnTimer = 0;  
const coinSpawnInterval = 80; // Spawn coins more frequently  
than pipes
```



```
// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}
```

```

}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

```

```

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');

```

```

}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Spawn coin in a safe area (middle of screen, avoiding
pipes)
    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
100; // Random around center
    const coinSize = 20;

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
}

```

```

    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

```

```

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';

```

```

        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position

```

```

ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

```



```

    }
  }
}

// Update coins
function updateCoins() {
  if (gameState = 'playing') {
    // Spawn coins periodically
    coinSpawnTimer;
    if (coinSpawnTimer >= coinSpawnInterval) {
      // Only spawn coins if we haven't reached the pipe
limit
      if (pipesSpawned < totalPipes || pipes.length > 0)
{
        createCoin();
      }
      coinSpawnTimer = 0;
    }

    // Move coins
    coins.forEach(coin => {
      if (!coin.collected) {
        coin.x -= pipeSpeed;
      }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
      if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
        coins.splice(i, 1);
      }
    }
  }
}

// Collision detection
function checkCollisions() {
  if (gameState != 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&

```

```

        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {
                coin.collected = true;
                addCoin();
            }
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level

```

```

function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');

```

```

        gameScreenElement.classList.add('hidden');
        initLevelSelection(); // Refresh to show newly unlocked
levels
    }

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');

```

```

        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard – Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert – Pass 25 pipes"
  },
  {
```

```
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
```

```
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
```



```

    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

```

```

}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');

```

```

        gameScreenElement.classList.remove('hidden');
    }

    // Initialize first pipe
    function createPipe() {
        const minHeight = 50;
        const maxHeight = canvas.height - pipeGap - minHeight;
        const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

    // Create coin
    function createCoin() {
        // Spawn coin in a safe area (middle of screen, avoiding
pipes)
        const coinY = canvas.height / 2 + (Math.random() - 0.5) *
100; // Random around center
        const coinSize = 20;

        coins.push({
            x: canvas.width,
            y: coinY,
            size: coinSize,
            collected: false,
            rotation: 0
        });
    }

    // Draw finish line
    function drawFinishLine() {
        if (finishLineX = null) return;

        ctx.strokeStyle = '#FFD700';
        ctx.lineWidth = 5;
        ctx.setLineDash([10, 10]);
        ctx.beginPath();
        ctx.moveTo(finishLineX, 0);
    }

```

```

    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

```

```

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';

```

```

        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position

```

```

ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

```

```

    }
  }
}

// Update coins
function updateCoins() {
  if (gameState === 'playing') {
    // Spawn coins periodically
    coinSpawnTimer;
    if (coinSpawnTimer >= coinSpawnInterval) {
      // Only spawn coins if we haven't reached the pipe
limit
      if (pipesSpawned < totalPipes || pipes.length > 0)
{
        createCoin();
      }
      coinSpawnTimer = 0;
    }

    // Move coins
    coins.forEach(coin => {
      if (!coin.collected) {
        coin.x -= pipeSpeed;
      }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
      if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
        coins.splice(i, 1);
      }
    }
  }
}

// Collision detection
function checkCollisions() {
  if (gameState !== 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&

```



```

        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {
                coin.collected = true;
                addCoin();
            }
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level

```

```

function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

```

```

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}

```

```

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        checkCollisions();

        drawPipes();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
```

```
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard – Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
```

```

        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];

```

```
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
```

```

        return getCompletedLevels().includes(levelIndex);
    }

    // Coin functions
    function getCoins() {
        const coins = localStorage.getItem('totalCoins');
        return coins ? parseInt(coins) : 0;
    }

    function addCoin() {
        const currentCoins = getCoins();
        localStorage.setItem('totalCoins', (currentCoins +
1).toString());
        updateCoinDisplay();
    }

    function updateCoinDisplay() {
        const totalCoins = getCoins();
        if (totalCoinsElement) {
            totalCoinsElement.textContent = totalCoins;
        }
        if (coinsElement) {
            coinsElement.textContent = `💰 Coins: ${totalCoins}`;
        }
    }

    // Initialize level selection screen
    function initLevelSelection() {
        levelButtonsElement.innerHTML = '';
        updateCoinDisplay(); // Update coin display when showing
menu

        levels.forEach((level, index) => {
            const button = document.createElement('button');
            button.className = 'level-button';
            button.textContent = index + 1;
            button.title = level.description;

            if (!isLevelUnlocked(index)) {
                button.classList.add('locked');
            } else if (isLevelCompleted(index)) {
                button.classList.add('completed');
            }
        });
    }

```



```

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;

```

```

    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Spawn coin in a safe area (middle of screen, avoiding
pipes)
    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
100; // Random around center
    const coinSize = 20;

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {

```

```

    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);

```

```

        ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
        ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
        ctx.closePath();
        ctx.fill();
    }

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';

```

```

        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }
    }
}

```

```

    }

    // Show finish line when all pipes are passed
    if (score >= totalPipes && finishLineX = null) {
        // Find the rightmost pipe position and place
finish line after it
        let rightmostX = bird.x + 200; // Default position
ahead of bird
        if (pipes.length > 0) {
            pipes.forEach(pipe => {
                if (pipe.x + pipeWidth > rightmostX) {
                    rightmostX = pipe.x + pipeWidth;
                }
            });
            finishLineX = rightmostX + 150;
        } else {
            // If no pipes left, place finish line ahead of
bird
            finishLineX = bird.x + 200;
        }
    }

    // Move pipes
    pipes.forEach(pipe => {
        pipe.x -= pipeSpeed;

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score++;
            scoreElement.textContent = `Score: ${score}`;
            progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

            // Check if all pipes are passed - complete
level immediately
            if (score >= totalPipes) {
                completeLevel();
            }
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX != null) {

```

```

        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });

        // Remove off-screen or collected coins
        for (let i = coins.length - 1; i >= 0; i--) {
            if (coins[i].collected || coins[i].x +
            coins[i].size < 0) {
                coins.splice(i, 1);
            }
        }
    }
}

// Collision detection

```

```

function checkCollisions() {
  if (gameState !== 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
      gameOver();
    }

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
      gameOver();
    }
  });

  // Check coin collisions
  coins.forEach(coin => {
    if (!coin.collected) {
      const coinCenterX = coin.x + coin.size / 2;
      const coinCenterY = coin.y + coin.size / 2;
      const birdCenterX = bird.x + bird.width / 2;
      const birdCenterY = bird.y + bird.height / 2;

      const distance = Math.sqrt(
        Math.pow(coinCenterX - birdCenterX, 2) +
        Math.pow(coinCenterY - birdCenterY, 2)
      );

      // Check if bird is close enough to collect coin
      if (distance < (coin.size / 2 + bird.width / 2)) {
        coin.collected = true;
        addCoin();
      }
    }
  });
}

// Game over
function gameOver() {

```



```

    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

```

```

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

```

```

// Game loop
function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    checkCollisions();

    drawPipes();
    drawFinishLine();
    drawBird();
  }

  requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
gameLoop();

```

Adding coin update and drawing to the game loop:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');

```

```
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
```

```

        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,

```

```

    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',

```

```

JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');

```

```

        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;

```



```

    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Spawn coin in a safe area (middle of screen, avoiding
pipes)
    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
100; // Random around center
    const coinSize = 20;

    coins.push({

```

```

        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

```

```

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;

```

```

        ctx.rotate(coin.rotation);

        // Draw coin (golden circle)
        ctx.fillStyle = '#FFD700';
        ctx.beginPath();
        ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
        ctx.fill();

        // Draw coin border
        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes

```

```

function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score;
                scoreElement.textContent = `Score: ${score}`;
                progressElement.textContent = `Progress: $
                {score} / ${totalPipes}`;
            }
        });
    }
}

```

```

        // Check if all pipes are passed - complete
        level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });

        // Remove off-screen or collected coins
    }
}

```

```

        for (let i = coins.length - 1; i >= 0; i--) {
            if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
                coins.splice(i, 1);
            }
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Check coin collisions
coins.forEach(coin => {
    if (!coin.collected) {
        const coinCenterX = coin.x + coin.size / 2;
        const coinCenterY = coin.y + coin.size / 2;
        const birdCenterX = bird.x + bird.width / 2;
        const birdCenterY = bird.y + bird.height / 2;

        const distance = Math.sqrt(
            Math.pow(coinCenterX - birdCenterX, 2) +
            Math.pow(coinCenterY - birdCenterY, 2)
        );

        // Check if bird is close enough to collect coin
    }
});

```

```

        if (distance < (coin.size / 2 + bird.width / 2)) {
            coin.collected = true;
            addCoin();
        }
    }
});
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    }
}

```



```

    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;
  }
}

// Return to menu
function returnToMenu() {
  gameState = 'menu';
  levelSelectionElement.classList.remove('hidden');
  gameScreenElement.classList.add('hidden');
  initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
  if (gameState = 'playing') {
    bird.velocity = bird.jumpStrength;
  } else if (gameState = 'gameOver') {
    resetGame();
  } else if (gameState = 'levelComplete') {
    returnToMenu();
  }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
  if (e.code = 'Space') {
    e.preventDefault();
    jump();
  }
});

```

```
// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
initLevelSelection();
```

```
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
```

```

        pipeSpeed: 3,
        pipeGap: 150,
        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;

```

```

let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {

```

```

        return getUnlockedLevels().includes(levelIndex);
    }

    function markLevelCompleted(levelIndex) {
        const completed = getCompletedLevels();
        if (!completed.includes(levelIndex)) {
            completed.push(levelIndex);
            localStorage.setItem('completedLevels',
JSON.stringify(completed));
        }
    }

    function getCompletedLevels() {
        const completed = localStorage.getItem('completedLevels');
        return completed ? JSON.parse(completed) : [];
    }

    function isLevelCompleted(levelIndex) {
        return getCompletedLevels().includes(levelIndex);
    }

    // Coin functions
    function getCoins() {
        const coins = localStorage.getItem('totalCoins');
        return coins ? parseInt(coins) : 0;
    }

    function addCoin() {
        const currentCoins = getCoins();
        localStorage.setItem('totalCoins', (currentCoins +
1).toString());
        updateCoinDisplay();
    }

    function updateCoinDisplay() {
        const totalCoins = getCoins();
        if (totalCoinsElement) {
            totalCoinsElement.textContent = totalCoins;
        }
        if (coinsElement) {
            coinsElement.textContent = `💰 Coins: ${totalCoins}`;
        }
    }
}

```

```

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
    menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;

```

```

    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

```



```

// Create coin
function createCoin() {
    // Spawn coin in a safe area (middle of screen, avoiding
    pipes)
    const coinY = canvas.height / 2 + (Math.random() - 0.5) *
    100; // Random around center
    const coinSize = 20;

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

// Draw bird
function drawBird() {
    ctx.fillStyle = bird.color;
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,

```

```

bird.width/2, 0, Math.PI * 2);
    ctx.fill();

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

```

```

}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';
            ctx.textBaseline = 'middle';
            ctx.fillText('$', 0, 0);

            ctx.restore();
        }
    });
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;

```

```

        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {

```

```

        pipe.x -= pipeSpeed;

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score++;
            scoreElement.textContent = `Score: ${score}`;
            progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

            // Check if all pipes are passed - complete
level immediately
            if (score >= totalPipes) {
                completeLevel();
            }
        });

        // Move finish line (visual indicator only)
        if (finishLineX != null) {
            finishLineX -= pipeSpeed;
        }

        // Remove off-screen pipes
        while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
            pipes.shift();
        }
    }

    // Update coins
    function updateCoins() {
        if (gameState == 'playing') {
            // Spawn coins periodically
            coinSpawnTimer++;
            if (coinSpawnTimer >= coinSpawnInterval) {
                // Only spawn coins if we haven't reached the pipe
limit
                if (pipesSpawned < totalPipes || pipes.length > 0)
{
                    createCoin();
                }
                coinSpawnTimer = 0;
            }
        }
    }

```

```

    }

    // Move coins
    coins.forEach(coin => {
        if (!coin.collected) {
            coin.x -= pipeSpeed;
        }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
        if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
            coins.splice(i, 1);
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });
}

// Check coin collisions
coins.forEach(coin => {
    if (!coin.collected) {
        const coinCenterX = coin.x + coin.size / 2;

```

```

        const coinCenterY = coin.y + coin.size / 2;
        const birdCenterX = bird.x + bird.width / 2;
        const birdCenterY = bird.y + bird.height / 2;

        const distance = Math.sqrt(
            Math.pow(coinCenterX - birdCenterX, 2) +
            Math.pow(coinCenterY - birdCenterY, 2)
        );

        // Check if bird is close enough to collect coin
        if (distance < (coin.size / 2 + bird.width / 2)) {
            coin.collected = true;
            addCoin();
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState === 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game

```

```

function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

```



```

    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
    }
}

```

```

        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

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

body {
    font-family: 'Arial', sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: 20px;
}

.container {
    text-align: center;
    background: white;
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

h1 {
    color: #333;
    margin-bottom: 15px;
    font-size: 2.5em;

```

```
}

#gameCanvas {
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB
70%, #90EE90 70%, #90EE90 100%);
    display: block;
    margin: 0 auto;
}

.instructions {
    margin-top: 15px;
    color: #555;
}

.instructions p {
    margin: 8px 0;
    font-size: 1.1em;
}

#score {
    font-size: 1.5em;
    font-weight: bold;
    color: #667eea;
}

#gameOver {
    font-size: 1.3em;
    font-weight: bold;
    color: #e74c3c;
    margin-top: 10px;
}

.hidden {
    display: none;
}

/* Level Selection Styles */
.level-selection {
    padding: 20px;
}

.level-selection h2 {
```

```
    color: #333;
    margin-bottom: 20px;
    font-size: 2em;
}

.level-buttons {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px,
1fr));
    gap: 15px;
    margin-bottom: 20px;
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}

.level-button {
    padding: 20px;
    font-size: 1.5em;
    font-weight: bold;
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    color: white;
    cursor: pointer;
    transition: all 0.3s;
}

.level-button:hover:not(.locked) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}

.level-button.locked {
    background: #ccc;
    color: #888;
    cursor: not-allowed;
    opacity: 0.6;
}

.level-button.completed {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d
100%);
```

```
}

.level-instructions {
  color: #666;
  font-size: 1em;
  margin-top: 15px;
}

.coin-display {
  font-size: 1.5em;
  font-weight: bold;
  color: #FFD700;
  margin-bottom: 20px;
  padding: 10px;
  background: linear-gradient(135deg, #FFD700 0%, #FFA500
100%);
  border-radius: 10px;
  color: #333;
  display: inline-block;
}

#coins {
  font-size: 1.2em;
  font-weight: bold;
  color: #FFD700;
}

#levelInfo {
  font-size: 1.3em;
  font-weight: bold;
  color: #764ba2;
}

#progress {
  font-size: 1.2em;
  font-weight: bold;
  color: #e67e22;
}

#levelComplete {
  font-size: 1.3em;
  font-weight: bold;
  color: #27ae60;
  margin-top: 10px;
}
```

```
}

.back-button {
  margin-top: 15px;
  padding: 10px 20px;
  font-size: 1em;
  background: #667eea;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background 0.3s;
}

.back-button:hover {
  background: #5568d3;
}
```

Fixing a CSS issue:

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

body {
  font-family: 'Arial', sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

.container {
  text-align: center;
```

```
        background: white;
        padding: 20px;
        border-radius: 15px;
        box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
    }

    h1 {
        color: #333;
        margin-bottom: 15px;
        font-size: 2.5em;
    }

    #gameCanvas {
        border: 3px solid #333;
        border-radius: 10px;
        background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB
70%, #90EE90 70%, #90EE90 100%);
        display: block;
        margin: 0 auto;
    }

    .instructions {
        margin-top: 15px;
        color: #555;
    }

    .instructions p {
        margin: 8px 0;
        font-size: 1.1em;
    }

    #score {
        font-size: 1.5em;
        font-weight: bold;
        color: #667eea;
    }

    #gameOver {
        font-size: 1.3em;
        font-weight: bold;
        color: #e74c3c;
        margin-top: 10px;
    }
```

```
.hidden {
    display: none;
}

/* Level Selection Styles */
.level-selection {
    padding: 20px;
}

.level-selection h2 {
    color: #333;
    margin-bottom: 20px;
    font-size: 2em;
}

.level-buttons {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px,
1fr));
    gap: 15px;
    margin-bottom: 20px;
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}

.level-button {
    padding: 20px;
    font-size: 1.5em;
    font-weight: bold;
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    color: white;
    cursor: pointer;
    transition: all 0.3s;
}

.level-button:hover:not(.locked) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}
```



```
.level-button.locked {
    background: #ccc;
    color: #888;
    cursor: not-allowed;
    opacity: 0.6;
}

.level-button.completed {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d
100%);
}

.level-instructions {
    color: #666;
    font-size: 1em;
    margin-top: 15px;
}

.coin-display {
    font-size: 1.5em;
    font-weight: bold;
    margin-bottom: 20px;
    padding: 10px;
    background: linear-gradient(135deg, #FFD700 0%, #FFA500
100%);
    border-radius: 10px;
    color: #333;
    display: inline-block;
}

#coins {
    font-size: 1.2em;
    font-weight: bold;
    color: #FFD700;
}

#levelInfo {
    font-size: 1.3em;
    font-weight: bold;
    color: #764ba2;
}

#progress {
```

```

    font-size: 1.2em;
    font-weight: bold;
    color: #e67e22;
}

#levelComplete {
    font-size: 1.3em;
    font-weight: bold;
    color: #27ae60;
    margin-top: 10px;
}

.back-button {
    margin-top: 15px;
    padding: 10px 20px;
    font-size: 1em;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
}

.back-button:hover {
    background: #5568d3;
}

```

Improving coin placement to avoid pipes:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const gameOverElement = document.getElementById('gameOver');
const levelCompleteElement =
document.getElementById('levelComplete');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');

```

```
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard – Pass 18 pipes"
  }
]
```

```

    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,

```

```

        color: '#FFD700'
    };

    // Pipes array
    const pipes = [];
    let pipeSpawnTimer = 0;

    // Coins array
    const coins = [];
    let coinSpawnTimer = 0;
    const coinSpawnInterval = 80; // Spawn coins more frequently
    than pipes

    // LocalStorage functions
    function getUnlockedLevels() {
        const unlocked = localStorage.getItem('unlockedLevels');
        return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
        (first level) is always unlocked
    }

    function unlockLevel(levelIndex) {
        const unlocked = getUnlockedLevels();
        if (!unlocked.includes(levelIndex) && levelIndex <
        levels.length) {
            unlocked.push(levelIndex);
            localStorage.setItem('unlockedLevels',
            JSON.stringify(unlocked));
        }
    }

    function isLevelUnlocked(levelIndex) {
        return getUnlockedLevels().includes(levelIndex);
    }

    function markLevelCompleted(levelIndex) {
        const completed = getCompletedLevels();
        if (!completed.includes(levelIndex)) {
            completed.push(levelIndex);
            localStorage.setItem('completedLevels',
            JSON.stringify(completed));
        }
    }

    function getCompletedLevels() {

```

```

    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {

```

```

        button.classList.add('locked');
    } else if (isLevelCompleted(index)) {
        button.classList.add('completed');
    }

    if (isLevelUnlocked(index)) {
        button.addEventListener('click', () =>
startLevel(index));
    }

    levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;

```

```

    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

        // Check if this position would overlap with any nearby

```



```

pipes
    let tooCloseToPipe = false;
    pipes.forEach(pipe => {
        const distance = Math.abs(pipe.x - canvas.width);
        // Only check pipes that are close to spawn point
        if (distance < 100) {
            // Check if coin would be in pipe area
            if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                tooCloseToPipe = true;
            }
        }
    });

    if (!tooCloseToPipe) {
        safePosition = true;
    }
    attempts++;
}

// If we couldn't find a safe position, use center
if (!safePosition) {
    coinY = canvas.height / 2;
}

coins.push({
    x: canvas.width,
    y: coinY,
    size: coinSize,
    collected: false,
    rotation: 0
});
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
}

```

```

    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

```

```

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';

```

```

        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position

```

```

ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed – complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

```

```

    }
  }
}

// Update coins
function updateCoins() {
  if (gameState = 'playing') {
    // Spawn coins periodically
    coinSpawnTimer;
    if (coinSpawnTimer >= coinSpawnInterval) {
      // Only spawn coins if we haven't reached the pipe
      limit
      if (pipesSpawned < totalPipes || pipes.length > 0)
      {
        createCoin();
      }
      coinSpawnTimer = 0;
    }

    // Move coins
    coins.forEach(coin => {
      if (!coin.collected) {
        coin.x -= pipeSpeed;
      }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
      if (coins[i].collected || coins[i].x +
      coins[i].size < 0) {
        coins.splice(i, 1);
      }
    }
  }
}

// Collision detection
function checkCollisions() {
  if (gameState != 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&

```

```

        bird.x + bird.width > pipe.x &&
bird.y < pipe.topHeight) {
    gameOver();
}

// Check collision with bottom pipe
if (bird.x < pipe.x + pipeWidth &&
bird.x + bird.width > pipe.x &&
bird.y + bird.height > pipe.bottomY) {
    gameOver();
}
});

// Check coin collisions
coins.forEach(coin => {
    if (!coin.collected) {
        const coinCenterX = coin.x + coin.size / 2;
const coinCenterY = coin.y + coin.size / 2;
const birdCenterX = bird.x + bird.width / 2;
const birdCenterY = bird.y + bird.height / 2;

        const distance = Math.sqrt(
            Math.pow(coinCenterX - birdCenterX, 2) +
            Math.pow(coinCenterY - birdCenterY, 2)
        );

        // Check if bird is close enough to collect coin
if (distance < (coin.size / 2 + bird.width / 2)) {
            coin.collected = true;
            addCoin();
        }
    }
});
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level

```

```

function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

```



```
// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}
```

```
// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}
```

```
// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});
```

```
// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});
```

```
// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);
```

```
// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}
```

```

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Added a coin system to your Flappy Bird game. Here's what was implemented:

Coin Features:

1. **Coin spawning:** Coins spawn periodically during gameplay, placed to avoid pipes.
2. **Visual design:**
 - Gold coins with a rotating "\$" symbol
 - Smooth rotation animation

3. Collection system:

- Collect coins by touching them with the bird
- Coins are removed when collected

4. Persistent storage:

- Total coins are saved in localStorage
- Coins persist between levels and game sessions
- Total coin count is displayed in the level selection menu and during gameplay

5. Smart placement: Coins spawn in safe areas, avoiding pipe collisions

=====

User

whne you die, add a pop up menu with a button on the screen that you have to press to play again, add another button that lets you keep going where you left off for 10 coins. if the user spends the coins to keep going, delete the next couple of pipes and move them to the back of the sequence to give thme time to recover, make the user start in the middle of the screen and not the group

=====

Cursor

Adding a game-over popup with "Play Again" and "Continue (10 coins)" options. Implementing the continue logic: removing the next few pipes, moving them to the back, and resetting the bird to the middle.

Adding the game-over popup with continue and restart options:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-  
scale=1.0">  
    <title>Flappy Bird</title>  
    <link rel="stylesheet" href="style.css">  
</head>
```

```

<body>
  <div class="container">
    <h1>Flappy Bird</h1>

    <!-- Level Selection Screen -->
    <div id="levelSelection" class="level-selection">
      <h2>Select Level</h2>
      <div id="coinDisplay" class="coin-display">💰
Coins: <span id="totalCoins">0</span></div>
      <div id="levelButtons" class="level-buttons"></div>
      <p class="level-instructions">Complete a level to
unlock the next one!</p>
    </div>

    <!-- Game Screen -->
    <div id="gameScreen" class="hidden">
      <canvas id="gameCanvas" width="400" height="600"></
canvas>
      <div class="instructions">
        <p>Press <strong>SPACE</strong> to flap and
avoid the pipes!</p>
        <p id="levelInfo">Level: 1</p>
        <p id="progress">Progress: 0 / 0</p>
        <p id="score">Score: 0</p>
        <p id="coins">💰 Coins: 0</p>
        <p id="levelComplete" class="hidden">Level
Complete! Press SPACE to continue</p>
        <button id="backToMenu" class="back-
button">Back to Menu</button>
      </div>

      <!-- Game Over Popup -->
      <div id="gameOverPopup" class="popup-overlay hidden">
        <div class="popup-content">
          <h2>Game Over!</h2>
          <p>What would you like to do?</p>
          <div class="popup-buttons">
            <button id="playAgainBtn" class="popup-
button play-again-btn">Play Again</button>
            <button id="continueBtn" class="popup-
button continue-btn">Continue (10 coins)</button>
          </div>
          <p id="notEnoughCoins" class="error-message
hidden">Not enough coins!</p>

```

```

        </div>
    </div>
    </div>
</div>
<script src="game.js"></script>
</body>
</html>

```

Fixing the HTML structure, then adding the popup functionality:

Fixing the HTML structure:

```

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

        <!-- Level Selection Screen -->
        <div id="levelSelection" class="level-selection">
            <h2>Select Level</h2>
            <div id="coinDisplay" class="coin-display">💰
Coins: <span id="totalCoins">0</span></div>
            <div id="levelButtons" class="level-buttons"></div>
            <p class="level-instructions">Complete a level to
unlock the next one!</p>
        </div>

        <!-- Game Screen -->
    </div>

```

```

    <div id="gameScreen" class="hidden">
        <canvas id="gameCanvas" width="400" height="600"></
canvas>
        <div class="instructions">
            <p>Press <strong>SPACE</strong> to flap and
avoid the pipes!</p>
            <p id="levelInfo">Level: 1</p>
            <p id="progress">Progress: 0 / 0</p>
            <p id="score">Score: 0</p>
            <p id="coins">💰 Coins: 0</p>
            <p id="levelComplete" class="hidden">Level
Complete! Press SPACE to continue</p>
            <button id="backToMenu" class="back-
button">Back to Menu</button>
        </div>
    </div>

    <!-- Game Over Popup -->
    <div id="gameOverPopup" class="popup-overlay hidden">
        <div class="popup-content">
            <h2>Game Over!</h2>
            <p>What would you like to do?</p>
            <div class="popup-buttons">
                <button id="playAgainBtn" class="popup-
button play-again-btn">Play Again</button>
                <button id="continueBtn" class="popup-
button continue-btn">Continue (10 coins)</button>
            </div>
            <p id="notEnoughCoins" class="error-message
hidden">Not enough coins!</p>
        </div>
    </div>
</div>
<script src="game.js"></script>
</body>
</html>

```

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');

```

```
const levelCompleteElement =  
document.getElementById('levelComplete');  
const gameOverPopup = document.getElementById('gameOverPopup');  
const playAgainBtn = document.getElementById('playAgainBtn');  
const continueBtn = document.getElementById('continueBtn');  
const notEnoughCoinsElement =  
document.getElementById('notEnoughCoins');  
const levelInfoElement = document.getElementById('levelInfo');  
const progressElement = document.getElementById('progress');  
const coinsElement = document.getElementById('coins');  
const totalCoinsElement =  
document.getElementById('totalCoins');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
```

```
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy – Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium – Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,
```

```

        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
}
};

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

```



```
// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
  (first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
  levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
    JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}
```

```

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {

```

```
levelButtonsElement.innerHTML = '';  
updateCoinDisplay(); // Update coin display when showing  
menu
```

```
levels.forEach((level, index) => {  
    const button = document.createElement('button');  
    button.className = 'level-button';  
    button.textContent = index + 1;  
    button.title = level.description;  
  
    if (!isLevelUnlocked(index)) {  
        button.classList.add('locked');  
    } else if (isLevelCompleted(index)) {  
        button.classList.add('completed');  
    }  
  
    if (isLevelUnlocked(index)) {  
        button.addEventListener('click', () =>  
startLevel(index));  
    }  
  
    levelButtonsElement.appendChild(button);  
});  
}
```

```
// Start a level  
function startLevel(levelIndex) {  
    currentLevelIndex = levelIndex;  
    currentLevel = levels[levelIndex];  
  
    // Set level-specific settings  
    pipeSpeed = currentLevel.pipeSpeed;  
    pipeGap = currentLevel.pipeGap;  
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;  
    totalPipes = currentLevel.totalPipes;  
  
    // Reset game state  
    gameState = 'playing';  
    score = 0;  
    pipesSpawned = 0;  
    finishLineX = null;  
    scoreElement.textContent = `Score: ${score}`;  
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;  
    progressElement.textContent = `Progress: ${score} / $
```

```

{totalPipes}`;
  gameOverElement.classList.add('hidden');
  levelCompleteElement.classList.add('hidden');

  // Reset bird
  bird.y = canvas.height / 2;
  bird.velocity = 0;

  // Reset pipes
  pipes.length = 0;
  pipeSpawnTimer = 0;
  createPipe();
  pipesSpawned = 1;

  // Reset coins
  coins.length = 0;
  coinSpawnTimer = 0;

  // Update coin display
  updateCoinDisplay();

  // Show game screen, hide level selection
  levelSelectionElement.classList.add('hidden');
  gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
  const minHeight = 50;
  const maxHeight = canvas.height - pipeGap - minHeight;
  const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
  // Find a safe Y position that avoids pipes

```

```

const coinSize = 20;
let coinY;
let attempts = 0;
let safePosition = false;

// Try to find a safe position
while (!safePosition && attempts < 10) {
    coinY = 50 + Math.random() * (canvas.height - 100); //
    Random Y, avoiding top/bottom edges

    // Check if this position would overlap with any nearby
    pipes
    let tooCloseToPipe = false;
    pipes.forEach(pipe => {
        const distance = Math.abs(pipe.x - canvas.width);
        // Only check pipes that are close to spawn point
        if (distance < 100) {
            // Check if coin would be in pipe area
            if (coinY < pipe.topHeight || coinY + coinSize
                > pipe.bottomY) {
                tooCloseToPipe = true;
            }
        }
    });

    if (!tooCloseToPipe) {
        safePosition = true;
    }
    attempts;
}

// If we couldn't find a safe position, use center
if (!safePosition) {
    coinY = canvas.height / 2;
}

coins.push({
    x: canvas.width,
    y: coinY,
    size: coinSize,
    collected: false,
    rotation: 0
});
}

```

```

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak

```

```

    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

```

```

        // Draw coin border
        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState === 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer++;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
            }
        }
    }
}

```



```

        pipesSpawned;
        pipeSpawnTimer = 0;
    }
}

// Show finish line when all pipes are passed
if (score >= totalPipes && finishLineX = null) {
    // Find the rightmost pipe position and place
finish line after it
    let rightmostX = bird.x + 200; // Default position
ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed – complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

```

```

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });

        // Remove off-screen or collected coins
        for (let i = coins.length - 1; i >= 0; i--) {
            if (coins[i].collected || coins[i].x +
            coins[i].size < 0) {
                coins.splice(i, 1);
            }
        }
    }
}

```

```

}

// Collision detection
function checkCollisions() {
  if (gameState !== 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
      gameOver();
    }

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
      gameOver();
    }
  });

  // Check coin collisions
  coins.forEach(coin => {
    if (!coin.collected) {
      const coinCenterX = coin.x + coin.size / 2;
      const coinCenterY = coin.y + coin.size / 2;
      const birdCenterX = bird.x + bird.width / 2;
      const birdCenterY = bird.y + bird.height / 2;

      const distance = Math.sqrt(
        Math.pow(coinCenterX - birdCenterX, 2) +
        Math.pow(coinCenterY - birdCenterY, 2)
      );

      // Check if bird is close enough to collect coin
      if (distance < (coin.size / 2 + bird.width / 2)) {
        coin.collected = true;
        addCoin();
      }
    }
  });
}

```

```

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
    }
}

```

```

        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

```

```

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    updateCoins();
    checkCollisions();

    drawPipes();
    drawCoins();
    drawFinishLine();
    drawBird();
  }

  requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Updating the game over logic and adding continue functionality:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },

```

```

    {
      name: "Level 3",
      pipeSpeed: 3,
      pipeGap: 150,
      pipeSpawnInterval: 120,
      totalPipes: 12,
      description: "Hard – Pass 12 pipes"
    },
    {
      name: "Level 4",
      pipeSpeed: 3.5,
      pipeGap: 140,
      pipeSpawnInterval: 110,
      totalPipes: 18,
      description: "Very Hard – Pass 18 pipes"
    },
    {
      name: "Level 5",
      pipeSpeed: 4,
      pipeGap: 130,
      pipeSpawnInterval: 100,
      totalPipes: 25,
      description: "Expert – Pass 25 pipes"
    },
    {
      name: "Level 6",
      pipeSpeed: 4.5,
      pipeGap: 120,
      pipeSpawnInterval: 90,
      totalPipes: 35,
      description: "Master – Pass 35 pipes"
    }
  ];

  // Game state
  let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
  'levelComplete'
  let score = 0;
  let currentLevel = null;
  let currentLevelIndex = 0;

  // Level settings (will be set based on selected level)
  let pipeWidth = 60;
  let pipeGap = 150;

```



```

let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

```

```
function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
}
```

```

        return false;
    }

    function updateCoinDisplay() {
        const totalCoins = getCoins();
        if (totalCoinsElement) {
            totalCoinsElement.textContent = totalCoins;
        }
        if (coinsElement) {
            coinsElement.textContent = `💰 Coins: ${totalCoins}`;
        }
    }

    // Initialize level selection screen
    function initLevelSelection() {
        levelButtonsElement.innerHTML = '';
        updateCoinDisplay(); // Update coin display when showing
        menu

        levels.forEach((level, index) => {
            const button = document.createElement('button');
            button.className = 'level-button';
            button.textContent = index + 1;
            button.title = level.description;

            if (!isLevelUnlocked(index)) {
                button.classList.add('locked');
            } else if (isLevelCompleted(index)) {
                button.classList.add('completed');
            }

            if (isLevelUnlocked(index)) {
                button.addEventListener('click', () =>
startLevel(index));
            }

            levelButtonsElement.appendChild(button);
        });
    }

    // Start a level
    function startLevel(levelIndex) {
        currentLevelIndex = levelIndex;
        currentLevel = levels[levelIndex];
    }

```

```

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
{totalPipes}}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;

```

```

    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

  // Create coin
function createCoin() {
  // Find a safe Y position that avoids pipes
  const coinSize = 20;
  let coinY;
  let attempts = 0;
  let safePosition = false;

  // Try to find a safe position
  while (!safePosition && attempts < 10) {
    coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

    // Check if this position would overlap with any nearby
pipes
    let tooCloseToPipe = false;
    pipes.forEach(pipe => {
      const distance = Math.abs(pipe.x - canvas.width);
      // Only check pipes that are close to spawn point
      if (distance < 100) {
        // Check if coin would be in pipe area
        if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
          tooCloseToPipe = true;
        }
      }
    });

    if (!tooCloseToPipe) {
      safePosition = true;
    }
    attempts++;
  }
}

```

```

    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

// Draw bird
function drawBird() {
    ctx.fillStyle = bird.color;
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2, bird.y + bird.height/2,
bird.width/2, 0, Math.PI * 2);
    ctx.fill();
}

```

```

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

```

```

// Draw coins
function drawCoins() {
  coins.forEach(coin => {
    if (!coin.collected) {
      ctx.save();
      ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
      coin.rotation += 0.1;
      ctx.rotate(coin.rotation);

      // Draw coin (golden circle)
      ctx.fillStyle = '#FFD700';
      ctx.beginPath();
      ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
      ctx.fill();

      // Draw coin border
      ctx.strokeStyle = '#FFA500';
      ctx.lineWidth = 2;
      ctx.stroke();

      // Draw $ symbol
      ctx.fillStyle = '#FF8C00';
      ctx.font = `${coin.size * 0.6}px Arial`;
      ctx.textAlign = 'center';
      ctx.textBaseline = 'middle';
      ctx.fillText('$', 0, 0);

      ctx.restore();
    }
  });
}

```

```

// Update bird
function updateBird() {
  if (gameState === 'playing') {
    bird.velocity += bird.gravity;
    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
      bird.y = canvas.height - bird.height;
      gameOver();
    }
  }
}

```



```

    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX === null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

```

```

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;
            progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

            // Check if all pipes are passed - complete
level immediately
            if (score >= totalPipes) {
                completeLevel();
            }
        });

        // Move finish line (visual indicator only)
        if (finishLineX ! null) {
            finishLineX -= pipeSpeed;
        }

        // Remove off-screen pipes
        while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
        {
            pipes.shift();
        }
    }

    // Update coins
    function updateCoins() {
        if (gameState = 'playing') {
            // Spawn coins periodically
            coinSpawnTimer;
            if (coinSpawnTimer >= coinSpawnInterval) {
                // Only spawn coins if we haven't reached the pipe
limit
                if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }
    }

```

```

    // Move coins
    coins.forEach(coin => {
        if (!coin.collected) {
            coin.x -= pipeSpeed;
        }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
        if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
            coins.splice(i, 1);
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;

```

```

        const birdCenterX = bird.x + bird.width / 2;
        const birdCenterY = bird.y + bird.height / 2;

        const distance = Math.sqrt(
            Math.pow(coinCenterX - birdCenterX, 2) +
            Math.pow(coinCenterY - birdCenterY, 2)
        );

        // Check if bird is close enough to collect coin
        if (distance < (coin.size / 2 + bird.width / 2)) {
            coin.collected = true;
            addCoin();
        }
    }
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverElement.classList.remove('hidden');
    }
}

// Complete level
function completeLevel() {
    if (gameState === 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Reset game
function resetGame() {

```

```

    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverElement.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

```

```
}
```

```
// Keyboard input
```

```
document.addEventListener('keydown', (e) => {  
    if (e.code === 'Space') {  
        e.preventDefault();  
        jump();  
    }  
});
```

```
// Mouse/touch input
```

```
canvas.addEventListener('click', jump);  
canvas.addEventListener('touchstart', (e) => {  
    e.preventDefault();  
    jump();  
});
```

```
// Back to menu button
```

```
backToMenuButton.addEventListener('click', returnToMenu);
```

```
// Game loop
```

```
function gameLoop() {  
    if (gameState !== 'menu') {  
        // Clear canvas  
        ctx.clearRect(0, 0, canvas.width, canvas.height);  
  
        // Draw background  
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);  
        gradient.addColorStop(0, '#87CEEB');  
        gradient.addColorStop(0.7, '#87CEEB');  
        gradient.addColorStop(0.7, '#90EE90');  
        gradient.addColorStop(1, '#90EE90');  
        ctx.fillStyle = gradient;  
        ctx.fillRect(0, 0, canvas.width, canvas.height);  
  
        // Update and draw  
        updateBird();  
        updatePipes();  
        updateCoins();  
        checkCollisions();  
  
        drawPipes();  
        drawCoins();  
    }  
}
```

```

        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
    {
        name: "Level 1",
        pipeSpeed: 2,
        pipeGap: 180,
    }

```

```
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard – Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert – Pass 25 pipes"
  },
  {
    name: "Level 6",
    pipeSpeed: 4.5,
    pipeGap: 120,
    pipeSpawnInterval: 90,
    totalPipes: 35,
    description: "Master – Pass 35 pipes"
  }
}
```


};

```
// Game state  
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',  
'levelComplete'  
let score = 0;  
let currentLevel = null;  
let currentLevelIndex = 0;
```

```
// Level settings (will be set based on selected level)  
let pipeWidth = 60;  
let pipeGap = 150;  
let pipeSpeed = 3;  
let pipeSpawnInterval = 120;  
let totalPipes = 0;  
let pipesSpawned = 0;  
let finishLineX = null;
```

```
// Bird properties  
const bird = {  
    x: 50,  
    y: canvas.height / 2,  
    width: 30,  
    height: 30,  
    velocity: 0,  
    gravity: 0.25,  
    jumpStrength: -5,  
    color: '#FFD700'  
};
```

```
// Pipes array  
const pipes = [];  
let pipeSpawnTimer = 0;
```

```
// Coins array  
const coins = [];  
let coinSpawnTimer = 0;  
const coinSpawnInterval = 80; // Spawn coins more frequently  
than pipes
```

```
// LocalStorage functions  
function getUnlockedLevels() {  
    const unlocked = localStorage.getItem('unlockedLevels');  
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
```

```
(first level) is always unlocked  
}
```

```
function unlockLevel(levelIndex) {  
    const unlocked = getUnlockedLevels();  
    if (!unlocked.includes(levelIndex) && levelIndex <  
levels.length) {  
        unlocked.push(levelIndex);  
        localStorage.setItem('unlockedLevels',  
JSON.stringify(unlocked));  
    }  
}
```

```
function isLevelUnlocked(levelIndex) {  
    return getUnlockedLevels().includes(levelIndex);  
}
```

```
function markLevelCompleted(levelIndex) {  
    const completed = getCompletedLevels();  
    if (!completed.includes(levelIndex)) {  
        completed.push(levelIndex);  
        localStorage.setItem('completedLevels',  
JSON.stringify(completed));  
    }  
}
```

```
function getCompletedLevels() {  
    const completed = localStorage.getItem('completedLevels');  
    return completed ? JSON.parse(completed) : [];  
}
```

```
function isLevelCompleted(levelIndex) {  
    return getCompletedLevels().includes(levelIndex);  
}
```

```
// Coin functions
```

```
function getCoins() {  
    const coins = localStorage.getItem('totalCoins');  
    return coins ? parseInt(coins) : 0;  
}
```

```
function addCoin() {  
    const currentCoins = getCoins();  
    localStorage.setItem('totalCoins', (currentCoins +
```

```

1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {

```

```

        button.addEventListener('click', () =>
startLevel(index));
    }

    levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

```

```

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

        // Check if this position would overlap with any nearby
pipes
        let tooCloseToPipe = false;
        pipes.forEach(pipe => {
            const distance = Math.abs(pipe.x - canvas.width);
            // Only check pipes that are close to spawn point
            if (distance < 100) {

```

```

        // Check if coin would be in pipe area
        if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
            tooCloseToPipe = true;
        }
    });

    if (!tooCloseToPipe) {
        safePosition = true;
    }
    attempts;
}

// If we couldn't find a safe position, use center
if (!safePosition) {
    coinY = canvas.height / 2;
}

coins.push({
    x: canvas.width,
    y: coinY,
    size: coinSize,
    collected: false,
    rotation: 0
});
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';

```

```

    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe

```

```

        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';
            ctx.textBaseline = 'middle';
            ctx.fillText('$', 0, 0);

            ctx.restore();
        }
    });
}

```



```

// Update bird
function updateBird() {
  if (gameState === 'playing') {
    bird.velocity += bird.gravity;
    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
      bird.y = canvas.height - bird.height;
      gameOver();
    }
    if (bird.y < 0) {
      bird.y = 0;
      bird.velocity = 0;
    }
  }
}

// Update pipes
function updatePipes() {
  if (gameState === 'playing') {
    // Spawn new pipes until we reach the total
    if (pipesSpawned < totalPipes) {
      pipeSpawnTimer++;
      if (pipeSpawnTimer >= pipeSpawnInterval) {
        createPipe();
        pipesSpawned++;
        pipeSpawnTimer = 0;
      }
    }

    // Show finish line when all pipes are passed
    if (score >= totalPipes && finishLineX === null) {
      // Find the rightmost pipe position and place
      // finish line after it
      let rightmostX = bird.x + 200; // Default position
      // ahead of bird
      if (pipes.length > 0) {
        pipes.forEach(pipe => {
          if (pipe.x + pipeWidth > rightmostX) {
            rightmostX = pipe.x + pipeWidth;
          }
        });
      }
    }
  }
}

```

```

        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Update coins
function updateCoins() {
    if (gameState == 'playing') {

```

```

    // Spawn coins periodically
    coinSpawnTimer;
    if (coinSpawnTimer >= coinSpawnInterval) {
        // Only spawn coins if we haven't reached the pipe
limit
        if (pipesSpawned < totalPipes || pipes.length > 0)
        {
            createCoin();
        }
        coinSpawnTimer = 0;
    }

    // Move coins
    coins.forEach(coin => {
        if (!coin.collected) {
            coin.x -= pipeSpeed;
        }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
        if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
            coins.splice(i, 1);
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&

```

```

        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {
                coin.collected = true;
                addCoin();
            }
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

```

```

    }
  }
}

// Complete level
function completeLevel() {
  if (gameState = 'playing') {
    gameState = 'levelComplete';
    levelCompleteElement.classList.remove('hidden');
    gameOverElement.classList.add('hidden');

    // Mark level as completed
    markLevelCompleted(currentLevelIndex);

    // Unlock next level
    if (currentLevelIndex + 1 < levels.length) {
      unlockLevel(currentLevelIndex + 1);
    }
  }
}

// Reset game
function resetGame() {
  if (gameState = 'gameOver' || gameState = 'levelComplete')
  {
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins

```

```

        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'gameOver') {
        resetGame();
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop

```

```

function gameLoop() {
  if (gameState !== 'menu') {
    // Clear canvas
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    updateCoins();
    checkCollisions();

    drawPipes();
    drawCoins();
    drawFinishLine();
    drawBird();
  }

  requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');

```

```
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  },
  {
```



```

        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,

```

```

    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
    }
}

```

```

        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {

```

```

        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
    menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
}

```

```

    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
{totalPipes}}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

```

```

    });
}

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
        Random Y, avoiding top/bottom edges

        // Check if this position would overlap with any nearby
        pipes
        let tooCloseToPipe = false;
        pipes.forEach(pipe => {
            const distance = Math.abs(pipe.x - canvas.width);
            // Only check pipes that are close to spawn point
            if (distance < 100) {
                // Check if coin would be in pipe area
                if (coinY < pipe.topHeight || coinY + coinSize
                > pipe.bottomY) {
                    tooCloseToPipe = true;
                }
            }
        });

        if (!tooCloseToPipe) {
            safePosition = true;
        }
        attempts++;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,

```

```

        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX == null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';

```

```

    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

```



```

    // Draw coin (golden circle)
    ctx.fillStyle = '#FFD700';
    ctx.beginPath();
    ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
    ctx.fill();

    // Draw coin border
    ctx.strokeStyle = '#FFA500';
    ctx.lineWidth = 2;
    ctx.stroke();

    // Draw $ symbol
    ctx.fillStyle = '#FF8C00';
    ctx.font = `${coin.size * 0.6}px Arial`;
    ctx.textAlign = 'center';
    ctx.textBaseline = 'middle';
    ctx.fillText('$', 0, 0);

    ctx.restore();
  }
}

// Update bird
function updateBird() {
  if (gameState === 'playing') {
    bird.velocity += bird.gravity;
    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
      bird.y = canvas.height - bird.height;
      gameOver();
    }
    if (bird.y < 0) {
      bird.y = 0;
      bird.velocity = 0;
    }
  }
}

// Update pipes
function updatePipes() {

```

```

    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

            // Check if bird passed the pipe
            if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
                pipe.passed = true;
                score++;
                scoreElement.textContent = `Score: ${score}`;
                progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

                // Check if all pipes are passed – complete

```

```

level immediately
    if (score >= totalPipes) {
        completeLevel();
    }
});

// Move finish line (visual indicator only)
if (finishLineX ! null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });

        // Remove off-screen or collected coins
        for (let i = coins.length - 1; i >= 0; i--) {

```

```

        if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
            coins.splice(i, 1);
        }
    }
}

```

// Collision detection

```

function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {

```

```

        coin.collected = true;
        addCoin();
    }
}

});

}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

```

```

// Continue game (revive)
function continueGame() {
  if (gameState = 'gameOver') {
    // Check if user has enough coins
    if (!spendCoins(10)) {
      notEnoughCoinsElement.classList.remove('hidden');
      return;
    }

    // Remove next 2-3 pipes and move them to the back
    const pipesToRemove = Math.min(3, pipes.length);
    const removedPipes = [];

    // Remove pipes that are ahead of the bird
    for (let i = pipes.length - 1; i >= 0; i--) {
      if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
        removedPipes.push(pipes.splice(i, 1)[0]);
      }
    }

    // Move removed pipes to the back (far right)
    removedPipes.forEach(pipe => {
      pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
      pipe.passed = false; // Reset passed status
      pipes.push(pipe);
    });

    // Reset bird to middle of screen
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Clear collected coins that are off-screen
    coins.forEach(coin => {
      if (coin.x < bird.x - 100) {
        coin.collected = true;
      }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
  }
}

```

```

        notEnoughCoinsElement.classList.add('hidden');
    }
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {

```

```

        bird.velocity = bird.jumpStrength;
    } else if (gameState === 'gameOver') {
        resetGame();
    } else if (gameState === 'levelComplete') {
        returnToMenu();
    }
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
    }
}

```



```

        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();


// Canvas setup
const canvas = document.getElementById('gameCanvas');;
const ctx = canvas.getContext('2d');;
const scoreElement = document.getElementById('score');;
const levelCompleteElement =
document.getElementById('levelComplete');;
const gameOverPopup = document.getElementById('gameOverPopup');;
const playAgainBtn = document.getElementById('playAgainBtn');;
const continueBtn = document.getElementById('continueBtn');;
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');;
const levelInfoElement = document.getElementById('levelInfo');;
const progressElement = document.getElementById('progress');;
const coinsElement = document.getElementById('coins');;
const totalCoinsElement =
document.getElementById('totalCoins');;
const levelSelectionElement =
document.getElementById('levelSelection');;
const gameScreenElement =
document.getElementById('gameScreen');;
const levelButtonsElement =
document.getElementById('levelButtons');;
const backToMenuButton = document.getElementById('backToMenu');;

```

```
// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard - Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert - Pass 25 pipes"
  }
]
```

```

        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
};

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;

```

```
const coinSpawnInterval = 80; // Spawn coins more frequently  
than pipes
```

```
// LocalStorage functions
```

```
function getUnlockedLevels() {  
    const unlocked = localStorage.getItem('unlockedLevels');  
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0  
(first level) is always unlocked  
}
```

```
function unlockLevel(levelIndex) {  
    const unlocked = getUnlockedLevels();  
    if (!unlocked.includes(levelIndex) && levelIndex <  
levels.length) {  
        unlocked.push(levelIndex);  
        localStorage.setItem('unlockedLevels',  
JSON.stringify(unlocked));  
    }  
}
```

```
function isLevelUnlocked(levelIndex) {  
    return getUnlockedLevels().includes(levelIndex);  
}
```

```
function markLevelCompleted(levelIndex) {  
    const completed = getCompletedLevels();  
    if (!completed.includes(levelIndex)) {  
        completed.push(levelIndex);  
        localStorage.setItem('completedLevels',  
JSON.stringify(completed));  
    }  
}
```

```
function getCompletedLevels() {  
    const completed = localStorage.getItem('completedLevels');  
    return completed ? JSON.parse(completed) : [];  
}
```

```
function isLevelCompleted(levelIndex) {  
    return getCompletedLevels().includes(levelIndex);  
}
```

```
// Coin functions
```

```
function getCoins() {
```

```

    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;
    });
}

```

```

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;

```

```

    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

```

```

    // Check if this position would overlap with any nearby
    pipes
    let tooCloseToPipe = false;
    pipes.forEach(pipe => {
        const distance = Math.abs(pipe.x - canvas.width);
        // Only check pipes that are close to spawn point
        if (distance < 100) {
            // Check if coin would be in pipe area
            if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                tooCloseToPipe = true;
            }
        }
    });

    if (!tooCloseToPipe) {
        safePosition = true;
    }
    attempts;
}

// If we couldn't find a safe position, use center
if (!safePosition) {
    coinY = canvas.height / 2;
}

coins.push({
    x: canvas.width,
    y: coinY,
    size: coinSize,
    collected: false,
    rotation: 0
});
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);

```



```

    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

```

```

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';

```

```

        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position

```

```

ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

```

```

    }
  }
}

```

```

// Update coins
function updateCoins() {
  if (gameState === 'playing') {
    // Spawn coins periodically
    coinSpawnTimer;
    if (coinSpawnTimer >= coinSpawnInterval) {
      // Only spawn coins if we haven't reached the pipe
      limit
      if (pipesSpawned < totalPipes || pipes.length > 0)
      {
        createCoin();
      }
      coinSpawnTimer = 0;
    }

    // Move coins
    coins.forEach(coin => {
      if (!coin.collected) {
        coin.x -= pipeSpeed;
      }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
      if (coins[i].collected || coins[i].x +
      coins[i].size < 0) {
        coins.splice(i, 1);
      }
    }
  }
}

```

```

// Collision detection
function checkCollisions() {
  if (gameState !== 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&

```

```

        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {
                coin.collected = true;
                addCoin();
            }
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
    }
}

```

```

        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverElement.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');
            return;
        }

        // Remove next 2-3 pipes and move them to the back
        const pipesToRemove = Math.min(3, pipes.length);
        const removedPipes = [];

        // Remove pipes that are ahead of the bird
        for (let i = pipes.length - 1; i >= 0; i--) {
            if (pipes[i].x > bird.x && removedPipes.length <

```

```

pipesToRemove) {
    removedPipes.push(pipes.splice(i, 1)[0]);
}

// Move removed pipes to the back (far right)
removedPipes.forEach(pipe => {
    pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
    pipe.passed = false; // Reset passed status
    pipes.push(pipe);
});

// Reset bird to middle of screen
bird.y = canvas.height / 2;
bird.velocity = 0;

// Clear collected coins that are off-screen
coins.forEach(coin => {
    if (coin.x < bird.x - 100) {
        coin.collected = true;
    }
});

// Resume game
gameState = 'playing';
gameOverPopup.classList.add('hidden');
notEnoughCoinsElement.classList.add('hidden');
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');
    }
}

```



```

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);

```

```

canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Adding button event listeners and fixing the completeLevel function:

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
```

```

        pipeGap: 160,
        pipeSpawnInterval: 130,
        totalPipes: 8,
        description: "Medium – Pass 8 pipes"
    },
    {
        name: "Level 3",
        pipeSpeed: 3,
        pipeGap: 150,
        pipeSpawnInterval: 120,
        totalPipes: 12,
        description: "Hard – Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;

```

```

let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {

```

```

        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {

```

```

        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

```

```

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');

```



```

}

// Initialize first pipe
function createPipe() {
  const minHeight = 50;
  const maxHeight = canvas.height - pipeGap - minHeight;
  const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
  // Find a safe Y position that avoids pipes
  const coinSize = 20;
  let coinY;
  let attempts = 0;
  let safePosition = false;

  // Try to find a safe position
  while (!safePosition && attempts < 10) {
    coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

    // Check if this position would overlap with any nearby
pipes
    let tooCloseToPipe = false;
    pipes.forEach(pipe => {
      const distance = Math.abs(pipe.x - canvas.width);
      // Only check pipes that are close to spawn point
      if (distance < 100) {
        // Check if coin would be in pipe area
        if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
          tooCloseToPipe = true;
        }
      }
    });
  }
}

```

```

        if (!tooCloseToPipe) {
            safePosition = true;
        }
        attempts++;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX == null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

// Draw bird
function drawBird() {

```

```

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
    });
}

```

```
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,  
20);  
    });  
}
```

```
// Draw coins  
function drawCoins() {  
    coins.forEach(coin => {  
        if (!coin.collected) {  
            ctx.save();  
            ctx.translate(coin.x + coin.size / 2, coin.y +  
coin.size / 2);  
            coin.rotation += 0.1;  
            ctx.rotate(coin.rotation);  
  
            // Draw coin (golden circle)  
            ctx.fillStyle = '#FFD700';  
            ctx.beginPath();  
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);  
            ctx.fill();  
  
            // Draw coin border  
            ctx.strokeStyle = '#FFA500';  
            ctx.lineWidth = 2;  
            ctx.stroke();  
  
            // Draw $ symbol  
            ctx.fillStyle = '#FF8C00';  
            ctx.font = `${coin.size * 0.6}px Arial`;  
            ctx.textAlign = 'center';  
            ctx.textBaseline = 'middle';  
            ctx.fillText('$', 0, 0);  
  
            ctx.restore();  
        }  
    });  
}
```

```
// Update bird  
function updateBird() {  
    if (gameState === 'playing') {  
        bird.velocity += bird.gravity;  
        bird.y += bird.velocity;
```

```

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
        bird.y = canvas.height - bird.height;
        gameOver();
    }
    if (bird.y < 0) {
        bird.y = 0;
        bird.velocity = 0;
    }
}

// Update pipes
function updatePipes() {
    if (gameState === 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned++;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX === null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }
    }
}

```

```

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
{

```

```

        createCoin();
    }
    coinSpawnTimer = 0;
}

// Move coins
coins.forEach(coin => {
    if (!coin.collected) {
        coin.x -= pipeSpeed;
    }
});

// Remove off-screen or collected coins
for (let i = coins.length - 1; i >= 0; i--) {
    if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
        coins.splice(i, 1);
    }
}

}

}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
bird.x + bird.width > pipe.x &&
bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
bird.x + bird.width > pipe.x &&
bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    }
});

// Check coin collisions

```

```

coins.forEach(coin => {
  if (!coin.collected) {
    const coinCenterX = coin.x + coin.size / 2;
    const coinCenterY = coin.y + coin.size / 2;
    const birdCenterX = bird.x + bird.width / 2;
    const birdCenterY = bird.y + bird.height / 2;

    const distance = Math.sqrt(
      Math.pow(coinCenterX - birdCenterX, 2) +
      Math.pow(coinCenterY - birdCenterY, 2)
    );

    // Check if bird is close enough to collect coin
    if (distance < (coin.size / 2 + bird.width / 2)) {
      coin.collected = true;
      addCoin();
    }
  }
});
}

```

```

// Game over
function gameOver() {
  if (gameState === 'playing') {
    gameState = 'gameOver';
    gameOverPopup.classList.remove('hidden');
    notEnoughCoinsElement.classList.add('hidden');

    // Update continue button based on available coins
    const hasEnoughCoins = getCoins() >= 10;
    continueBtn.disabled = !hasEnoughCoins;
    if (!hasEnoughCoins) {
      continueBtn.style.opacity = '0.5';
      continueBtn.style.cursor = 'not-allowed';
    } else {
      continueBtn.style.opacity = '1';
      continueBtn.style.cursor = 'pointer';
    }
  }
}

```

```

// Complete level
function completeLevel() {
  if (gameState === 'playing') {

```



```

    gameState = 'levelComplete';
    levelCompleteElement.classList.remove('hidden');
    gameOverPopup.classList.add('hidden');

    // Mark level as completed
    markLevelCompleted(currentLevelIndex);

    // Unlock next level
    if (currentLevelIndex + 1 < levels.length) {
        unlockLevel(currentLevelIndex + 1);
    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');
            return;
        }

        // Remove next 2-3 pipes and move them to the back
        const pipesToRemove = Math.min(3, pipes.length);
        const removedPipes = [];

        // Remove pipes that are ahead of the bird
        for (let i = pipes.length - 1; i >= 0; i--) {
            if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
                removedPipes.push(pipes.splice(i, 1)[0]);
            }
        }

        // Move removed pipes to the back (far right)
        removedPipes.forEach(pipe => {
            pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
            pipe.passed = false; // Reset passed status
            pipes.push(pipe);
        });

        // Reset bird to middle of screen

```

```

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Clear collected coins that are off-screen
    coins.forEach(coin => {
        if (coin.x < bird.x - 100) {
            coin.collected = true;
        }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
    notEnoughCoinsElement.classList.add('hidden');
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

```

```

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', _jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}

```

```

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    updateCoins();
    checkCollisions();

    drawPipes();
    drawCoins();
    drawFinishLine();
    drawBird();
}

requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Adding event listeners for the popup buttons:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');

```

```
const continueBtn = document.getElementById('continueBtn');  
const notEnoughCoinsElement =  
document.getElementById('notEnoughCoins');  
const levelInfoElement = document.getElementById('levelInfo');  
const progressElement = document.getElementById('progress');  
const coinsElement = document.getElementById('coins');  
const totalCoinsElement =  
document.getElementById('totalCoins');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
```

```
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy - Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium - Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,  
    description: "Hard - Pass 12 pipes"  
  },  
  {  
    name: "Level 4",
```

```

        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,

```

```

    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
  };

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
  const completed = getCompletedLevels();
  if (!completed.includes(levelIndex)) {
    completed.push(levelIndex);
    localStorage.setItem('completedLevels',

```

```

JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

```



```

    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;

```

```

    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
    gameOverElement.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

```

```
}
```

```
// Create coin
```

```
function createCoin() {
```

```
    // Find a safe Y position that avoids pipes
```

```
    const coinSize = 20;
```

```
    let coinY;
```

```
    let attempts = 0;
```

```
    let safePosition = false;
```

```
    // Try to find a safe position
```

```
    while (!safePosition && attempts < 10) {
```

```
        coinY = 50 + Math.random() * (canvas.height - 100); //  
Random Y, avoiding top/bottom edges
```

```
        // Check if this position would overlap with any nearby  
pipes
```

```
        let tooCloseToPipe = false;
```

```
        pipes.forEach(pipe => {
```

```
            const distance = Math.abs(pipe.x - canvas.width);
```

```
            // Only check pipes that are close to spawn point
```

```
            if (distance < 100) {
```

```
                // Check if coin would be in pipe area
```

```
                if (coinY < pipe.topHeight || coinY + coinSize  
> pipe.bottomY) {
```

```
                    tooCloseToPipe = true;
```

```
                }
```

```
            }
```

```
        });
```

```
        if (!tooCloseToPipe) {
```

```
            safePosition = true;
```

```
        }
```

```
        attempts;
```

```
    }
```

```
    // If we couldn't find a safe position, use center
```

```
    if (!safePosition) {
```

```
        coinY = canvas.height / 2;
```

```
    }
```

```
    coins.push({
```

```
        x: canvas.width,
```

```
        y: coinY,
```

```

        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();

```

```

    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

```

```

        // Draw coin (golden circle)
        ctx.fillStyle = '#FFD700';
        ctx.beginPath();
        ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
        ctx.fill();

        // Draw coin border
        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {

```

```

// Spawn new pipes until we reach the total
if (pipesSpawned < totalPipes) {
    pipeSpawnTimer;
    if (pipeSpawnTimer >= pipeSpawnInterval) {
        createPipe();
        pipesSpawned;
        pipeSpawnTimer = 0;
    }
}

// Show finish line when all pipes are passed
if (score >= totalPipes && finishLineX = null) {
    // Find the rightmost pipe position and place
    finish line after it
    let rightmostX = bird.x + 200; // Default position
    ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
    bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed – complete
        level immediately

```

```

        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX ! null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });

        // Remove off-screen or collected coins
        for (let i = coins.length - 1; i >= 0; i--) {
            if (coins[i].collected || coins[i].x +

```



```
coins[i].size < 0) {
    coins.splice(i, 1);
}
}
}
```

// Collision detection

```
function checkCollisions() {
    if (gameState !== 'playing') return;
```

// Check pipe collisions

```
pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
        gameOver();
    }
```

// Check collision with bottom pipe

```
if (bird.x < pipe.x + pipeWidth &&
    bird.x + bird.width > pipe.x &&
    bird.y + bird.height > pipe.bottomY) {
    gameOver();
}
```

```
});
```

// Check coin collisions

```
coins.forEach(coin => {
    if (!coin.collected) {
        const coinCenterX = coin.x + coin.size / 2;
        const coinCenterY = coin.y + coin.size / 2;
        const birdCenterX = bird.x + bird.width / 2;
        const birdCenterY = bird.y + bird.height / 2;
```

```
const distance = Math.sqrt(
    Math.pow(coinCenterX - birdCenterX, 2) +
    Math.pow(coinCenterY - birdCenterY, 2)
);
```

// Check if bird is close enough to collect coin

```
if (distance < (coin.size / 2 + bird.width / 2)) {
    coin.collected = true;
```

```

        addCoin();
    }
}

});
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Continue game (revive)

```

```

function continueGame() {
  if (gameState = 'gameOver') {
    // Check if user has enough coins
    if (!spendCoins(10)) {
      notEnoughCoinsElement.classList.remove('hidden');
      return;
    }

    // Remove next 2-3 pipes and move them to the back
    const pipesToRemove = Math.min(3, pipes.length);
    const removedPipes = [];

    // Remove pipes that are ahead of the bird
    for (let i = pipes.length - 1; i >= 0; i--) {
      if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
        removedPipes.push(pipes.splice(i, 1)[0]);
      }
    }

    // Move removed pipes to the back (far right)
    removedPipes.forEach(pipe => {
      pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
      pipe.passed = false; // Reset passed status
      pipes.push(pipe);
    });

    // Reset bird to middle of screen
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Clear collected coins that are off-screen
    coins.forEach(coin => {
      if (coin.x < bird.x - 100) {
        coin.collected = true;
      }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
    notEnoughCoinsElement.classList.add('hidden');
  }
}

```

```

}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {

```

```

        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click', resetGame);
continueBtn.addEventListener('click', continueGame);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
    }
}

```

```

        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Adding CSS for the popup:

```

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

body {
    font-family: 'Arial', sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: 20px;
}

.container {
    text-align: center;

```

```
    background: white;
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

h1 {
    color: #333;
    margin-bottom: 15px;
    font-size: 2.5em;
}

#gameCanvas {
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB
70%, #90EE90 70%, #90EE90 100%);
    display: block;
    margin: 0 auto;
}

.instructions {
    margin-top: 15px;
    color: #555;
}

.instructions p {
    margin: 8px 0;
    font-size: 1.1em;
}

#score {
    font-size: 1.5em;
    font-weight: bold;
    color: #667eea;
}

#gameOver {
    font-size: 1.3em;
    font-weight: bold;
    color: #e74c3c;
    margin-top: 10px;
}
```

```
.hidden {
    display: none;
}

/* Level Selection Styles */
.level-selection {
    padding: 20px;
}

.level-selection h2 {
    color: #333;
    margin-bottom: 20px;
    font-size: 2em;
}

.level-buttons {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px,
1fr));
    gap: 15px;
    margin-bottom: 20px;
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}

.level-button {
    padding: 20px;
    font-size: 1.5em;
    font-weight: bold;
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    color: white;
    cursor: pointer;
    transition: all 0.3s;
}

.level-button:hover:not(.locked) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}
```



```
.level-button.locked {
    background: #ccc;
    color: #888;
    cursor: not-allowed;
    opacity: 0.6;
}

.level-button.completed {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d
100%);
}

.level-instructions {
    color: #666;
    font-size: 1em;
    margin-top: 15px;
}

.coin-display {
    font-size: 1.5em;
    font-weight: bold;
    margin-bottom: 20px;
    padding: 10px;
    background: linear-gradient(135deg, #FFD700 0%, #FFA500
100%);
    border-radius: 10px;
    color: #333;
    display: inline-block;
}

#coins {
    font-size: 1.2em;
    font-weight: bold;
    color: #FFD700;
}

#levelInfo {
    font-size: 1.3em;
    font-weight: bold;
    color: #764ba2;
}

#progress {
```

```

    font-size: 1.2em;
    font-weight: bold;
    color: #e67e22;
}

#levelComplete {
    font-size: 1.3em;
    font-weight: bold;
    color: #27ae60;
    margin-top: 10px;
}

.back-button {
    margin-top: 15px;
    padding: 10px 20px;
    font-size: 1em;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
}

.back-button:hover {
    background: #5568d3;
}

/* Game Over Popup Styles */
.popup-overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.7);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
}

.popup-content {
    background: white;

```

```

padding: 30px;
border-radius: 15px;
box-shadow: 0 10px 30px rgba(0, 0, 0, 0.5);
text-align: center;
max-width: 400px;
width: 90%;
}

.popup-content h2 {
color: #e74c3c;
margin-bottom: 15px;
font-size: 2em;
}

.popup-content p {
color: #555;
margin-bottom: 20px;
font-size: 1.1em;
}

.popup-buttons {
display: flex;
flex-direction: column;
gap: 15px;
margin-top: 20px;
}

.popup-button {
padding: 15px 30px;
font-size: 1.2em;
font-weight: bold;
border: none;
border-radius: 10px;
cursor: pointer;
transition: all 0.3s;
color: white;
}

.play-again-btn {
background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
}

.play-again-btn:hover {

```

```

    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(102, 126, 234, 0.4);
}

.continue-btn {
    background: linear-gradient(135deg, #FFD700 0%, #FFA500
100%);
    color: #333;
}

.continue-btn:hover:not(:disabled) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(255, 215, 0, 0.4);
}

.continue-btn:disabled {
    opacity: 0.5;
    cursor: not-allowed;
}

.error-message {
    color: #e74c3c;
    font-size: 0.9em;
    margin-top: 10px;
    font-weight: bold;
}

```

Verifying the bird starts in the middle of the screen. Checking the startLevel function:

Fixing the reference to the removed gameOverElement:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =

```

```
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy – Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium – Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard – Pass 12 pipes"
  }
]
```

```

    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard – Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties

```

```

const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
  const completed = getCompletedLevels();

```

```

        if (!completed.includes(levelIndex)) {
            completed.push(levelIndex);
            localStorage.setItem('completedLevels',
JSON.stringify(completed));
        }
    }

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
}

```



```

    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
    menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

```

```

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverPopup.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

    pipes.push({
        x: canvas.width,
        topHeight: topHeight,

```

```

        bottomY: topHeight + pipeGap,
        passed: false
    });
}

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

        // Check if this position would overlap with any nearby
pipes
        let tooCloseToPipe = false;
        pipes.forEach(pipe => {
            const distance = Math.abs(pipe.x - canvas.width);
            // Only check pipes that are close to spawn point
            if (distance < 100) {
                // Check if coin would be in pipe area
                if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                    tooCloseToPipe = true;
                }
            }
        });

        if (!tooCloseToPipe) {
            safePosition = true;
        }
        attempts++;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }
}

```

```

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX == null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();
}

```

```

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);

```

```

    coin.rotation += 0.1;
    ctx.rotate(coin.rotation);

    // Draw coin (golden circle)
    ctx.fillStyle = '#FFD700';
    ctx.beginPath();
    ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
    ctx.fill();

    // Draw coin border
    ctx.strokeStyle = '#FFA500';
    ctx.lineWidth = 2;
    ctx.stroke();

    // Draw $ symbol
    ctx.fillStyle = '#FF8C00';
    ctx.font = `${coin.size * 0.6}px Arial`;
    ctx.textAlign = 'center';
    ctx.textBaseline = 'middle';
    ctx.fillText('$', 0, 0);

    ctx.restore();
  }
});
}

// Update bird
function updateBird() {
  if (gameState === 'playing') {
    bird.velocity += bird.gravity;
    bird.y += bird.velocity;

    // Check boundaries
    if (bird.y + bird.height > canvas.height) {
      bird.y = canvas.height - bird.height;
      gameOver();
    }
    if (bird.y < 0) {
      bird.y = 0;
      bird.velocity = 0;
    }
  }
}

```

```

// Update pipes
function updatePipes() {
  if (gameState === 'playing') {
    // Spawn new pipes until we reach the total
    if (pipesSpawned < totalPipes) {
      pipeSpawnTimer;
      if (pipeSpawnTimer >= pipeSpawnInterval) {
        createPipe();
        pipesSpawned++;
        pipeSpawnTimer = 0;
      }
    }

    // Show finish line when all pipes are passed
    if (score >= totalPipes && finishLineX === null) {
      // Find the rightmost pipe position and place
      finish line after it
      let rightmostX = bird.x + 200; // Default position
      ahead of bird
      if (pipes.length > 0) {
        pipes.forEach(pipe => {
          if (pipe.x + pipeWidth > rightmostX) {
            rightmostX = pipe.x + pipeWidth;
          }
        });
        finishLineX = rightmostX + 150;
      } else {
        // If no pipes left, place finish line ahead of
        bird
        finishLineX = bird.x + 200;
      }
    }

    // Move pipes
    pipes.forEach(pipe => {
      pipe.x -= pipeSpeed;

      // Check if bird passed the pipe
      if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $

```

```

{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX ! null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
{
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });
    };
}

```



```

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
        if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
            coins.splice(i, 1);
        }
    }
}

```

```

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

```

```

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }
    }

```

```

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
        gameOver();
    }
});

```

```

// Check coin collisions
coins.forEach(coin => {
    if (!coin.collected) {
        const coinCenterX = coin.x + coin.size / 2;
        const coinCenterY = coin.y + coin.size / 2;
        const birdCenterX = bird.x + bird.width / 2;
        const birdCenterY = bird.y + bird.height / 2;

        const distance = Math.sqrt(
            Math.pow(coinCenterX - birdCenterX, 2) +
            Math.pow(coinCenterY - birdCenterY, 2)
        );
    }
}

```

```

        // Check if bird is close enough to collect coin
        if (distance < (coin.size / 2 + bird.width / 2)) {
            coin.collected = true;
            addCoin();
        }
    }
});
}

```

```

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

```

```

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

```

```

    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');
            return;
        }

        // Remove next 2-3 pipes and move them to the back
        const pipesToRemove = Math.min(3, pipes.length);
        const removedPipes = [];

        // Remove pipes that are ahead of the bird
        for (let i = pipes.length - 1; i >= 0; i--) {
            if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
                removedPipes.push(pipes.splice(i, 1)[0]);
            }
        }

        // Move removed pipes to the back (far right)
        removedPipes.forEach(pipe => {
            pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
            pipe.passed = false; // Reset passed status
            pipes.push(pipe);
        });

        // Reset bird to middle of screen
        bird.y = canvas.height / 2;
        bird.velocity = 0;

        // Clear collected coins that are off-screen
        coins.forEach(coin => {
            if (coin.x < bird.x - 100) {
                coin.collected = true;
            }
        });

        // Resume game
    }
}

```

```

        gameState = 'playing';
        gameOverPopup.classList.add('hidden');
        notEnoughCoinsElement.classList.add('hidden');
    }
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked levels
}

// Jump function

```

```

function jump() {
    if (gameState === 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState === 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click', resetGame);
continueBtn.addEventListener('click', continueGame);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
    }
}

```

```

    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    updateCoins();
    checkCollisions();

    drawPipes();
    drawCoins();
    drawFinishLine();
    drawBird();
}

requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Improving the continue function to ensure pipes are properly sorted:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =

```

```
document.getElementById('totalCoins');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
```

```
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy - Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium - Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,  
    description: "Hard - Pass 12 pipes"  
  },  
  {  
    name: "Level 4",  
    pipeSpeed: 3.5,  
    pipeGap: 140,  
    pipeSpawnInterval: 110,  
    totalPipes: 18,  
    description: "Very Hard - Pass 18 pipes"  
  },  
]
```

```

        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert - Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

```



```

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
    (first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

```

```

}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing

```

menu

```
levels.forEach((level, index) => {
    const button = document.createElement('button');
    button.className = 'level-button';
    button.textContent = index + 1;
    button.title = level.description;

    if (!isLevelUnlocked(index)) {
        button.classList.add('locked');
    } else if (isLevelCompleted(index)) {
        button.classList.add('completed');
    }

    if (isLevelUnlocked(index)) {
        button.addEventListener('click', () =>
startLevel(index));
    }

    levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverPopup.classList.add('hidden');
```

```

    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;

```

```

    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

        // Check if this position would overlap with any nearby
pipes
        let tooCloseToPipe = false;
        pipes.forEach(pipe => {
            const distance = Math.abs(pipe.x - canvas.width);
            // Only check pipes that are close to spawn point
            if (distance < 100) {
                // Check if coin would be in pipe area
                if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                    tooCloseToPipe = true;
                }
            }
        });

        if (!tooCloseToPipe) {
            safePosition = true;
        }
        attempts;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line

```

```

function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();

```

```

        ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
        ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
        ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
        ctx.closePath();
        ctx.fill();
    }

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border

```

```

        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }
    }
}

```



```

    }
}

// Show finish line when all pipes are passed
if (score >= totalPipes && finishLineX = null) {
    // Find the rightmost pipe position and place
finish line after it
    let rightmostX = bird.x + 200; // Default position
ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)

```

```

    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            {
                if (pipesSpawned < totalPipes || pipes.length > 0)
                {
                    createCoin();
                }
                coinSpawnTimer = 0;
            }

            // Move coins
            coins.forEach(coin => {
                if (!coin.collected) {
                    coin.x -= pipeSpeed;
                }
            });

            // Remove off-screen or collected coins
            for (let i = coins.length - 1; i >= 0; i--) {
                if (coins[i].collected || coins[i].x +
                coins[i].size < 0) {
                    coins.splice(i, 1);
                }
            }
        }
    }
}

```

```

// Collision detection
function checkCollisions() {
  if (gameState !== 'playing') return;

  // Check pipe collisions
  pipes.forEach(pipe => {
    // Check collision with top pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
      gameOver();
    }

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
      gameOver();
    }
  });

  // Check coin collisions
  coins.forEach(coin => {
    if (!coin.collected) {
      const coinCenterX = coin.x + coin.size / 2;
      const coinCenterY = coin.y + coin.size / 2;
      const birdCenterX = bird.x + bird.width / 2;
      const birdCenterY = bird.y + bird.height / 2;

      const distance = Math.sqrt(
        Math.pow(coinCenterX - birdCenterX, 2) +
        Math.pow(coinCenterY - birdCenterY, 2)
      );

      // Check if bird is close enough to collect coin
      if (distance < (coin.size / 2 + bird.width / 2)) {
        coin.collected = true;
        addCoin();
      }
    }
  });
}

```

```

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');

```

```

        return;
    }

    // Remove next 2-3 pipes and move them to the back
    const pipesToRemove = Math.min(3, pipes.length);
    const removedPipes = [];

    // Remove pipes that are ahead of the bird
    for (let i = pipes.length - 1; i >= 0; i--) {
        if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
            removedPipes.push(pipes.splice(i, 1)[0]);
        }
    }

    // Move removed pipes to the back (far right)
    removedPipes.forEach(pipe => {
        pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
        pipe.passed = false; // Reset passed status
        pipes.push(pipe);
    });

    // Sort pipes by x position to maintain order
    pipes.sort((a, b) => a.x - b.x);

    // Reset bird to middle of screen
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Clear collected coins that are off-screen
    coins.forEach(coin => {
        if (coin.x < bird.x - 100) {
            coin.collected = true;
        }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
    notEnoughCoinsElement.classList.add('hidden');
}
}

```

```

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
}

```

```

    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code === 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click', resetGame);
continueBtn.addEventListener('click', continueGame);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
    }
}

```

```

        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();

```

Features Added:

1. Game Over Popup:

- Modal overlay appears when you die
- Two buttons: "Play Again" and "Continue (10 coins)"

2. Play Again Button:

- Restarts the level from the beginning
- Resets score, pipes, and bird position

3. Continue Button (10 coins):

- Costs 10 coins to continue
- Removes the next 2-3 pipes ahead of the bird
- Moves those pipes to the back of the sequence
- Resets bird to the middle of the screen (not on the ground)
- Gives you time to recover before the next obstacles

4. Smart Features:

- Button is disabled if you don't have enough coins
- Shows error message if you try to continue without enough coins
- Coins are deducted when you continue
- Pipes are properly sorted after being moved to the back

The popup appears when you die, and you can either restart or continue for 10 coins. When continuing, the next few pipes are removed and moved to the back, and the bird starts in the middle of the screen.

User

the pop up for death is appearing on the main menu, it should just appear while playing the game and after death

Cursor

Fixing the popup so it only appears during gameplay and after death, not on the main menu. Checking the code:

Ensuring the popup is hidden when returning to the menu and when starting a level:

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
```

```
const backButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard - Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert - Pass 25 pipes"
  },
]
```

```

    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];

```

```

let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions

```

```

function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;
    });
}

```

```

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverPopup.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes

```

```

    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

```

```

        // Check if this position would overlap with any nearby
pipes
        let tooCloseToPipe = false;
        pipes.forEach(pipe => {
            const distance = Math.abs(pipe.x - canvas.width);
            // Only check pipes that are close to spawn point
            if (distance < 100) {
                // Check if coin would be in pipe area
                if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                    tooCloseToPipe = true;
                }
            }
        });

        if (!tooCloseToPipe) {
            safePosition = true;
        }
        attempts++;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();

```



```

    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();

```

```

}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;

```

```

        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it

```

```

        let rightmostX = bird.x + 200; // Default position
ahead of bird
        if (pipes.length > 0) {
            pipes.forEach(pipe => {
                if (pipe.x + pipeWidth > rightmostX) {
                    rightmostX = pipe.x + pipeWidth;
                }
            });
            finishLineX = rightmostX + 150;
        } else {
            // If no pipes left, place finish line ahead of
bird
            finishLineX = bird.x + 200;
        }
    }

    // Move pipes
    pipes.forEach(pipe => {
        pipe.x -= pipeSpeed;

        // Check if bird passed the pipe
        if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
            pipe.passed = true;
            score;
            scoreElement.textContent = `Score: ${score}`;
            progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

            // Check if all pipes are passed - complete
level immediately
            if (score >= totalPipes) {
                completeLevel();
            }
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{

```

```

        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer++;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
            {
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });

        // Remove off-screen or collected coins
        for (let i = coins.length - 1; i >= 0; i--) {
            if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
                coins.splice(i, 1);
            }
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState != 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe

```

```

    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
        gameOver();
    }

    // Check collision with bottom pipe
    if (bird.x < pipe.x + pipeWidth &&
        bird.x + bird.width > pipe.x &&
        bird.y + bird.height > pipe.bottomY) {
        gameOver();
    }
});

// Check coin collisions
coins.forEach(coin => {
    if (!coin.collected) {
        const coinCenterX = coin.x + coin.size / 2;
        const coinCenterY = coin.y + coin.size / 2;
        const birdCenterX = bird.x + bird.width / 2;
        const birdCenterY = bird.y + bird.height / 2;

        const distance = Math.sqrt(
            Math.pow(coinCenterX - birdCenterX, 2) +
            Math.pow(coinCenterY - birdCenterY, 2)
        );

        // Check if bird is close enough to collect coin
        if (distance < (coin.size / 2 + bird.width / 2)) {
            coin.collected = true;
            addCoin();
        }
    }
});
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins

```

```

const hasEnoughCoins = getCoins() >= 10;
continueBtn.disabled = !hasEnoughCoins;
if (!hasEnoughCoins) {
    continueBtn.style.opacity = '0.5';
    continueBtn.style.cursor = 'not-allowed';
} else {
    continueBtn.style.opacity = '1';
    continueBtn.style.cursor = 'pointer';
}
}
}

```

```

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

```

```

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');
            return;
        }

        // Remove next 2-3 pipes and move them to the back
        const pipesToRemove = Math.min(3, pipes.length);
        const removedPipes = [];

        // Remove pipes that are ahead of the bird
        for (let i = pipes.length - 1; i >= 0; i--) {

```

```

        if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
            removedPipes.push(pipes.splice(i, 1)[0]);
        }
    }

    // Move removed pipes to the back (far right)
    removedPipes.forEach(pipe => {
        pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
        pipe.passed = false; // Reset passed status
        pipes.push(pipe);
    });

    // Sort pipes by x position to maintain order
    pipes.sort((a, b) => a.x - b.x);

    // Reset bird to middle of screen
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Clear collected coins that are off-screen
    coins.forEach(coin => {
        if (coin.x < bird.x - 100) {
            coin.collected = true;
        }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
    notEnoughCoinsElement.classList.add('hidden');
}

// Reset game (play again)
function resetGame() {
    if (gameState === 'gameOver' || gameState === 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
    }
}

```



```

        progressElement.textContent = `Progress: ${score} / ${
totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    gameOverPopup.classList.add('hidden'); // Hide popup when
returning to menu
    notEnoughCoinsElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {

```

```

        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click',_jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click',_returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click',_resetGame);
continueBtn.addEventListener('click',_continueGame);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
    }
}

```

```
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
initLevelSelection();
gameLoop();
```

Checking CSS to ensure the popup is properly hidden:

Ensuring the popup is hidden on initialization and adding a safeguard:

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backToMenuButton = document.getElementById('backToMenu');
```

```

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard - Pass 18 pipes"
  },
  {
    name: "Level 5",
    pipeSpeed: 4,
    pipeGap: 130,
    pipeSpawnInterval: 100,
    totalPipes: 25,
    description: "Expert - Pass 25 pipes"
  }
]

```

```

        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
};

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;

```

```
const coinSpawnInterval = 80; // Spawn coins more frequently  
than pipes
```

```
// LocalStorage functions
```

```
function getUnlockedLevels() {  
    const unlocked = localStorage.getItem('unlockedLevels');  
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0  
(first level) is always unlocked  
}
```

```
function unlockLevel(levelIndex) {  
    const unlocked = getUnlockedLevels();  
    if (!unlocked.includes(levelIndex) && levelIndex <  
levels.length) {  
        unlocked.push(levelIndex);  
        localStorage.setItem('unlockedLevels',  
JSON.stringify(unlocked));  
    }  
}
```

```
function isLevelUnlocked(levelIndex) {  
    return getUnlockedLevels().includes(levelIndex);  
}
```

```
function markLevelCompleted(levelIndex) {  
    const completed = getCompletedLevels();  
    if (!completed.includes(levelIndex)) {  
        completed.push(levelIndex);  
        localStorage.setItem('completedLevels',  
JSON.stringify(completed));  
    }  
}
```

```
function getCompletedLevels() {  
    const completed = localStorage.getItem('completedLevels');  
    return completed ? JSON.parse(completed) : [];  
}
```

```
function isLevelCompleted(levelIndex) {  
    return getCompletedLevels().includes(levelIndex);  
}
```

```
// Coin functions
```

```
function getCoins() {
```

```

    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;
    });
}

```

```

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;
    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverPopup.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;

```



```

    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

```

```

    // Check if this position would overlap with any nearby
    pipes
    let tooCloseToPipe = false;
    pipes.forEach(pipe => {
        const distance = Math.abs(pipe.x - canvas.width);
        // Only check pipes that are close to spawn point
        if (distance < 100) {
            // Check if coin would be in pipe area
            if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                tooCloseToPipe = true;
            }
        }
    });

    if (!tooCloseToPipe) {
        safePosition = true;
    }
    attempts;
}

// If we couldn't find a safe position, use center
if (!safePosition) {
    coinY = canvas.height / 2;
}

coins.push({
    x: canvas.width,
    y: coinY,
    size: coinSize,
    collected: false,
    rotation: 0
});
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);

```

```

    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

```

```

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {
            ctx.save();
            ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
            coin.rotation += 0.1;
            ctx.rotate(coin.rotation);

            // Draw coin (golden circle)
            ctx.fillStyle = '#FFD700';
            ctx.beginPath();
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
            ctx.fill();

            // Draw coin border
            ctx.strokeStyle = '#FFA500';
            ctx.lineWidth = 2;
            ctx.stroke();

            // Draw $ symbol
            ctx.fillStyle = '#FF8C00';
            ctx.font = `${coin.size * 0.6}px Arial`;
            ctx.textAlign = 'center';

```

```

        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position

```

```

ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score++;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)
if (finishLineX != null) {
    finishLineX -= pipeSpeed;
}

// Remove off-screen pipes
while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
    pipes.shift();
}

```

```
    }  
  }  
}
```

```
// Update coins
```

```
function updateCoins() {
```

```
  if (gameState === 'playing') {
```

```
    // Spawn coins periodically
```

```
    coinSpawnTimer;
```

```
    if (coinSpawnTimer >= coinSpawnInterval) {
```

```
      // Only spawn coins if we haven't reached the pipe
```

```
limit
```

```
      if (pipesSpawned < totalPipes || pipes.length > 0)
```

```
{
```

```
        createCoin();
```

```
      }
```

```
      coinSpawnTimer = 0;
```

```
    }
```

```
    // Move coins
```

```
    coins.forEach(coin => {
```

```
      if (!coin.collected) {
```

```
        coin.x -= pipeSpeed;
```

```
      }
```

```
    });
```

```
    // Remove off-screen or collected coins
```

```
    for (let i = coins.length - 1; i >= 0; i--) {
```

```
      if (coins[i].collected || coins[i].x +
```

```
coins[i].size < 0) {
```

```
        coins.splice(i, 1);
```

```
      }
```

```
    }
```

```
  }
```

```
}
```

```
// Collision detection
```

```
function checkCollisions() {
```

```
  if (gameState !== 'playing') return;
```

```
  // Check pipe collisions
```

```
  pipes.forEach(pipe => {
```

```
    // Check collision with top pipe
```

```
    if (bird.x < pipe.x + pipeWidth &&
```

```

        bird.x + bird.width > pipe.x &&
        bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {
                coin.collected = true;
                addCoin();
            }
        }
    });
}

// Game over
function gameOver() {
    if (gameState === 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
    }
}

```



```

        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');
            return;
        }

        // Remove next 2-3 pipes and move them to the back
        const pipesToRemove = Math.min(3, pipes.length);
        const removedPipes = [];

        // Remove pipes that are ahead of the bird
        for (let i = pipes.length - 1; i >= 0; i--) {
            if (pipes[i].x > bird.x && removedPipes.length <

```

```

pipesToRemove) {
    removedPipes.push(pipes.splice(i, 1)[0]);
}

// Move removed pipes to the back (far right)
removedPipes.forEach(pipe => {
    pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
    pipe.passed = false; // Reset passed status
    pipes.push(pipe);
});

// Sort pipes by x position to maintain order
pipes.sort((a, b) => a.x - b.x);

// Reset bird to middle of screen
bird.y = canvas.height / 2;
bird.velocity = 0;

// Clear collected coins that are off-screen
coins.forEach(coin => {
    if (coin.x < bird.x - 100) {
        coin.collected = true;
    }
});

// Resume game
gameState = 'playing';
gameOverPopup.classList.add('hidden');
notEnoughCoinsElement.classList.add('hidden');
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $

```

```

{totalPipes}`;
    gameOverPopup.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    bird.y = canvas.height / 2;
    bird.velocity = 0;

    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;
}
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    gameOverPopup.classList.add('hidden'); // Hide popup when
returning to menu
    notEnoughCoinsElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
    }
});

```

```

        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click', resetGame);
continueBtn.addEventListener('click', continueGame);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }
}

```

```

    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
gameOverPopup.classList.add('hidden'); // Ensure popup is
hidden on startup
notEnoughCoinsElement.classList.add('hidden');
initLevelSelection();
gameLoop();

```

Verifying the popup CSS and ensuring it only shows when appropriate:

```

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

body {
    font-family: 'Arial', sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    padding: 20px;
}

.container {
    text-align: center;
    background: white;
    padding: 20px;
    border-radius: 15px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);
}

```

```
h1 {
    color: #333;
    margin-bottom: 15px;
    font-size: 2.5em;
}

#gameCanvas {
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(to bottom, #87CEEB 0%, #87CEEB
70%, #90EE90 70%, #90EE90 100%);
    display: block;
    margin: 0 auto;
}

.instructions {
    margin-top: 15px;
    color: #555;
}

.instructions p {
    margin: 8px 0;
    font-size: 1.1em;
}

#score {
    font-size: 1.5em;
    font-weight: bold;
    color: #667eea;
}

#gameOver {
    font-size: 1.3em;
    font-weight: bold;
    color: #e74c3c;
    margin-top: 10px;
}

.hidden {
    display: none;
}

/* Level Selection Styles */
```

```
.level-selection {
    padding: 20px;
}

.level-selection h2 {
    color: #333;
    margin-bottom: 20px;
    font-size: 2em;
}

.level-buttons {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(120px,
1fr));
    gap: 15px;
    margin-bottom: 20px;
    max-width: 500px;
    margin-left: auto;
    margin-right: auto;
}

.level-button {
    padding: 20px;
    font-size: 1.5em;
    font-weight: bold;
    border: 3px solid #333;
    border-radius: 10px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
    color: white;
    cursor: pointer;
    transition: all 0.3s;
}

.level-button:hover:not(.locked) {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}

.level-button.locked {
    background: #ccc;
    color: #888;
    cursor: not-allowed;
    opacity: 0.6;
}
```

```
}

.level-button.completed {
    background: linear-gradient(135deg, #11998e 0%, #38ef7d
100%);
}

.level-instructions {
    color: #666;
    font-size: 1em;
    margin-top: 15px;
}

.coin-display {
    font-size: 1.5em;
    font-weight: bold;
    margin-bottom: 20px;
    padding: 10px;
    background: linear-gradient(135deg, #FFD700 0%, #FFA500
100%);
    border-radius: 10px;
    color: #333;
    display: inline-block;
}

#coins {
    font-size: 1.2em;
    font-weight: bold;
    color: #FFD700;
}

#levelInfo {
    font-size: 1.3em;
    font-weight: bold;
    color: #764ba2;
}

#progress {
    font-size: 1.2em;
    font-weight: bold;
    color: #e67e22;
}

#levelComplete {
```



```
    font-size: 1.3em;
    font-weight: bold;
    color: #27ae60;
    margin-top: 10px;
}

.back-button {
    margin-top: 15px;
    padding: 10px 20px;
    font-size: 1em;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s;
}

.back-button:hover {
    background: #5568d3;
}

/* Game Over Popup Styles */
.popup-overlay {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.7);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 1000;
}

.popup-overlay.hidden {
    display: none !important;
}

.popup-content {
    background: white;
    padding: 30px;
    border-radius: 15px;
```

```
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.5);
    text-align: center;
    max-width: 400px;
    width: 90%;
}

.popup-content h2 {
    color: #e74c3c;
    margin-bottom: 15px;
    font-size: 2em;
}

.popup-content p {
    color: #555;
    margin-bottom: 20px;
    font-size: 1.1em;
}

.popup-buttons {
    display: flex;
    flex-direction: column;
    gap: 15px;
    margin-top: 20px;
}

.popup-button {
    padding: 15px 30px;
    font-size: 1.2em;
    font-weight: bold;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    transition: all 0.3s;
    color: white;
}

.play-again-btn {
    background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
}

.play-again-btn:hover {
    transform: scale(1.05);
    box-shadow: 0 5px 15px rgba(102, 126, 234, 0.4);
}
```

```

}

.continue-btn {
  background: linear-gradient(135deg, #FFD700 0%, #FFA500
100%);
  color: #333;
}

.continue-btn:hover:not(:disabled) {
  transform: scale(1.05);
  box-shadow: 0 5px 15px rgba(255, 215, 0, 0.4);
}

.continue-btn:disabled {
  opacity: 0.5;
  cursor: not-allowed;
}

.error-message {
  color: #e74c3c;
  font-size: 0.9em;
  margin-top: 10px;
  font-weight: bold;
}

```

Fixed the popup issue. Changes:

1. **Hide popup when returning to menu:** The `returnToMenu()` function now explicitly hides the popup
2. **Hide popup on game initialization:** Ensures the popup is hidden when the game first loads
3. **CSS fix:** Added a specific rule to ensure the popup overlay is hidden when it has the "hidden" class

The popup now only appears:

- During gameplay when you die (gameState is 'gameOver')
- Not on the main menu
- Not when starting a new level

The popup is hidden when:

- Returning to the menu
- Starting a new level
- Game initializes
- Continuing after death

The popup should no longer appear on the main menu.

User

after I spent the coins to keep going the pop up didnt dissappear and it didnt return me back into the game

Cursor

Checking the `continueGame()` function to fix the issue:

Checking the CSS and ensuring the popup hides correctly. The issue may be CSS specificity. Verifying and fixing:

CSS looks correct. Ensuring the game screen stays visible and the popup hides. Adding a safeguard and verifying the continue function:

```
// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');
const levelCompleteElement =
document.getElementById('levelComplete');
const gameOverPopup = document.getElementById('gameOverPopup');
const playAgainBtn = document.getElementById('playAgainBtn');
const continueBtn = document.getElementById('continueBtn');
const notEnoughCoinsElement =
document.getElementById('notEnoughCoins');
const levelInfoElement = document.getElementById('levelInfo');
const progressElement = document.getElementById('progress');
const coinsElement = document.getElementById('coins');
const totalCoinsElement =
document.getElementById('totalCoins');
```

```
const levelSelectionElement =
document.getElementById('levelSelection');
const gameScreenElement =
document.getElementById('gameScreen');
const levelButtonsElement =
document.getElementById('levelButtons');
const backButton = document.getElementById('backToMenu');

// Level definitions
const levels = [
  {
    name: "Level 1",
    pipeSpeed: 2,
    pipeGap: 180,
    pipeSpawnInterval: 140,
    totalPipes: 5,
    description: "Easy - Pass 5 pipes"
  },
  {
    name: "Level 2",
    pipeSpeed: 2.5,
    pipeGap: 160,
    pipeSpawnInterval: 130,
    totalPipes: 8,
    description: "Medium - Pass 8 pipes"
  },
  {
    name: "Level 3",
    pipeSpeed: 3,
    pipeGap: 150,
    pipeSpawnInterval: 120,
    totalPipes: 12,
    description: "Hard - Pass 12 pipes"
  },
  {
    name: "Level 4",
    pipeSpeed: 3.5,
    pipeGap: 140,
    pipeSpawnInterval: 110,
    totalPipes: 18,
    description: "Very Hard - Pass 18 pipes"
  },
  {
    name: "Level 5",
```

```

        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert – Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master – Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

// Bird properties
const bird = {
    x: 50,
    y: canvas.height / 2,
    width: 30,
    height: 30,
    velocity: 0,
    gravity: 0.25,
    jumpStrength: -5,
    color: '#FFD700'
};

```

```

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
    const unlocked = localStorage.getItem('unlockedLevels');
    return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
    const unlocked = getUnlockedLevels();
    if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
        unlocked.push(levelIndex);
        localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
    }
}

function isLevelUnlocked(levelIndex) {
    return getUnlockedLevels().includes(levelIndex);
}

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

```

```

}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {
    const totalCoins = getCoins();
    if (totalCoinsElement) {
        totalCoinsElement.textContent = totalCoins;
    }
    if (coinsElement) {
        coinsElement.textContent = `💰 Coins: ${totalCoins}`;
    }
}

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing

```


menu

```
levels.forEach((level, index) => {
  const button = document.createElement('button');
  button.className = 'level-button';
  button.textContent = index + 1;
  button.title = level.description;

  if (!isLevelUnlocked(index)) {
    button.classList.add('locked');
  } else if (isLevelCompleted(index)) {
    button.classList.add('completed');
  }

  if (isLevelUnlocked(index)) {
    button.addEventListener('click', () =>
startLevel(index));
  }

  levelButtonsElement.appendChild(button);
});
}

// Start a level
function startLevel(levelIndex) {
  currentLevelIndex = levelIndex;
  currentLevel = levels[levelIndex];

  // Set level-specific settings
  pipeSpeed = currentLevel.pipeSpeed;
  pipeGap = currentLevel.pipeGap;
  pipeSpawnInterval = currentLevel.pipeSpawnInterval;
  totalPipes = currentLevel.totalPipes;

  // Reset game state
  gameState = 'playing';
  score = 0;
  pipesSpawned = 0;
  finishLineX = null;
  scoreElement.textContent = `Score: ${score}`;
  levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
  progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
  gameOverPopup.classList.add('hidden');
```

```

    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

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

// Create coin
function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;

```

```

    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
        coinY = 50 + Math.random() * (canvas.height - 100); //
Random Y, avoiding top/bottom edges

        // Check if this position would overlap with any nearby
pipes
        let tooCloseToPipe = false;
        pipes.forEach(pipe => {
            const distance = Math.abs(pipe.x - canvas.width);
            // Only check pipes that are close to spawn point
            if (distance < 100) {
                // Check if coin would be in pipe area
                if (coinY < pipe.topHeight || coinY + coinSize
> pipe.bottomY) {
                    tooCloseToPipe = true;
                }
            }
        });

        if (!tooCloseToPipe) {
            safePosition = true;
        }
        attempts++;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {
        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line

```

```

function drawFinishLine() {
    if (finishLineX == null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();

```

```
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);  
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2  
- 3);  
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2  
+ 3);  
    ctx.closePath();  
    ctx.fill();  
}
```

```
// Draw pipes  
function drawPipes() {  
    ctx.fillStyle = '#228B22';  
  
    pipes.forEach(pipe => {  
        // Top pipe  
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);  
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth  
+ 10, 20);  
  
        // Bottom pipe  
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,  
canvas.height - pipe.bottomY);  
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,  
20);  
    });  
}
```

```
// Draw coins  
function drawCoins() {  
    coins.forEach(coin => {  
        if (!coin.collected) {  
            ctx.save();  
            ctx.translate(coin.x + coin.size / 2, coin.y +  
coin.size / 2);  
            coin.rotation += 0.1;  
            ctx.rotate(coin.rotation);  
  
            // Draw coin (golden circle)  
            ctx.fillStyle = '#FFD700';  
            ctx.beginPath();  
            ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);  
            ctx.fill();  
  
            // Draw coin border
```

```

        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
    });
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }
    }
}

```

```

    }
}

// Show finish line when all pipes are passed
if (score >= totalPipes && finishLineX = null) {
    // Find the rightmost pipe position and place
finish line after it
    let rightmostX = bird.x + 200; // Default position
ahead of bird
    if (pipes.length > 0) {
        pipes.forEach(pipe => {
            if (pipe.x + pipeWidth > rightmostX) {
                rightmostX = pipe.x + pipeWidth;
            }
        });
        finishLineX = rightmostX + 150;
    } else {
        // If no pipes left, place finish line ahead of
bird
        finishLineX = bird.x + 200;
    }
}

// Move pipes
pipes.forEach(pipe => {
    pipe.x -= pipeSpeed;

    // Check if bird passed the pipe
    if (!pipe.passed && pipe.x + pipeWidth < bird.x) {
        pipe.passed = true;
        score;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    }
});

// Move finish line (visual indicator only)

```

```

    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
    {
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
            limit
            {
                if (pipesSpawned < totalPipes || pipes.length > 0)
                {
                    createCoin();
                }
                coinSpawnTimer = 0;
            }

            // Move coins
            coins.forEach(coin => {
                if (!coin.collected) {
                    coin.x -= pipeSpeed;
                }
            });

            // Remove off-screen or collected coins
            for (let i = coins.length - 1; i >= 0; i--) {
                if (coins[i].collected || coins[i].x +
                coins[i].size < 0) {
                    coins.splice(i, 1);
                }
            }
        }
    }
}

```



```

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +
                Math.pow(coinCenterY - birdCenterY, 2)
            );

            // Check if bird is close enough to collect coin
            if (distance < (coin.size / 2 + bird.width / 2)) {
                coin.collected = true;
                addCoin();
            }
        }
    });
}

```

```

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');

```

```

        return;
    }

    // Remove next 2-3 pipes and move them to the back
    const pipesToRemove = Math.min(3, pipes.length);
    const removedPipes = [];

    // Remove pipes that are ahead of the bird
    for (let i = pipes.length - 1; i >= 0; i--) {
        if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
            removedPipes.push(pipes.splice(i, 1)[0]);
        }
    }

    // Move removed pipes to the back (far right)
    removedPipes.forEach(pipe => {
        pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
        pipe.passed = false; // Reset passed status
        pipes.push(pipe);
    });

    // Sort pipes by x position to maintain order
    pipes.sort((a, b) => a.x - b.x);

    // Reset bird to middle of screen
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Clear collected coins that are off-screen
    coins.forEach(coin => {
        if (coin.x < bird.x - 100) {
            coin.collected = true;
        }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
    notEnoughCoinsElement.classList.add('hidden');

    // Ensure game screen is visible
    if (gameScreenElement) {

```

```

        gameScreenElement.classList.remove('hidden');
    }

    // Force hide popup with inline style as backup
    if (gameOverPopup) {
        gameOverPopup.style.display = 'none';
    }
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / ${totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    gameOverPopup.classList.add('hidden'); // Hide popup when

```

```

    returning to menu
        notEnoughCoinsElement.classList.add('hidden');
        initLevelSelection(); // Refresh to show newly unlocked
levels
    }

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click', resetGame);
continueBtn.addEventListener('click', continueGame);

// Game loop
function gameLoop() {
    if (gameState != 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);
    }
}

```

```

    // Draw background
    const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
    gradient.addColorStop(0, '#87CEEB');
    gradient.addColorStop(0.7, '#87CEEB');
    gradient.addColorStop(0.7, '#90EE90');
    gradient.addColorStop(1, '#90EE90');
    ctx.fillStyle = gradient;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Update and draw
    updateBird();
    updatePipes();
    updateCoins();
    checkCollisions();

    drawPipes();
    drawCoins();
    drawFinishLine();
    drawBird();
}

requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
gameOverPopup.classList.add('hidden'); // Ensure popup is
hidden on startup
notEnoughCoinsElement.classList.add('hidden');
initLevelSelection();
gameLoop();

```

Updating the [gameOver](#) function to set the display style, and ensuring [continueGame](#) removes it:

```

// Canvas setup
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
const scoreElement = document.getElementById('score');

```

```
const levelCompleteElement =  
document.getElementById('levelComplete');  
const gameOverPopup = document.getElementById('gameOverPopup');  
const playAgainBtn = document.getElementById('playAgainBtn');  
const continueBtn = document.getElementById('continueBtn');  
const notEnoughCoinsElement =  
document.getElementById('notEnoughCoins');  
const levelInfoElement = document.getElementById('levelInfo');  
const progressElement = document.getElementById('progress');  
const coinsElement = document.getElementById('coins');  
const totalCoinsElement =  
document.getElementById('totalCoins');  
const levelSelectionElement =  
document.getElementById('levelSelection');  
const gameScreenElement =  
document.getElementById('gameScreen');  
const levelButtonsElement =  
document.getElementById('levelButtons');  
const backToMenuButton = document.getElementById('backToMenu');
```

```
// Level definitions
```

```
const levels = [  
  {  
    name: "Level 1",  
    pipeSpeed: 2,  
    pipeGap: 180,  
    pipeSpawnInterval: 140,  
    totalPipes: 5,  
    description: "Easy - Pass 5 pipes"  
  },  
  {  
    name: "Level 2",  
    pipeSpeed: 2.5,  
    pipeGap: 160,  
    pipeSpawnInterval: 130,  
    totalPipes: 8,  
    description: "Medium - Pass 8 pipes"  
  },  
  {  
    name: "Level 3",  
    pipeSpeed: 3,  
    pipeGap: 150,  
    pipeSpawnInterval: 120,  
    totalPipes: 12,
```

```

        description: "Hard - Pass 12 pipes"
    },
    {
        name: "Level 4",
        pipeSpeed: 3.5,
        pipeGap: 140,
        pipeSpawnInterval: 110,
        totalPipes: 18,
        description: "Very Hard - Pass 18 pipes"
    },
    {
        name: "Level 5",
        pipeSpeed: 4,
        pipeGap: 130,
        pipeSpawnInterval: 100,
        totalPipes: 25,
        description: "Expert - Pass 25 pipes"
    },
    {
        name: "Level 6",
        pipeSpeed: 4.5,
        pipeGap: 120,
        pipeSpawnInterval: 90,
        totalPipes: 35,
        description: "Master - Pass 35 pipes"
    }
];

// Game state
let gameState = 'menu'; // 'menu', 'playing', 'gameOver',
'levelComplete'
let score = 0;
let currentLevel = null;
let currentLevelIndex = 0;

// Level settings (will be set based on selected level)
let pipeWidth = 60;
let pipeGap = 150;
let pipeSpeed = 3;
let pipeSpawnInterval = 120;
let totalPipes = 0;
let pipesSpawned = 0;
let finishLineX = null;

```



```

// Bird properties
const bird = {
  x: 50,
  y: canvas.height / 2,
  width: 30,
  height: 30,
  velocity: 0,
  gravity: 0.25,
  jumpStrength: -5,
  color: '#FFD700'
};

// Pipes array
const pipes = [];
let pipeSpawnTimer = 0;

// Coins array
const coins = [];
let coinSpawnTimer = 0;
const coinSpawnInterval = 80; // Spawn coins more frequently
than pipes

// LocalStorage functions
function getUnlockedLevels() {
  const unlocked = localStorage.getItem('unlockedLevels');
  return unlocked ? JSON.parse(unlocked) : [0]; // Level 0
(first level) is always unlocked
}

function unlockLevel(levelIndex) {
  const unlocked = getUnlockedLevels();
  if (!unlocked.includes(levelIndex) && levelIndex <
levels.length) {
    unlocked.push(levelIndex);
    localStorage.setItem('unlockedLevels',
JSON.stringify(unlocked));
  }
}

function isLevelUnlocked(levelIndex) {
  return getUnlockedLevels().includes(levelIndex);
}

```

```

function markLevelCompleted(levelIndex) {
    const completed = getCompletedLevels();
    if (!completed.includes(levelIndex)) {
        completed.push(levelIndex);
        localStorage.setItem('completedLevels',
JSON.stringify(completed));
    }
}

function getCompletedLevels() {
    const completed = localStorage.getItem('completedLevels');
    return completed ? JSON.parse(completed) : [];
}

function isLevelCompleted(levelIndex) {
    return getCompletedLevels().includes(levelIndex);
}

// Coin functions
function getCoins() {
    const coins = localStorage.getItem('totalCoins');
    return coins ? parseInt(coins) : 0;
}

function addCoin() {
    const currentCoins = getCoins();
    localStorage.setItem('totalCoins', (currentCoins +
1).toString());
    updateCoinDisplay();
}

function spendCoins(amount) {
    const currentCoins = getCoins();
    if (currentCoins >= amount) {
        localStorage.setItem('totalCoins', (currentCoins -
amount).toString());
        updateCoinDisplay();
        return true;
    }
    return false;
}

function updateCoinDisplay() {

```

```

const totalCoins = getCoins();
if (totalCoinsElement) {
    totalCoinsElement.textContent = totalCoins;
}
if (coinsElement) {
    coinsElement.textContent = `💰 Coins: ${totalCoins}`;
}
}

```

```

// Initialize level selection screen
function initLevelSelection() {
    levelButtonsElement.innerHTML = '';
    updateCoinDisplay(); // Update coin display when showing
menu

```

```

    levels.forEach((level, index) => {
        const button = document.createElement('button');
        button.className = 'level-button';
        button.textContent = index + 1;
        button.title = level.description;

        if (!isLevelUnlocked(index)) {
            button.classList.add('locked');
        } else if (isLevelCompleted(index)) {
            button.classList.add('completed');
        }

        if (isLevelUnlocked(index)) {
            button.addEventListener('click', () =>
startLevel(index));
        }

        levelButtonsElement.appendChild(button);
    });
}

```

```

// Start a level
function startLevel(levelIndex) {
    currentLevelIndex = levelIndex;
    currentLevel = levels[levelIndex];

    // Set level-specific settings
    pipeSpeed = currentLevel.pipeSpeed;
    pipeGap = currentLevel.pipeGap;

```

```

    pipeSpawnInterval = currentLevel.pipeSpawnInterval;
    totalPipes = currentLevel.totalPipes;

    // Reset game state
    gameState = 'playing';
    score = 0;
    pipesSpawned = 0;
    finishLineX = null;
    scoreElement.textContent = `Score: ${score}`;
    levelInfoElement.textContent = `Level: ${levelIndex + 1}`;
    progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
    gameOverPopup.classList.add('hidden');
    levelCompleteElement.classList.add('hidden');

    // Reset bird
    bird.y = canvas.height / 2;
    bird.velocity = 0;

    // Reset pipes
    pipes.length = 0;
    pipeSpawnTimer = 0;
    createPipe();
    pipesSpawned = 1;

    // Reset coins
    coins.length = 0;
    coinSpawnTimer = 0;

    // Update coin display
    updateCoinDisplay();

    // Show game screen, hide level selection
    levelSelectionElement.classList.add('hidden');
    gameScreenElement.classList.remove('hidden');
}

// Initialize first pipe
function createPipe() {
    const minHeight = 50;
    const maxHeight = canvas.height - pipeGap - minHeight;
    const topHeight = Math.random() * (maxHeight - minHeight) +
minHeight;

```

```

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

  // Create coin
  function createCoin() {
    // Find a safe Y position that avoids pipes
    const coinSize = 20;
    let coinY;
    let attempts = 0;
    let safePosition = false;

    // Try to find a safe position
    while (!safePosition && attempts < 10) {
      coinY = 50 + Math.random() * (canvas.height - 100); //
      Random Y, avoiding top/bottom edges

      // Check if this position would overlap with any nearby
      pipes
      let tooCloseToPipe = false;
      pipes.forEach(pipe => {
        const distance = Math.abs(pipe.x - canvas.width);
        // Only check pipes that are close to spawn point
        if (distance < 100) {
          // Check if coin would be in pipe area
          if (coinY < pipe.topHeight || coinY + coinSize
          > pipe.bottomY) {
            tooCloseToPipe = true;
          }
        }
      });

      if (!tooCloseToPipe) {
        safePosition = true;
      }
      attempts;
    }

    // If we couldn't find a safe position, use center
    if (!safePosition) {

```

```

        coinY = canvas.height / 2;
    }

    coins.push({
        x: canvas.width,
        y: coinY,
        size: coinSize,
        collected: false,
        rotation: 0
    });
}

// Draw finish line
function drawFinishLine() {
    if (finishLineX = null) return;

    ctx.strokeStyle = '#FFD700';
    ctx.lineWidth = 5;
    ctx.setLineDash([10, 10]);
    ctx.beginPath();
    ctx.moveTo(finishLineX, 0);
    ctx.lineTo(finishLineX, canvas.height);
    ctx.stroke();
    ctx.setLineDash([]);

    // Draw "FINISH" text
    ctx.fillStyle = '#FFD700';
    ctx.font = 'bold 24px Arial';
    ctx.textAlign = 'center';
    ctx.fillText('FINISH', finishLineX, canvas.height / 2);
    ctx.textAlign = 'left';
}

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

    // Draw eye
    ctx.fillStyle = 'white';
    ctx.beginPath();

```

```

    ctx.arc(bird.x + bird.width/2 + 5, bird.y + bird.height/2 -
5, 5, 0, Math.PI * 2);
    ctx.fill();

    ctx.fillStyle = 'black';
    ctx.beginPath();
    ctx.arc(bird.x + bird.width/2 + 6, bird.y + bird.height/2 -
5, 3, 0, Math.PI * 2);
    ctx.fill();

    // Draw beak
    ctx.fillStyle = '#FF8C00';
    ctx.beginPath();
    ctx.moveTo(bird.x + bird.width, bird.y + bird.height/2);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
- 3);
    ctx.lineTo(bird.x + bird.width + 8, bird.y + bird.height/2
+ 3);
    ctx.closePath();
    ctx.fill();
}

// Draw pipes
function drawPipes() {
    ctx.fillStyle = '#228B22';

    pipes.forEach(pipe => {
        // Top pipe
        ctx.fillRect(pipe.x, 0, pipeWidth, pipe.topHeight);
        ctx.fillRect(pipe.x - 5, pipe.topHeight - 20, pipeWidth
+ 10, 20);

        // Bottom pipe
        ctx.fillRect(pipe.x, pipe.bottomY, pipeWidth,
canvas.height - pipe.bottomY);
        ctx.fillRect(pipe.x - 5, pipe.bottomY, pipeWidth + 10,
20);
    });
}

// Draw coins
function drawCoins() {
    coins.forEach(coin => {
        if (!coin.collected) {

```

```

        ctx.save();
        ctx.translate(coin.x + coin.size / 2, coin.y +
coin.size / 2);
        coin.rotation += 0.1;
        ctx.rotate(coin.rotation);

        // Draw coin (golden circle)
        ctx.fillStyle = '#FFD700';
        ctx.beginPath();
        ctx.arc(0, 0, coin.size / 2, 0, Math.PI * 2);
        ctx.fill();

        // Draw coin border
        ctx.strokeStyle = '#FFA500';
        ctx.lineWidth = 2;
        ctx.stroke();

        // Draw $ symbol
        ctx.fillStyle = '#FF8C00';
        ctx.font = `${coin.size * 0.6}px Arial`;
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillText('$', 0, 0);

        ctx.restore();
    }
});
}

// Update bird
function updateBird() {
    if (gameState = 'playing') {
        bird.velocity += bird.gravity;
        bird.y += bird.velocity;

        // Check boundaries
        if (bird.y + bird.height > canvas.height) {
            bird.y = canvas.height - bird.height;
            gameOver();
        }
        if (bird.y < 0) {
            bird.y = 0;
            bird.velocity = 0;
        }
    }
}

```



```

    }
}

// Update pipes
function updatePipes() {
    if (gameState = 'playing') {
        // Spawn new pipes until we reach the total
        if (pipesSpawned < totalPipes) {
            pipeSpawnTimer;
            if (pipeSpawnTimer >= pipeSpawnInterval) {
                createPipe();
                pipesSpawned;
                pipeSpawnTimer = 0;
            }
        }

        // Show finish line when all pipes are passed
        if (score >= totalPipes && finishLineX = null) {
            // Find the rightmost pipe position and place
            finish line after it
            let rightmostX = bird.x + 200; // Default position
            ahead of bird
            if (pipes.length > 0) {
                pipes.forEach(pipe => {
                    if (pipe.x + pipeWidth > rightmostX) {
                        rightmostX = pipe.x + pipeWidth;
                    }
                });
                finishLineX = rightmostX + 150;
            } else {
                // If no pipes left, place finish line ahead of
                bird
                finishLineX = bird.x + 200;
            }
        }

        // Move pipes
        pipes.forEach(pipe => {
            pipe.x -= pipeSpeed;

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

```

```

        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: $
{score} / ${totalPipes}`;

        // Check if all pipes are passed - complete
level immediately
        if (score >= totalPipes) {
            completeLevel();
        }
    });

    // Move finish line (visual indicator only)
    if (finishLineX ! null) {
        finishLineX -= pipeSpeed;
    }

    // Remove off-screen pipes
    while (pipes.length > 0 && pipes[0].x + pipeWidth < 0)
{
        pipes.shift();
    }
}

// Update coins
function updateCoins() {
    if (gameState = 'playing') {
        // Spawn coins periodically
        coinSpawnTimer;
        if (coinSpawnTimer >= coinSpawnInterval) {
            // Only spawn coins if we haven't reached the pipe
limit
            if (pipesSpawned < totalPipes || pipes.length > 0)
{
                createCoin();
            }
            coinSpawnTimer = 0;
        }

        // Move coins
        coins.forEach(coin => {
            if (!coin.collected) {
                coin.x -= pipeSpeed;
            }
        });
    }
}

```

```

        }
    });

    // Remove off-screen or collected coins
    for (let i = coins.length - 1; i >= 0; i--) {
        if (coins[i].collected || coins[i].x +
coins[i].size < 0) {
            coins.splice(i, 1);
        }
    }
}

// Collision detection
function checkCollisions() {
    if (gameState !== 'playing') return;

    // Check pipe collisions
    pipes.forEach(pipe => {
        // Check collision with top pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y < pipe.topHeight) {
            gameOver();
        }

        // Check collision with bottom pipe
        if (bird.x < pipe.x + pipeWidth &&
            bird.x + bird.width > pipe.x &&
            bird.y + bird.height > pipe.bottomY) {
            gameOver();
        }
    });

    // Check coin collisions
    coins.forEach(coin => {
        if (!coin.collected) {
            const coinCenterX = coin.x + coin.size / 2;
            const coinCenterY = coin.y + coin.size / 2;
            const birdCenterX = bird.x + bird.width / 2;
            const birdCenterY = bird.y + bird.height / 2;

            const distance = Math.sqrt(
                Math.pow(coinCenterX - birdCenterX, 2) +

```

```

        Math.pow(coinCenterY - birdCenterY, 2)
    );

    // Check if bird is close enough to collect coin
    if (distance < (coin.size / 2 + bird.width / 2)) {
        coin.collected = true;
        addCoin();
    }
}

});
}

// Game over
function gameOver() {
    if (gameState = 'playing') {
        gameState = 'gameOver';
        gameOverPopup.classList.remove('hidden');
        gameOverPopup.style.display = 'flex'; // Ensure popup
is visible
        notEnoughCoinsElement.classList.add('hidden');

        // Update continue button based on available coins
        const hasEnoughCoins = getCoins() >= 10;
        continueBtn.disabled = !hasEnoughCoins;
        if (!hasEnoughCoins) {
            continueBtn.style.opacity = '0.5';
            continueBtn.style.cursor = 'not-allowed';
        } else {
            continueBtn.style.opacity = '1';
            continueBtn.style.cursor = 'pointer';
        }
    }
}

// Complete level
function completeLevel() {
    if (gameState = 'playing') {
        gameState = 'levelComplete';
        levelCompleteElement.classList.remove('hidden');
        gameOverPopup.classList.add('hidden');

        // Mark level as completed
        markLevelCompleted(currentLevelIndex);
    }
}

```

```

        // Unlock next level
        if (currentLevelIndex + 1 < levels.length) {
            unlockLevel(currentLevelIndex + 1);
        }
    }
}

// Continue game (revive)
function continueGame() {
    if (gameState = 'gameOver') {
        // Check if user has enough coins
        if (!spendCoins(10)) {
            notEnoughCoinsElement.classList.remove('hidden');
            return;
        }

        // Remove next 2-3 pipes and move them to the back
        const pipesToRemove = Math.min(3, pipes.length);
        const removedPipes = [];

        // Remove pipes that are ahead of the bird
        for (let i = pipes.length - 1; i >= 0; i--) {
            if (pipes[i].x > bird.x && removedPipes.length <
pipesToRemove) {
                removedPipes.push(pipes.splice(i, 1)[0]);
            }
        }

        // Move removed pipes to the back (far right)
        removedPipes.forEach(pipe => {
            pipe.x = canvas.width + 200 + Math.random() *
200; // Place them far ahead
            pipe.passed = false; // Reset passed status
            pipes.push(pipe);
        });

        // Sort pipes by x position to maintain order
        pipes.sort((a, b) => a.x - b.x);

        // Reset bird to middle of screen
        bird.y = canvas.height / 2;
        bird.velocity = 0;

        // Clear collected coins that are off-screen

```

```

    coins.forEach(coin => {
        if (coin.x < bird.x - 100) {
            coin.collected = true;
        }
    });

    // Resume game
    gameState = 'playing';
    gameOverPopup.classList.add('hidden');
    notEnoughCoinsElement.classList.add('hidden');

    // Ensure game screen is visible
    if (gameScreenElement) {
        gameScreenElement.classList.remove('hidden');
    }

    // Force hide popup with inline style as backup
    if (gameOverPopup) {
        gameOverPopup.style.display = 'none';
    }
}

// Reset game (play again)
function resetGame() {
    if (gameState = 'gameOver' || gameState = 'levelComplete')
    {
        gameState = 'playing';
        score = 0;
        pipesSpawned = 0;
        finishLineX = null;
        scoreElement.textContent = `Score: ${score}`;
        progressElement.textContent = `Progress: ${score} / $
{totalPipes}`;
        gameOverPopup.classList.add('hidden');
        levelCompleteElement.classList.add('hidden');

        bird.y = canvas.height / 2;
        bird.velocity = 0;

        pipes.length = 0;
        pipeSpawnTimer = 0;
        createPipe();
        pipesSpawned = 1;
    }
}

```

```

        // Reset coins
        coins.length = 0;
        coinSpawnTimer = 0;
    }
}

// Return to menu
function returnToMenu() {
    gameState = 'menu';
    levelSelectionElement.classList.remove('hidden');
    gameScreenElement.classList.add('hidden');
    gameOverPopup.classList.add('hidden'); // Hide popup when
returning to menu
    notEnoughCoinsElement.classList.add('hidden');
    initLevelSelection(); // Refresh to show newly unlocked
levels
}

// Jump function
function jump() {
    if (gameState = 'playing') {
        bird.velocity = bird.jumpStrength;
    } else if (gameState = 'levelComplete') {
        returnToMenu();
    }
    // Game over now uses popup buttons, not space key
}

// Keyboard input
document.addEventListener('keydown', (e) => {
    if (e.code = 'Space') {
        e.preventDefault();
        jump();
    }
});

// Mouse/touch input
canvas.addEventListener('click', jump);
canvas.addEventListener('touchstart', (e) => {
    e.preventDefault();
    jump();
});

```

```

// Back to menu button
backToMenuButton.addEventListener('click', returnToMenu);

// Game over popup buttons
playAgainBtn.addEventListener('click', resetGame);
continueBtn.addEventListener('click', continueGame);

// Game loop
function gameLoop() {
    if (gameState !== 'menu') {
        // Clear canvas
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        // Draw background
        const gradient = ctx.createLinearGradient(0, 0, 0,
canvas.height);
        gradient.addColorStop(0, '#87CEEB');
        gradient.addColorStop(0.7, '#87CEEB');
        gradient.addColorStop(0.7, '#90EE90');
        gradient.addColorStop(1, '#90EE90');
        ctx.fillStyle = gradient;
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        // Update and draw
        updateBird();
        updatePipes();
        updateCoins();
        checkCollisions();

        drawPipes();
        drawCoins();
        drawFinishLine();
        drawBird();
    }

    requestAnimationFrame(gameLoop);
}

// Initialize game
updateCoinDisplay(); // Initialize coin display
gameOverPopup.classList.add('hidden'); // Ensure popup is
hidden on startup
notEnoughCoinsElement.classList.add('hidden');

```



```
initLevelSelection();  
gameLoop();
```

Fixed the continue functionality. Changes:

1. **Explicit display style:** When showing the popup (**gameOver**), it sets **display: flex** inline.
2. **Force hide on continue:** When continuing, it sets **display: none** inline and adds the **hidden** class.
3. **Ensure game screen is visible:** Verifies the game screen is visible when continuing.

When you click "Continue (10 coins)":

- The popup hides (both via class and inline style)
- The game state returns to 'playing'
- The game screen remains visible
- You resume gameplay

The popup should now disappear and the game should resume when you continue.

User

it still doesnt put me back into the game. the button looks like it was pressed but doesnt do anything. also remove the pipe that killed me