***מסמך דרישות תוכנה (SRS)***

1. **שם הפרויקט : MazeRun**  
   קורס: אלגוריתמים מתקדמים  
   תאריך: יולי 2025  
   שמות השותפים: אלעד אהרון 206866220, רועי טרבינוביץ 316225382,  
    רותם רוזנברג 319044160, מיכאל גרינברג 208570572.
2. **תקציר:**  
   MazeRun הוא משחק מבוך דו־ממדי רספונסיבי לדפדפן, בו שחקן אנושי מנסה להגיע ליציאה לפני שאויב אחד או שניים תופסים אותו. המשחק מבוסס על לוגיקה בזמן אמת, הכוללת בינה מלאכותית שמבצעת חישובי מסלול בזמן אמת בעזרת אלגוריתם A\*. המבוך נבנה אקראית אך נבדק לפתרונות, וישנם פריטים כמו מטבעות, רנדומיזר ותור נוסף שנכנסים כחלק מהאתגר. המשחק כתוב כולו ב־HTML, CSS ו־JavaScript ללא תלות בשרת.
3. **קישורים:**

* הדגמת משחק: [לחץ כאן](https://rotemrosenberg.github.io/MazeRun/html/index.html)
* GitHub: [לחץ כאן](https://github.com/RotemRosenberg/MazeRun)
* מצגת הפרויקט בדרייב: [לחץ כאן](https://docs.google.com/presentation/d/1nBWyMiAreDt2IcaTmkbH-jCorvXP-vho-sqRngnGzPY/edit?usp=sharing)
* סרטון המצגת בדרייב: [לחץ כאן](https://drive.google.com/file/d/1J5giZRN9J9RB76ndvoXvL46rgzDVO1Re/view?usp=sharing)
* מצגת המשחק בדרייב: [לחץ כאן](https://drive.google.com/file/d/1ZuXUOLLaGB8aA5J1y0HBDW2AxjlF9sC4/view?usp=sharing)
* סרטון המצגת ביוטיוב: [לחץ כאן](https://youtu.be/ePvC0VaVkf4-sqRngnGzPY/edit?usp=sharing)
* סרטון המשחק ביוטיוב: [לחץ כאן](https://youtu.be/jEzuZvmKndw)
* הפנייה לreadme : [לחץ כאן](https://github.com/RotemRosenberg/MazeRun/blob/main/README.md)

1. **תיאור הבעיה:**  
   המשחק נועד לתת מענה לחוויה אינטראקטיבית שבה השחקן צריך להתמודד מול אויבים שמגיבים באופן חכם למיקומו בזמן אמת. האתגר היה לבנות AI שאינו צפוי אך הוגן, מבוך שתמיד פתיר, ולשלב אלמנטים אקראיים תוך שמירה על ביצועים בזמן אמת.
2. **מאפיינים פונקציונליים**

* משחק רספונסיבי הפועל ב־Canvas
* תנועה של השחקן באמצעות חיצים (keydown events)
* יצירת מבוך אקראי בגודל 30x30 עם 25% או 35% קירות
* אויב אחד (מצב רגיל) או שניים (מצב Hard)
* Power-ups מיוחדים:  
  - Extra Turn מאפשר תור נוסף לשחקן   
   Randomizer - מגריל מבוך חדש מיידית
* AI מבוסס A\* עם ארבעה מצבים:  
  • תקיפה (אם קרוב מאוד)  
   • חסימה (אם במרחק בינוני)  
  • רדיפה (במרחק רגיל)  
   • חיזוי (אם רחוק מאוד)
* מנגנון ניקוד לפי מטבעות שנאספו
* מסך תפריט ראשי (index.html), הדרכה (tutorial.html), מסך משחק (game.html)
* הודעות סוף משחק עם SweetAlert2

1. **אלגוריתמים ולוגיקה:**

* A\* Pathfinding:  
   •חישוב f(n) = g(n) + h(n)  
   h(n): מרחק מנהטן מותאם עם בונוס קרבה לשחקן ועונש קרבה ליציאה  
  • עלות נוספת אם יש Power-up במשבצת או סמיכות לקירות
* BFS:  
  • משמש לבדיקת תקפות המבוך בכל יצירה  
  • מוודא שיש מסלול גם לשחקן וגם לאויב
* לוגיקת AI :  
  • ניתוח מיקום שחקן ואויב בכל תור  
  • מעבר בין אסטרטגיות לפי מרחק ומצב

1. **ארכיטקטורת מערכת:**

* HTML + CSS + JS
* Canvas לייצוג ויזואלי של לוח המשחק (800x800)
* הפרדה לקבצים:  
   index.html תפריט ובחירת רמת קושי  
   game.html לוח המשחק  
   tutorial.html עמוד הסבר  
   game.js מנוע המשחק הראשי  
   index.js, tutorial.js התנהגות העמודים האחרים  
   style.css עיצוב בסיסי לכל הרכיבים
* ללא שימוש בספריות צד שלישי, מלבד SweetAlert2

1. **מבנה תיקיות וקבצים:**

* /html:  
  index.html, game.html, tutorial.html
* /js:  
  game.js, index.js, tutorial.js
* /css:  
  style.css

1. **אתגרים מרכזיים ופתרונות:**

* מבוכים לא פתירים: פתרנו עם BFS בעת כל יצירת לוח
* התנגשות "חילוף מקום": זיהוי מצב בו שחקן ואויב מחליפים תאים באותו תור
* איזון התנהגות האויב: פונקציית heuristic מותאמת שגרמה לו להתקרב אך לא לשהות על היציאה
* שמירה על ביצועים: חישובי A\* נעשים רק בתורו של האויב

1. **חוזקות ומרחב לשיפור  
   חוזקות:**  
    ✔ חוויית משחק מאתגרת ודינמית  
   ✔ בינה מלאכותית אדפטיבית עם 4 מצבים  
   ✔ ביצועים טובים בזמן אמת גם בדפדפן  
   ✔ קוד מודולרי ונוח להרחבה  
   ✔ ממשק ויזואלי נעים ואינטואיטיבי

**שיפורים עתידיים:**

* הוספת צלילים ואנימציות
* מעבר בין שלבים
* שמירת שיאים או תוצאות
* הוספת מצב מרובה שחקנים

11. **קוד הפרוייקט:**  
/\* Base styles for all pages \*/

body {

    font-family: Arial, sans-serif;

    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

    min-height: 100vh;

    margin: 0;

    padding: 20px;

    color: #333;

}

/\* For index page centered layout \*/

body.index-page {

    display: flex;

    align-items: center;

    justify-content: center;

}

/\* Common container styles \*/

.container {

    background: white;

    border-radius: 15px;

    padding: 30px;

    box-shadow: 0 10px 30px rgba(0,0,0,0.2);

    max-width: 1200px;

    margin: 0 auto;

}

/\* Typography \*/

h1 {

    text-align: center;

    color: #333;

    margin-bottom: 30px;

    font-size: 2.5em;

    text-shadow: 2px 2px 4px rgba(0,0,0,0.1);

    background: linear-gradient(45deg, #667eea, #764ba2);

    -webkit-background-clip: text;

    -webkit-text-fill-color: transparent;

    background-clip: text;

}

h2 {

    color: #333;

    margin-bottom: 20px;

    font-size: 1.8em;

}

h3 {

    color: #333;

    margin-bottom: 15px;

    font-size: 1.3em;

}

/\* Button styles \*/

.btn {

    padding: 12px 25px;

    border: none;

    border-radius: 8px;

    cursor: pointer;

    font-weight: bold;

    transition: all 0.3s ease;

    text-decoration: none;

    display: inline-block;

    margin: 5px;

    font-size: 1em;

}

.btn:hover {

    transform: translateY(-2px);

    box-shadow: 0 4px 12px rgba(0,0,0,0.2);

}

.btn-primary {

    background: linear-gradient(45deg, #667eea, #764ba2);

    color: white;

}

.btn-secondary {

    background: linear-gradient(45deg, #56ccf2, #2f80ed);

    color: white;

}

.btn-success {

    background: linear-gradient(45deg, #4CAF50, #45a049);

    color: white;

}

.btn-danger {

    background: linear-gradient(45deg, #ff4444, #cc0000);

    color: white;

}

.btn-warning {

    background: linear-gradient(45deg, #ff9800, #f57c00);

    color: white;

}

/\* Game specific styles \*/

.game-container {

    background: white;

    border-radius: 15px;

    padding: 30px;

    box-shadow: 0 10px 30px rgba(0,0,0,0.2);

    max-width: 1200px;

    margin: 0 auto;

}

.game-controls {

    display: flex;

    gap: 10px;

    flex-wrap: wrap;

    justify-content: center;

}

.game-info-panel-horizontal {

    background: linear-gradient(135deg, #f093fb 0%, #f5576c 100%);

    color: white;

    padding: 20px;

    border-radius: 15px;

    box-shadow: 0 4px 15px rgba(0,0,0,0.1);

    margin: 20px 0;

    display: flex;

    gap: 40px;

    justify-content: center;

    align-items: center;

    flex-wrap: wrap;

}

.info-item-horizontal {

    padding: 15px 25px;

    background: rgba(255,255,255,0.15);

    border-radius: 10px;

    backdrop-filter: blur(10px);

    text-align: center;

    min-width: 200px;

    box-shadow: 0 2px 8px rgba(0,0,0,0.1);

    flex: 1;

    max-width: 350px;

}

.info-item-line {

    margin-bottom: 8px;

}

.info-group {

    background: #f8f9fa;

    padding: 15px 20px;

    border-radius: 10px;

    box-shadow: 0 2px 8px rgba(0,0,0,0.1);

    flex: 1;

    text-align: center;

}

.info-group h3 {

    margin: 0 0 10px 0;

    color: #333;

    font-size: 1.1em;

}

.info-group p {

    margin: 5px 0;

    color: #666;

    font-size: 0.9em;

}

.game-canvas-container {

    display: flex;

    justify-content: center;

    margin: 20px 0;

}

#gameCanvas {

    border: 3px solid #333;

    border-radius: 15px;

    box-shadow: 0 8px 25px rgba(0,0,0,0.2);

    background: white;

}

.game-legend {

    margin: 30px auto;

    text-align: center;

    background: #f9f9f9;

    padding: 25px;

    border-radius: 15px;

}

.game-legend h3 {

    color: #333;

    margin-bottom: 20px;

    font-size: 1.5em;

}

.game-legend-items {

    display: flex;

    gap: 15px;

    flex-wrap: wrap;

    justify-content: center;

}

.game-legend p {

    margin-top: 20px;

    color: #666;

    font-size: 1.1em;

}

.btn-home {

    background: linear-gradient(45deg, #56ccf2, #2f80ed);

    color: white;

}

/\* Legend styles \*/

.legend {

    background: #f8f9fa;

    padding: 20px;

    border-radius: 10px;

    box-shadow: 0 2px 8px rgba(0,0,0,0.1);

    margin-top: 20px;

    max-width: 800px;

    margin-left: auto;

    margin-right: auto;

}

.legend h3 {

    margin-top: 0;

    color: #333;

    text-align: center;

}

.legend h4 {

    margin-top: 0;

    color: #333;

}

.legend-grid {

    display: grid;

    grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));

    gap: 15px;

    margin-top: 15px;

}

.legend-item {

    display: flex;

    align-items: center;

    gap: 8px;

    padding: 8px 12px;

    background: white;

    border-radius: 6px;

    box-shadow: 0 1px 3px rgba(0,0,0,0.1);

    font-size: 0.9em;

    color: black;

}

.legend-item:hover {

    transform: translateY(-1px);

    box-shadow: 0 2px 6px rgba(0,0,0,0.15);

}

.game-legend-items .legend-item {

    padding: 10px 15px;

    border-radius: 8px;

    box-shadow: 0 2px 8px rgba(0,0,0,0.1);

    transition: transform 0.2s ease;

}

.game-legend-items .legend-item:hover {

    transform: translateY(-2px);

    box-shadow: 0 4px 12px rgba(0,0,0,0.15);

}

/\* Index page specific styles \*/

.main-container {

    background: white;

    border-radius: 20px;

    padding: 50px;

    box-shadow: 0 15px 35px rgba(0,0,0,0.2);

    max-width: 600px;

    text-align: center;

    margin: 0 auto;

}

.main-container h1 {

    font-size: 3.5em;

    margin-bottom: 20px;

}

.subtitle {

    color: #666;

    font-size: 1.2em;

    margin-bottom: 40px;

}

.menu-buttons {

    display: flex;

    flex-direction: column;

    gap: 20px;

    max-width: 300px;

    margin: 0 auto;

}

.menu-buttons .btn {

    padding: 15px 30px;

    font-size: 1.1em;

    border-radius: 12px;

}

.menu-buttons .btn:hover {

    transform: translateY(-3px);

    box-shadow: 0 6px 20px rgba(0,0,0,0.2);

}

.features {

    margin-top: 40px;

    padding-top: 30px;

    border-top: 2px solid #eee;

}

.features h3 {

    color: #333;

    margin-bottom: 20px;

}

.feature-list {

    display: flex;

    gap: 30px;

    justify-content: center;

    flex-wrap: wrap;

}

.feature-item {

    background: #f8f9fa;

    padding: 15px;

    border-radius: 10px;

    min-width: 120px;

    box-shadow: 0 2px 8px rgba(0,0,0,0.1);

    text-align: center;

}

.feature-item:hover {

    transform: translateY(-2px);

    box-shadow: 0 4px 12px rgba(0,0,0,0.15);

}

/\* Tutorial specific styles \*/

.tutorial-container {

    background: white;

    border-radius: 15px;

    padding: 30px;

    box-shadow: 0 10px 30px rgba(0,0,0,0.2);

    max-width: 800px;

    margin: 0 auto;

}

.tutorial-content {

    display: flex;

    gap: 30px;

    align-items: flex-start;

    margin-bottom: 30px;

}

.canvas-container {

    flex: 1;

    text-align: center;

}

.instructions-panel {

    flex: 1;

    background: #f8f9fa;

    padding: 20px;

    border-radius: 10px;

    box-shadow: 0 2px 8px rgba(0,0,0,0.1);

}

#tutorialCanvas {

    border: 3px solid #333;

    border-radius: 10px;

    box-shadow: 0 4px 15px rgba(0,0,0,0.2);

}

#instruction {

    font-size: 1.2em;

    color: #333;

    margin-bottom: 20px;

    line-height: 1.5;

}

.legend .legend-items {

    display: grid;

    grid-template-columns: 1fr 1fr;

    gap: 10px;

    font-size: 0.9em;

}

.legend .legend-item {

    display: flex;

    align-items: center;

    gap: 5px;

    padding: 5px;

    background: transparent;

    box-shadow: none;

}

.legend .legend-item:hover {

    transform: none;

    box-shadow: none;

}

.legend .legend-items .legend-item {

    background: #e9ecef;

    padding: 8px;

    border-radius: 6px;

}

.step-indicator {

    text-align: center;

    margin-bottom: 20px;

}

.step-dots {

    display: flex;

    justify-content: center;

    gap: 10px;

    margin-top: 10px;

}

.step-dot {

    width: 12px;

    height: 12px;

    border-radius: 50%;

    background: #ddd;

    transition: all 0.3s ease;

}

.step-dot.active {

    background: linear-gradient(45deg, #667eea, #764ba2);

    transform: scale(1.2);

}

.step-dot.completed {

    background: #4CAF50;

}

.button-group {

    display: flex;

    gap: 15px;

    justify-content: center;

    margin-top: 20px;

}

.game-container .button-group {

    margin-bottom: 20px;

    gap: 10px;

}

.game-container .btn {

    padding: 10px 20px;

}

/\* Responsive design \*/

@media (max-width: 768px) {

    body {

        padding: 10px;

    }

    .tutorial-content {

        flex-direction: column;

        gap: 20px;

    }

    .canvas-container {

        order: 2;

    }

    .instructions-panel {

        order: 1;

    }

    .game-info-panel-horizontal {

        flex-direction: column;

        gap: 15px;

    }

    .legend-grid {

        grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));

        gap: 10px;

    }

    .main-container {

        padding: 30px 20px;

    }

    .main-container h1 {

        font-size: 2.5em;

    }

    .legend .legend-items {

        grid-template-columns: 1fr;

    }

}

<!DOCTYPE html>

<html lang="he">

<head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Maze Run – Game</title>

    <link rel="stylesheet" href="../css/style.css" />

    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

</head>

<body>

    <div class="game-container">

        <h1>🎮 Maze Run – Game</h1>

        <div class="button-group">

            <button class="btn btn-home" onclick="window.location.href='index.html'">🏠 Back to Home</button>

            <button class="btn btn-primary" onclick="initGrid('normal')">🔄 New Maze (Normal)</button>

            <button class="btn btn-danger" onclick="initGrid('hard')">🔥 Hard Mode (2 Enemies)</button>

        </div>

        <!-- Game Info Panel - Now horizontal above the game -->

        <div class="game-info-panel-horizontal">

            <div class="info-item-horizontal">

                <div class="info-item-line"><strong>Score:</strong> <span id="scoreDisplay">0</span></div>

                <div><strong>Mode:</strong> <span id="modeDisplay">NORMAL</span></div>

            </div>

            <div class="info-item-horizontal">

                <div class="info-item-line"><strong>Goal:</strong> Collect coins 🪙 and reach the exit 🚪</div>

                <div><strong>Controls:</strong> ⬆️ ⬇️ ⬅️ ➡️ Arrow keys to move</div>

            </div>

        </div>

        <!-- Game Canvas - Now centered -->

        <div class="game-canvas-container">

            <canvas id="gameCanvas" width="800" height="800"></canvas>

        </div>

        <!-- Game Legend at bottom -->

        <div class="game-legend">

            <h3>🎯 Game Legend</h3>

            <div class="game-legend-items">

                <div class="legend-item">👤 Player</div>

                <div class="legend-item">👹 Enemy 1</div>

                <div class="legend-item">😈 Enemy 2</div>

                <div class="legend-item">🚪 Exit</div>

                <div class="legend-item">🪙 Coins</div>

                <div class="legend-item">⚡ Extra Turn</div>

                <div class="legend-item">🔄 Randomizer</div>

                <div class="legend-item">🧱 Wall</div>

            </div>

            <p>

                <strong>Normal Mode:</strong> Face 1 enemy with 25% walls |

                <strong>Hard Mode:</strong> Face 2 enemies with 35% walls - Ultimate challenge!

            </p>

        </div>

    </div>

    <script src="../js/game.js"></script>

    <script>

        // Auto-start the game when page loads

        window.onload = function() {

            initGrid('normal');

        };

    </script>

</body>

</html>

<!DOCTYPE html>

<html lang="he">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Maze Run - Start</title>

    <link rel="stylesheet" href="../css/style.css">

    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

</head>

<body class="index-page">

    <div class="main-container">

        <h1>🎮 MAZE RUN</h1>

        <p class="subtitle">Navigate through mazes, collect coins, and escape from intelligent enemies!</p>

        <div class="menu-buttons">

            <button class="btn btn-primary" onclick="startGame()">🎮 Start Game</button>

            <button class="btn btn-secondary" onclick="openTutorial()">📘 Tutorial</button>

        </div>

        <div class="features">

            <h3>🎯 Game Features</h3>

            <div class="feature-list">

                <div class="feature-item">

                    <div>🪙</div>

                    <div><strong>Collect Coins</strong></div>

                </div>

                <div class="feature-item">

                    <div>👹</div>

                    <div><strong>Smart AI</strong></div>

                </div>

                <div class="feature-item">

                    <div>⚡</div>

                    <div><strong>Power-ups</strong></div>

                </div>

                <div class="feature-item">

                    <div>🔥</div>

                    <div><strong>Hard Mode</strong></div>

                </div>

                <div class="feature-item">

                    <div>🧱</div>

                    <div><strong>Random Mazes</strong></div>

                </div>

                <div class="feature-item">

                    <div>🎯</div>

                    <div><strong>Score System</strong></div>

                </div>

            </div>

        </div>

        <p style="text-align: center; margin-top: 20px; font-size: 14px; color: #666;">Created by Rotem, Roee, Elad & Michael</p>

    </div>

    <script src="../js/index.js"></script>

</body>

</html>

<!DOCTYPE html>

<html lang="he">

<head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Maze Run Tutorial</title>

    <link rel="stylesheet" href="../css/style.css">

    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

</head>

<body>

    <div class="tutorial-container">

        <h1>📘 Maze Run – Tutorial</h1>

        <div class="step-indicator">

            <p><strong>Step <span id="stepNumber">1</span> of 6</strong></p>

            <div class="step-dots">

                <div class="step-dot active" id="dot1"></div>

                <div class="step-dot" id="dot2"></div>

                <div class="step-dot" id="dot3"></div>

                <div class="step-dot" id="dot4"></div>

                <div class="step-dot" id="dot5"></div>

                <div class="step-dot" id="dot6"></div>

            </div>

        </div>

        <div class="tutorial-content">

            <div class="canvas-container">

                <canvas id="tutorialCanvas" width="400" height="400"></canvas>

            </div>

            <div class="instructions-panel">

                <p id="instruction">Use arrow keys ⬆️ ⬇️ ⬅️ ➡️ to move the player 👤<br><em>Note: In tutorial mode, enemies don't move</em></p>

                <div class="legend">

                    <h4>🎯 Game Elements:</h4>

                    <div class="legend-items">

                        <div class="legend-item">👤 Player (you)</div>

                        <div class="legend-item">👹 Enemy (static)</div>

                        <div class="legend-item">🚪 Exit</div>

                        <div class="legend-item">🪙 Coins</div>

                        <div class="legend-item">⚡ Extra Turn</div>

                        <div class="legend-item">🔄 Randomizer</div>

                        <div class="legend-item">🧱 Wall</div>

                        <div class="legend-item">😈 Enemy 2 (static)</div>

                    </div>

                </div>

            </div>

        </div>

        <div class="button-group">

            <button class="btn btn-secondary" onclick="window.location.href='index.html'">🏠 Back to Home</button>

            <button class="btn btn-primary" id="nextBtn" onclick="nextStep()">Next Step ➡️</button>

        </div>

    </div>

    <script src="../js/tutorial.js"></script>

</body>

</html>

const canvas = document.getElementById("gameCanvas");

const ctx = canvas.getContext("2d");

const SIZE = 30; // Fixed 30x30 grid

// Update canvas size to 800x800 to accommodate 30x30 grid

canvas.width = 800;

canvas.height = 800;

const CELL\_SIZE = canvas.width / SIZE;

let grid = [];

let player = { x: 0, y: 0 };

let enemy = { x: SIZE - 1, y: SIZE - 1 };

let enemy2 = { x: 0, y: SIZE - 1 }; // Second enemy starts at bottom-left

let exit = { x: Math.floor(SIZE / 2), y: SIZE - 1 };

const WALLS\_COUNT\_NORMAL = Math.floor(SIZE \* SIZE \* 0.25); // Normal mode: 25%

const WALLS\_COUNT\_HARD = Math.floor(SIZE \* SIZE \* 0.35); // Hard mode: 35%

const RANDOMIZER\_COUNT = 3; // Increased from 2 to 3

const EXTRA\_TURN\_COUNT = 5; // Increased from 2 to 5 (2 + 3 more)

const COIN\_COUNT = 5; // Always 5 coins on the map

let gameMode = 'normal'; // 'normal' or 'hard'

let extraTurn = false;

let score = 0; // Player score from collecting coins

let playerHistory = []; // היסטוריית תנועות השחקן

let lastPlayerPosition = { x: 0, y: 0 };

let lastEnemyPosition = { x: SIZE - 1, y: SIZE - 1 };

let lastEnemy2Position = { x: 0, y: SIZE - 1 };

function createCell(x, y) {

    return {

        x,

        y,

        isWall: false,

        isExit: false,

        isRandomizer: false,

        isExtraTurn: false,

        isCoin: false

    };

}

// === פונקציות מניעת תקיעות ===

// פונקציה לבדיקת קישוריות בין שתי נקודות

function isReachable(start, end, gridToCheck = grid) {

    if (start.x === end.x && start.y === end.y) return true;

    const visited = new Set();

    const queue = [start];

    const key = (p) => `${p.x},${p.y}`;

    visited.add(key(start));

    while (queue.length > 0) {

        const current = queue.shift();

        const neighbors = [

            { x: current.x + 1, y: current.y },

            { x: current.x - 1, y: current.y },

            { x: current.x, y: current.y + 1 },

            { x: current.x, y: current.y - 1 }

        ];

        for (const neighbor of neighbors) {

            if (neighbor.x < 0 || neighbor.x >= SIZE ||

                neighbor.y < 0 || neighbor.y >= SIZE) continue;

            if (gridToCheck[neighbor.y][neighbor.x].isWall ||

                visited.has(key(neighbor))) continue;

            if (neighbor.x === end.x && neighbor.y === end.y) {

                return true;

            }

            visited.add(key(neighbor));

            queue.push(neighbor);

        }

    }

    return false;

}

// פונקציה לבדיקת תקינות המפה

function isGridValid(gridToCheck = grid) {

    let isValid = isReachable(player, enemy, gridToCheck) &&

        isReachable(player, exit, gridToCheck) &&

        isReachable(enemy, player, gridToCheck);

    // In hard mode, also check enemy2 connectivity

    if (gameMode === 'hard') {

        isValid = isValid &&

            isReachable(player, enemy2, gridToCheck) &&

            isReachable(enemy2, player, gridToCheck);

    }

    return isValid;

}

// פונקציה לבדיקה אם מיקום תפוס

function isPositionOccupied(x, y) {

    return (x === player.x && y === player.y) ||

        (x === enemy.x && y === enemy.y) ||

        (x === enemy2.x && y === enemy2.y && gameMode === 'hard') ||

        (x === exit.x && y === exit.y);

}

// פונקציה לבדיקה אם מיקום תפוס על ידי אלמנט מיוחד

function isSpecialElementOccupied(x, y) {

    const cell = grid[y][x];

    return cell.isWall || cell.isExit || cell.isRandomizer || cell.isExtraTurn || cell.isCoin;

}

// פונקציה לערבוב מערך

function shuffleArray(array) {

    for (let i = array.length - 1; i > 0; i--) {

        const j = Math.floor(Math.random() \* (i + 1));

        [array[i], array[j]] = [array[j], array[i]];

    }

}

// פונקציה מתקדמת להנחת קירות בבטחה

function placeWallsSafely() {

    const maxAttempts = 100;

    let attempts = 0;

    let wallsPlaced = 0;

    const targetWalls = gameMode === 'hard' ? WALLS\_COUNT\_HARD : WALLS\_COUNT\_NORMAL;

    const availableCells = [];

    for (let y = 0; y < SIZE; y++) {

        for (let x = 0; x < SIZE; x++) {

            if (!isPositionOccupied(x, y)) {

                availableCells.push({ x, y });

            }

        }

    }

    shuffleArray(availableCells);

    for (const cell of availableCells) {

        if (wallsPlaced >= targetWalls || attempts >= maxAttempts) break;

        grid[cell.y][cell.x].isWall = true;

        if (isGridValid()) {

            wallsPlaced++;

        } else {

            grid[cell.y][cell.x].isWall = false;

        }

        attempts++;

    }

    console.log(`Placed ${wallsPlaced}/${targetWalls} walls safely`);

}

// פונקציה מתקנת להנחת אלמנטים מיוחדים

function placeSpecialElementsSafely() {

    const specialElements = [

        { type: 'isRandomizer', count: RANDOMIZER\_COUNT },

        { type: 'isExtraTurn', count: EXTRA\_TURN\_COUNT },

        { type: 'isCoin', count: COIN\_COUNT }

    ];

    for (const element of specialElements) {

        let placed = 0;

        let attempts = 0;

        const maxAttempts = 100;

        while (placed < element.count && attempts < maxAttempts) {

            const x = Math.floor(Math.random() \* SIZE);

            const y = Math.floor(Math.random() \* SIZE);

            if (!isPositionOccupied(x, y) &&

                !isSpecialElementOccupied(x, y)) {

                grid[y][x][element.type] = true;

                placed++;

            }

            attempts++;

        }

        console.log(`Placed ${placed}/${element.count} ${element.type} elements`);

    }

}

// פונקציה משופרת להחלפת placeRandomElements

function placeRandomElementsSafe() {

    // איפוס כל האלמנטים המיוחדים (מלבד היציאה)

    for (let y = 0; y < SIZE; y++) {

        for (let x = 0; x < SIZE; x++) {

            if (x === exit.x && y === exit.y) continue;

            grid[y][x].isWall = false;

            grid[y][x].isRandomizer = false;

            grid[y][x].isExtraTurn = false;

            grid[y][x].isCoin = false;

        }

    }

    placeWallsSafely();

    placeSpecialElementsSafely();

}

// פונקציה לווידוא תקינות בהתחלה

function validateInitialGrid() {

    let attempts = 0;

    const maxAttempts = 10;

    while (!isGridValid() && attempts < maxAttempts) {

        console.log(`Grid invalid, regenerating... (attempt ${attempts + 1})`);

        for (let y = 0; y < SIZE; y++) {

            for (let x = 0; x < SIZE; x++) {

                if (x === exit.x && y === exit.y) continue;

                grid[y][x].isWall = false;

                grid[y][x].isRandomizer = false;

                grid[y][x].isExtraTurn = false;

                grid[y][x].isCoin = false;

            }

        }

        placeRandomElementsSafe();

        attempts++;

    }

    if (!isGridValid()) {

        console.warn("Could not generate valid grid after multiple attempts!");

        // מפה ריקה במקרה חירום

        for (let y = 0; y < SIZE; y++) {

            for (let x = 0; x < SIZE; x++) {

                if (x === exit.x && y === exit.y) continue;

                grid[y][x].isWall = false;

                grid[y][x].isRandomizer = false;

                grid[y][x].isExtraTurn = false;

                grid[y][x].isCoin = false;

            }

        }

    }

}

// === שאר הפונקציות הקיימות ===

function initGrid(mode = 'normal') {

    grid = [];

    score = 0; // Reset score when starting new game

    gameMode = mode; // Set game mode

    for (let y = 0; y < SIZE; y++) {

        const row = [];

        for (let x = 0; x < SIZE; x++) {

            row.push(createCell(x, y));

        }

        grid.push(row);

    }

    player = { x: 0, y: 0 };

    enemy = { x: SIZE - 1, y: SIZE - 1 };

    enemy2 = { x: 0, y: SIZE - 1 }; // Second enemy at bottom-left

    // Place exit in the middle area

    exit = {

        x: Math.floor((player.x + enemy.x) / 2),

        y: Math.floor((player.y + enemy.y) / 2)

    };

    exit.x = Math.max(0, Math.min(SIZE - 1, exit.x));

    exit.y = Math.max(0, Math.min(SIZE - 1, exit.y));

    grid[exit.y][exit.x].isExit = true;

    // שימוש בפונקציה הבטוחה החדשה

    placeRandomElementsSafe();

    validateInitialGrid();

    drawGrid();

    // עדכון הפאנל הצדדי

    updateGameInfo();

}

function drawGrid() {

    ctx.clearRect(0, 0, canvas.width, canvas.height);

    for (let y = 0; y < SIZE; y++) {

        for (let x = 0; x < SIZE; x++) {

            const cell = grid[y][x];

            // Draw white background for all cells

            ctx.fillStyle = "#fff";

            ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

            // Draw border

            ctx.strokeStyle = "#ddd";

            ctx.lineWidth = 1;

            ctx.strokeRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

            // Draw cell contents with icons

            const centerX = x \* CELL\_SIZE + CELL\_SIZE / 2;

            const centerY = y \* CELL\_SIZE + CELL\_SIZE / 2;

            ctx.font = `${CELL\_SIZE \* 0.7}px Arial`;

            ctx.textAlign = "center";

            ctx.textBaseline = "middle";

            if (cell.isWall) {

                ctx.fillStyle = "#444";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#fff";

                ctx.fillText("🧱", centerX, centerY);

            } else if (cell.isExit) {

                ctx.fillStyle = "#e8f5e8";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("🚪", centerX, centerY);

            } else if (cell.isRandomizer) {

                ctx.fillStyle = "#e8f8ff";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("🔄", centerX, centerY);

            } else if (cell.isExtraTurn) {

                ctx.fillStyle = "#fffacd";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("⚡", centerX, centerY);

            } else if (cell.isCoin) {

                ctx.fillStyle = "#fff8dc";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("🪙", centerX, centerY);

            }

            // Draw characters on top

            if (player.x === x && player.y === y) {

                ctx.fillStyle = "#000";

                ctx.fillText("👤", centerX, centerY);

            }

            if (enemy.x === x && enemy.y === y) {

                ctx.fillStyle = "#000";

                ctx.fillText("👹", centerX, centerY);

            }

            if (enemy2.x === x && enemy2.y === y && gameMode === 'hard') {

                ctx.fillStyle = "#000";

                ctx.fillText("😈", centerX, centerY);

            }

        }

    }

    // Update the side panel

    updateGameInfo();

}

// פונקציה לעדכון הפאנל הצדדי

function updateGameInfo() {

    const scoreElement = document.getElementById('scoreDisplay');

    const modeElement = document.getElementById('modeDisplay');

    if (scoreElement) {

        scoreElement.textContent = score;

    }

    if (modeElement) {

        modeElement.textContent = gameMode.toUpperCase();

        modeElement.style.color = gameMode === 'hard' ? '#cc0000' : '#0066cc';

        modeElement.style.fontWeight = gameMode === 'hard' ? 'bold' : 'normal';

    }

}

// פונקציה להוספת מטבע חדש כאשר מטבע נלקח

function spawnNewCoin() {

    let attempts = 0;

    const maxAttempts = 100;

    while (attempts < maxAttempts) {

        const x = Math.floor(Math.random() \* SIZE);

        const y = Math.floor(Math.random() \* SIZE);

        // בדיקה שהמיקום לא תפוס ושאין שם אלמנט מיוחד אחר

        if (!isPositionOccupied(x, y) &&

            !grid[y][x].isWall &&

            !grid[y][x].isExit &&

            !grid[y][x].isRandomizer &&

            !grid[y][x].isExtraTurn &&

            !grid[y][x].isCoin) {

            grid[y][x].isCoin = true;

            return;

        }

        attempts++;

    }

    console.log("Could not spawn new coin after multiple attempts");

}

document.addEventListener("keydown", (e) => {

    let dx = 0,

        dy = 0;

    switch (e.key) {

        case "ArrowUp":

            e.preventDefault(); // מניעת גלילת הדף

            dy = -1;

            break;

        case "ArrowDown":

            e.preventDefault(); // מניעת גלילת הדף

            dy = 1;

            break;

        case "ArrowLeft":

            e.preventDefault(); // מניעת גלילת הדף

            dx = -1;

            break;

        case "ArrowRight":

            e.preventDefault(); // מניעת גלילת הדף

            dx = 1;

            break;

        default:

            return;

    }

    const newX = player.x + dx;

    const newY = player.y + dy;

    if (newX >= 0 && newX < SIZE && newY >= 0 && newY < SIZE && !grid[newY][newX].isWall) {

        // שמירת מיקום קודם לבדיקת החלפת מקומות

        lastPlayerPosition = { x: player.x, y: player.y };

        lastEnemyPosition = { x: enemy.x, y: enemy.y };

        lastEnemy2Position = { x: enemy2.x, y: enemy2.y };

        // עדכון היסטוריית השחקן

        playerHistory.push({ x: player.x, y: player.y });

        if (playerHistory.length > 5) playerHistory.shift(); // שמירת 5 מהלכים אחרונים

        player.x = newX;

        player.y = newY;

        const cell = grid[newY][newX];

        // בדיקה לאיסוף מטבע

        if (cell.isCoin) {

            cell.isCoin = false;

            score += 1;

            spawnNewCoin(); // הוסף מטבע חדש במיקום אחר

        }

        // בדיקה לRandomizer - עם הפונקציה הבטוחה החדשה

        if (cell.isRandomizer) {

            grid[player.y][player.x].isRandomizer = false;

            placeRandomElementsSafe();

            validateInitialGrid();

            drawGrid();

        }

        if (cell.isExtraTurn) {

            cell.isExtraTurn = false;

            extraTurn = true;

        }

        if (cell.isExit) {

            Swal.fire({

                title: "🎉 You Escaped!",

                text: `Congratulations! You collected ${score} coins!`,

                icon: "success",

                confirmButtonText: "Play Again"

            }).then(() => {

                initGrid(gameMode);

            });

            return;

        }

        drawGrid();

        if (!extraTurn) {

            moveEnemy();

            if (gameMode === 'hard') {

                moveEnemy2();

            }

        } else {

            extraTurn = false;

        }

        drawGrid();

        // בדיקת תפיסה - כולל החלפת מקומות

        if (isPlayerCaught()) {

            Swal.fire({

                title: "💀 Caught!",

                text: "The enemy reached you.",

                icon: "error",

                confirmButtonText: "Try Again"

            }).then(() => {

                initGrid(gameMode);

            });

        }

    }

});

// פונקציה לבדיקת תפיסה משופרת

function isPlayerCaught() {

    // בדיקה רגילה - אותו מיקום עם enemy1

    if (player.x === enemy.x && player.y === enemy.y) {

        return true;

    }

    // בדיקת החלפת מקומות עם enemy1

    if (player.x === lastEnemyPosition.x && player.y === lastEnemyPosition.y &&

        enemy.x === lastPlayerPosition.x && enemy.y === lastPlayerPosition.y) {

        return true;

    }

    // בדיקות עבור enemy2 במוד קשה

    if (gameMode === 'hard') {

        // בדיקה רגילה - אותו מיקום עם enemy2

        if (player.x === enemy2.x && player.y === enemy2.y) {

            return true;

        }

        // בדיקת החלפת מקומות עם enemy2

        if (player.x === lastEnemy2Position.x && player.y === lastEnemy2Position.y &&

            enemy2.x === lastPlayerPosition.x && enemy2.y === lastPlayerPosition.y) {

            return true;

        }

    }

    return false;

}

// פונקציה לחיזוי מיקום השחקן המשופרת

function predictPlayerPosition() {

    // אם יש היסטוריה, נסה לחזות לפי דפוס

    if (playerHistory.length >= 2) {

        const lastMove = {

            dx: playerHistory[playerHistory.length - 1].x - playerHistory[playerHistory.length - 2].x,

            dy: playerHistory[playerHistory.length - 1].y - playerHistory[playerHistory.length - 2].y

        };

        const predictedX = player.x + lastMove.dx;

        const predictedY = player.y + lastMove.dy;

        if (predictedX >= 0 && predictedX < SIZE && predictedY >= 0 && predictedY < SIZE &&

            !grid[predictedY][predictedX].isWall) {

            return { x: predictedX, y: predictedY };

        }

    }

    // אחרת, חזור לשחקן הנוכחי

    return { x: player.x, y: player.y };

}

// פונקציה לבחירת אסטרטגיה

function chooseStrategy() {

    const distance = Math.abs(enemy.x - player.x) + Math.abs(enemy.y - player.y);

    // אם צמוד לשחקן - תקיפה ישירה

    if (distance === 1) {

        return 'attack';

    }

    // אם קרוב מאוד - נסה לחסום

    else if (distance <= 3) {

        return 'block';

    }

    // אם במרחק בינוני - רדיפה ישירה

    else if (distance <= 5) {

        return 'direct';

    }

    // אם רחוק - נסה לחזות

    else {

        return 'predict';

    }

}

// פונקציה לחיפוש מיקום תקיפה אגרסיבי

function findAttackPosition() {

    const dirs = [

        { x: 0, y: -1 }, { x: 0, y: 1 },

        { x: -1, y: 0 }, { x: 1, y: 0 },

        // הוסף מהלכים אלכסוניים לתקיפה יותר אגרסיבית

        { x: -1, y: -1 }, { x: 1, y: -1 },

        { x: -1, y: 1 }, { x: 1, y: 1 }

    ];

    let bestMove = { x: player.x, y: player.y, score: -1000 };

    for (const dir of dirs) {

        const nx = enemy.x + dir.x;

        const ny = enemy.y + dir.y;

        if (nx >= 0 && nx < SIZE && ny >= 0 && ny < SIZE && !grid[ny][nx].isWall) {

            // חשב ציון לפי קרבה לשחקן ומניעת בריחה

            const distanceToPlayer = Math.abs(nx - player.x) + Math.abs(ny - player.y);

            const blocksEscape = calculateEscapeBlocking(nx, ny);

            // העדף מיקומים שקרובים לשחקן וחוסמים בריחה

            const score = -distanceToPlayer \* 10 + blocksEscape \* 5;

            if (score > bestMove.score) {

                bestMove = { x: nx, y: ny, score: score };

            }

        }

    }

    return bestMove;

}

// פונקציה לחישוב כמה דרכי בריחה המיקום חוסם

function calculateEscapeBlocking(enemyX, enemyY) {

    const playerEscapeRoutes = [];

    const dirs = [

        { x: 0, y: -1 }, { x: 0, y: 1 },

        { x: -1, y: 0 }, { x: 1, y: 0 }

    ];

    // מצא את כל דרכי הבריחה של השחקן

    for (const dir of dirs) {

        const nx = player.x + dir.x;

        const ny = player.y + dir.y;

        if (nx >= 0 && nx < SIZE && ny >= 0 && ny < SIZE &&

            !grid[ny][nx].isWall && !(nx === enemyX && ny === enemyY)) {

            playerEscapeRoutes.push({ x: nx, y: ny });

        }

    }

    // חזור כמה דרכי בריחה נחסמו

    const totalRoutes = 4; // מקסימום 4 כיוונים

    const blockedRoutes = totalRoutes - playerEscapeRoutes.length;

    return blockedRoutes;

}

// פונקציה לחיפוש מיקום חסימה משופרת

function findBlockingPosition() {

    const playerNeighbors = [];

    const dirs = [

        { x: 0, y: -1 }, { x: 0, y: 1 },

        { x: -1, y: 0 }, { x: 1, y: 0 }

    ];

    // מצא את כל המיקומים הסמוכים לשחקן

    for (const dir of dirs) {

        const nx = player.x + dir.x;

        const ny = player.y + dir.y;

        if (nx >= 0 && nx < SIZE && ny >= 0 && ny < SIZE && !grid[ny][nx].isWall) {

            const distanceFromEnemy = Math.abs(enemy.x - nx) + Math.abs(enemy.y - ny);

            const escapeBlocking = calculateEscapeBlocking(nx, ny);

            // העדף מיקומים שקרובים לאויב וחוסמים יותר דרכי בריחה

            const score = -distanceFromEnemy + escapeBlocking \* 2;

            playerNeighbors.push({

                x: nx,

                y: ny,

                distance: distanceFromEnemy,

                score: score

            });

        }

    }

    // בחר את המיקום עם הציון הטוב ביותר

    if (playerNeighbors.length > 0) {

        playerNeighbors.sort((a, b) => b.score - a.score);

        return playerNeighbors[0];

    }

    return { x: player.x, y: player.y };

}

function moveEnemy() {

    const strategy = chooseStrategy();

    let target;

    switch (strategy) {

        case 'attack':

            target = findAttackPosition();

            break;

        case 'block':

            target = findBlockingPosition();

            break;

        case 'direct':

            target = { x: player.x, y: player.y };

            break;

        case 'predict':

            target = predictPlayerPosition();

            break;

        default:

            target = { x: player.x, y: player.y };

    }

    const path = aStar(grid[enemy.y][enemy.x], target);

    if (path.length > 1) {

        enemy = { x: path[1].x, y: path[1].y };

    }

}

function moveEnemy2() {

    // Enemy2 uses a simpler strategy - direct pursuit

    const target = { x: player.x, y: player.y };

    const path = aStar(grid[enemy2.y][enemy2.x], target);

    if (path.length > 1) {

        enemy2 = { x: path[1].x, y: path[1].y };

    }

}

function heuristic(a, b) {

    const manhattanDistance = Math.abs(a.x - b.x) + Math.abs(a.y - b.y);

    // בונוס אם המיקום קרוב לשחקן (עידוד רדיפה)

    const distanceToPlayer = Math.abs(a.x - player.x) + Math.abs(a.y - player.y);

    const playerProximityBonus = Math.max(0, 10 - distanceToPlayer);

    // עונש אם המיקום קרוב ליציאה (למנוע מהאויב להגן על היציאה)

    const distanceToExit = Math.abs(a.x - exit.x) + Math.abs(a.y - exit.y);

    const exitPenalty = distanceToExit < 3 ? 5 : 0;

    return manhattanDistance - playerProximityBonus + exitPenalty;

}

function getNeighbors(cell) {

    const dirs = [

        { x: 0, y: -1 }, { x: 0, y: 1 },

        { x: -1, y: 0 }, { x: 1, y: 0 }

    ];

    const neighbors = [];

    for (const dir of dirs) {

        const nx = cell.x + dir.x;

        const ny = cell.y + dir.y;

        if (nx >= 0 && nx < SIZE && ny >= 0 && ny < SIZE && !grid[ny][nx].isWall) {

            neighbors.push(grid[ny][nx]);

        }

    }

    return neighbors;

}

function getCost(x, y) {

    let cost = 1;

    if (grid[y][x].isRandomizer) cost += 3;

    if (grid[y][x].isExtraTurn) cost += 2;

    const dirs = [

        { x: 0, y: -1 }, { x: 0, y: 1 },

        { x: -1, y: 0 }, { x: 1, y: 0 }

    ];

    for (const d of dirs) {

        const nx = x + d.x,

            ny = y + d.y;

        if (nx >= 0 && nx < SIZE && ny >= 0 && ny < SIZE) {

            if (grid[ny][nx].isWall) cost += 1;

        }

    }

    return cost;

}

function aStar(start, goal) {

    const openSet = [start];

    const cameFrom = new Map();

    const gScore = new Map();

    const fScore = new Map();

    const key = (p) => `${p.x},${p.y}`;

    gScore.set(key(start), 0);

    fScore.set(key(start), heuristic(start, goal));

    while (openSet.length > 0) {

        openSet.sort((a, b) => fScore.get(key(a)) - fScore.get(key(b)));

        let current = openSet.shift();

        if (current.x === goal.x && current.y === goal.y) {

            const path = [current];

            while (cameFrom.has(key(current))) {

                current = cameFrom.get(key(current));

                path.unshift(current);

            }

            return path;

        }

        for (const neighbor of getNeighbors(current)) {

            const tempG = gScore.get(key(current)) + getCost(neighbor.x, neighbor.y);

            if (!gScore.has(key(neighbor)) || tempG < gScore.get(key(neighbor))) {

                cameFrom.set(key(neighbor), current);

                gScore.set(key(neighbor), tempG);

                fScore.set(key(neighbor), tempG + heuristic(neighbor, goal));

                if (!openSet.some(p => p.x === neighbor.x && p.y === neighbor.y)) {

                    openSet.push(neighbor);

                }

            }

        }

    }

    return [];

}

// Remove the auto-start call since we now auto-start in HTML

// initGrid() is now called by window.onload in game.html

     function startGame() {

         window.location.href = "game.html";

     }

     function openTutorial() {

         window.location.href = "tutorial.html";

     }

const canvas = document.getElementById("tutorialCanvas");

const ctx = canvas.getContext("2d");

const SIZE = 6;

const CELL\_SIZE = canvas.width / SIZE;

let currentStep = 1;

let score = 0;

let grid = [];

let player = { x: 0, y: 0 };

let enemy = { x: -1, y: -1 };

let enemy2 = { x: -1, y: -1 };

function createCell(x, y) {

    return {

        x,

        y,

        isWall: false,

        isExit: false,

        isRandomizer: false,

        isExtraTurn: false,

        isCoin: false

    };

}

function initGrid() {

    grid = [];

    for (let y = 0; y < SIZE; y++) {

        const row = [];

        for (let x = 0; x < SIZE; x++) {

            row.push(createCell(x, y));

        }

        grid.push(row);

    }

    player = { x: 0, y: 0 };

    enemy = { x: -1, y: -1 };

    enemy2 = { x: -1, y: -1 };

    score = 0;

}

function drawGrid() {

    ctx.clearRect(0, 0, canvas.width, canvas.height);

    for (let y = 0; y < SIZE; y++) {

        for (let x = 0; x < SIZE; x++) {

            const cell = grid[y][x];

            // Draw white background for all cells

            ctx.fillStyle = "#fff";

            ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

            // Draw border

            ctx.strokeStyle = "#ddd";

            ctx.lineWidth = 1;

            ctx.strokeRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

            // Draw cell contents with icons

            const centerX = x \* CELL\_SIZE + CELL\_SIZE / 2;

            const centerY = y \* CELL\_SIZE + CELL\_SIZE / 2;

            ctx.font = `${CELL\_SIZE \* 0.7}px Arial`;

            ctx.textAlign = "center";

            ctx.textBaseline = "middle";

            if (cell.isWall) {

                ctx.fillStyle = "#444";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#fff";

                ctx.fillText("🧱", centerX, centerY);

            } else if (cell.isExit) {

                ctx.fillStyle = "#e8f5e8";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("🚪", centerX, centerY);

            } else if (cell.isRandomizer) {

                ctx.fillStyle = "#e8f8ff";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("🔄", centerX, centerY);

            } else if (cell.isExtraTurn) {

                ctx.fillStyle = "#fffacd";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("⚡", centerX, centerY);

            } else if (cell.isCoin) {

                ctx.fillStyle = "#fff8dc";

                ctx.fillRect(x \* CELL\_SIZE, y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

                ctx.fillStyle = "#000";

                ctx.fillText("🪙", centerX, centerY);

            }

            // Draw characters on top

            if (player.x === x && player.y === y) {

                ctx.fillStyle = "#000";

                ctx.fillText("👤", centerX, centerY);

            }

            if (enemy.x === x && enemy.y === y) {

                ctx.fillStyle = "#000";

                ctx.fillText("👹", centerX, centerY);

            }

            if (enemy2.x === x && enemy2.y === y) {

                ctx.fillStyle = "#000";

                ctx.fillText("😈", centerX, centerY);

            }

        }

    }

}

function updateStepIndicator() {

    // Update step number

    document.getElementById('stepNumber').textContent = currentStep;

    // Update dots

    for (let i = 1; i <= 6; i++) {

        const dot = document.getElementById(`dot${i}`);

        dot.classList.remove('active', 'completed');

        if (i < currentStep) {

            dot.classList.add('completed');

        } else if (i === currentStep) {

            dot.classList.add('active');

        }

    }

}

document.addEventListener("keydown", (e) => {

    let dx = 0, dy = 0;

    switch (e.key) {

        case "ArrowUp":

            e.preventDefault(); // מניעת גלילת הדף

            dy = -1;

            break;

        case "ArrowDown":

            e.preventDefault(); // מניעת גלילת הדף

            dy = 1;

            break;

        case "ArrowLeft":

            e.preventDefault(); // מניעת גלילת הדף

            dx = -1;

            break;

        case "ArrowRight":

            e.preventDefault(); // מניעת גלילת הדף

            dx = 1;

            break;

        default: return;

    }

    const newX = player.x + dx;

    const newY = player.y + dy;

    if (newX >= 0 && newX < SIZE && newY >= 0 && newY < SIZE && !grid[newY][newX].isWall) {

        player.x = newX;

        player.y = newY;

        const cell = grid[newY][newX];

        // Check for coin collection

        if (cell.isCoin) {

            cell.isCoin = false;

            score++;

        }

        // Check for extra turn

        if (cell.isExtraTurn) {

            cell.isExtraTurn = false;

            Swal.fire({

                title: "⚡ Extra Turn!",

                text: "You got an extra turn! The enemy won't move this time.",

                icon: "success",

                timer: 1500,

                showConfirmButton: false

            });

        }

        // Check for randomizer

        if (cell.isRandomizer) {

            cell.isRandomizer = false;

            Swal.fire({

                title: "🔄 Randomizer!",

                text: "The maze layout has been shuffled!",

                icon: "info",

                timer: 1500,

                showConfirmButton: false

            });

        }

        drawGrid();

        // In tutorial, enemies don't move - they stay in their corners

        // This allows players to learn at their own pace

        // Check for enemy collision

        if ((player.x === enemy.x && player.y === enemy.y) ||

            (player.x === enemy2.x && player.y === enemy2.y)) {

            Swal.fire({

                title: "💀 Caught!",

                text: "The enemy caught you! Try again.",

                icon: "error",

                confirmButtonText: "Try Again"

            }).then(() => {

                restartCurrentStep();

            });

            return;

        }

        // Check for exit

        if (cell.isExit) {

            const isLastStep = currentStep === 6;

            let message = "You reached the exit!";

            if (score > 0) {

                message += ` You collected ${score} coins!`;

            }

            Swal.fire({

                title: "🎉 Great job!",

                text: message,

                icon: "success",

                confirmButtonText: isLastStep ? "Start Game!" : "Continue"

            }).then(() => {

                if (isLastStep) {

                    window.location.href = "game.html";

                } else {

                    nextStep();

                }

            });

        }

    }

});

// Enemies in tutorial don't move - they stay in their fixed positions

// This allows players to learn game mechanics without time pressure

function restartCurrentStep() {

    const steps = [step1, step2, step3, step4, step5, step6];

    if (currentStep >= 1 && currentStep <= 6) {

        steps[currentStep - 1]();

    }

}

function nextStep() {

    if (currentStep < 6) {

        currentStep++;

        const steps = [step1, step2, step3, step4, step5, step6];

        steps[currentStep - 1]();

        updateStepIndicator();

    }

}

function step1() {

    document.getElementById("instruction").innerHTML = "Use arrow keys ⬆️ ⬇️ ⬅️ ➡️ to move the player 👤<br><strong>Try moving around!</strong><br><em>Note: In tutorial mode, enemies don't move</em>";

    initGrid();

    drawGrid();

    updateStepIndicator();

}

function step2() {

    document.getElementById("instruction").innerHTML = "Now reach the green exit 🚪<br><strong>Goal: Get to the exit door!</strong>";

    initGrid();

    grid[SIZE - 1][SIZE - 1].isExit = true;

    drawGrid();

    updateStepIndicator();

}

function step3() {

    document.getElementById("instruction").innerHTML = "Collect the gold coin 🪙 and reach the exit 🚪<br><strong>Coins give you points!</strong>";

    initGrid();

    grid[2][3].isCoin = true;

    grid[SIZE - 1][SIZE - 1].isExit = true;

    drawGrid();

    updateStepIndicator();

}

function step4() {

    document.getElementById("instruction").innerHTML = "Avoid the red enemy 👹 and reach the exit 🚪<br><strong>Navigate around the enemy!</strong>";

    initGrid();

    enemy = { x: SIZE - 1, y: 0 }; // Top-right corner

    grid[SIZE - 1][SIZE - 1].isExit = true;

    drawGrid();

    updateStepIndicator();

}

function step5() {

    document.getElementById("instruction").innerHTML = "Collect power-ups! ⚡ = Extra Turn, 🔄 = Randomizer<br><strong>Learn what each power-up does!</strong>";

    initGrid();

    enemy = { x: 0, y: SIZE - 1 }; // Bottom-left corner

    grid[1][3].isExtraTurn = true;

    grid[3][1].isRandomizer = true;

    grid[2][4].isCoin = true;

    grid[SIZE - 1][SIZE - 1].isExit = true;

    drawGrid();

    updateStepIndicator();

}

function step6() {

    document.getElementById("instruction").innerHTML = "Hard Mode Preview: 2 enemies 👹😈! Collect coins and escape!<br><strong>In the real game, enemies will chase you!</strong>";

    initGrid();

    enemy = { x: SIZE - 1, y: 0 }; // Top-right corner

    enemy2 = { x: 0, y: SIZE - 1 }; // Bottom-left corner

    // Add some walls

    grid[2][2].isWall = true;

    grid[2][3].isWall = true;

    grid[3][2].isWall = true;

    // Add coins and power-ups

    grid[1][2].isCoin = true;

    grid[4][1].isCoin = true;

    grid[1][4].isExtraTurn = true;

    grid[SIZE - 2][SIZE - 2].isExit = true;

    // Update button text for final step

    const btn = document.getElementById("nextBtn");

    btn.textContent = "🎮 Start Game!";

    btn.className = "btn btn-success";

    btn.onclick = () => window.location.href = "game.html";

    drawGrid();

    updateStepIndicator();

}

// Start the tutorial

step1();

[[{"x": 0, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 0, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 0, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 0, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 1, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 1, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 1, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 1, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 2, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 2, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 2, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 2, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 3, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 3, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 3, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 4, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 4, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 4, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 4, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 4, "isWall": false, "isExit": false, "isRandomizer": true, "isExtraTurn": false}], [{"x": 0, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 5, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": true}, {"x": 7, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 5, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 5, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 6, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 6, "isWall": false, "isExit": false, "isRandomizer": true, "isExtraTurn": false}, {"x": 5, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 6, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 6, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 6, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 7, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 7, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": true}, {"x": 4, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 8, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 8, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 9, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 9, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 9, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 10, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 10, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 10, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 10, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}], [{"x": 0, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 1, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 2, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 3, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 4, "y": 11, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 5, "y": 11, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 6, "y": 11, "isWall": false, "isExit": true, "isRandomizer": false, "isExtraTurn": false}, {"x": 7, "y": 11, "isWall": true, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 8, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 9, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 10, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}, {"x": 11, "y": 11, "isWall": false, "isExit": false, "isRandomizer": false, "isExtraTurn": false}]]

**# 🎮 Maze Run Game**

משחק מבוך מתקדם עם בינה מלאכותית חכמה שרודפת אחריך!

**## 📁 מבנה הפרויקט**

```

MazeRun/

├── html/           # קבצי HTML

│   ├── index.html     # עמוד הבית

│   ├── game.html      # עמוד המשחק

│   └── tutorial.html  # עמוד ההדרכה

├── js/             # קבצי JavaScript

│   ├── game.js        # לוגיקת המשחק הראשית

│   ├── tutorial.js    # לוגיקת ההדרכה

│   └── index.js       # ניווט בין עמודים

├── css/            # קבצי CSS

│   └── style.css      # עיצוב המשחק

├── grid.json       # קובץ הגדרות המבוך

└── README.md       # התיעוד הזה

```

**## 🚀 איך להתחיל**

1. פתח את `html/index.html` בדפדפן

2. בחר "🎮 Start Game" או "📘 Tutorial"

3. תהנה מהמשחק!

**## 🎯 איך לשחק**

**### מטרת המשחק**

- אסוף מטבעות 🪙 והגע ליציאה הירוקה 🚪 לפני שהאויבים יתפסו אותך!

**### בקרות**

- **\*\*חצים\*\***: תנועה (למעלה, למטה, שמאלה, ימינה)

**### רמות קושי**

- **\*\*Normal Mode\*\***: אויב אחד עם 25% קירות

- **\*\*Hard Mode\*\***: שני אויבים עם 35% קירות - אתגר אולטימטיבי!

**### אלמנטים מיוחדים**

- 🪙 **\*\*מטבעות\*\***: אסוף מטבעות להעלאת הניקוד

- ⚡ **\*\*תור נוסף\*\***: נותן לך תור נוסף

- 🔄 **\*\*רנדומיזר\*\***: משנה את פריסת המבוך

**## 🤖 בינה מלאכותית מתקדמת**

המחשב משתמש באלגוריתם A\* משופר עם 4 אסטרטגיות:

**### 1. מצב תקיפה (מרחק 1)**

- תקיפה אגרסיבית עם 8 כיוונים (כולל אלכסוניים)

- חישוב חסימת דרכי בריחה

**### 2. מצב חסימה (מרחק 2-3)**

- ניסיון לסגור על השחקן

- העדפת מיקומים שחוסמים בריחה

**### 3. רדיפה ישירה (מרחק 4-5)**

- מסלול ישיר לשחקן

**### 4. חיזוי (מרחק 6+)**

- למידה מהיסטוריית התנועות (5 מהלכים אחרונים)

- חיזוי המהלך הבא של השחקן

**## 🔧 תכונות טכניות**

**### מניעת תקיעות**

- בדיקת קישוריות במפה

- וולידציה אוטומטית של פתירות המבוך

- רנדומיזציה בטוחה של מכשולים

**### בדיקת תפיסה משופרת**

- זיהוי התנגשות רגילה

- זיהוי החלפת מקומות (crossing paths)

**### אופטימיזציות**

- Heuristic משופר עם בונוסים וקנסות

- חישוב עלויות דינמי

- מעקב אחר היסטוריית השחקן

**## 🎨 עיצוב**

- ממשק נקי ואינטואיטיבי

- צבעים ברורים לכל אלמנט

- התראות יפות עם SweetAlert2

- רספונסיבי לכל הגדלים

- לוח משחק 30x30 עם אייקונים ברורים

**## 🔄 תכונות חדשות**

- ✅ מערכת ניקוד מתקדמת עם מטבעות

- ✅ שני אויבים במוד קשה

- ✅ מערכת רמות קושי מתקדמת

- ✅ אלגוריתם A\* משופר עם אסטרטגיות מרובות

- ✅ מערכת חיזוי התנהגות השחקן

- ✅ שמירת מוד המשחק אחרי ניצחון/הפסד

- ✅ תיקון בעיית הרנדומיזר שגרם לתקיעות

- ✅ ארגון הקוד בתיקיות מסודרות

- ✅ בדיקת החלפת מקומות

- ✅ מערכת הדרכה אינטראקטיבית

**## 🎮 טיפים למשחק**

1. **\*\*אסוף מטבעות\*\*** - כל מטבע נותן נקודה ומטבע חדש מופיע

2. **\*\*השתמש ברנדומיזר בחכמה\*\*** - יכול לעזור או להזיק

3. **\*\*תכנן מראש\*\*** - המחשב חכם ולומד מההתנהגות שלך

4. **\*\*נסה להימנע מפינות\*\*** - המחשב טוב בחסימה

5. **\*\*השתמש בקירות\*\*** - יכולים לעזור לך לברוח

6. **\*\*למד את דפוסי התנועה\*\*** - המחשב משתמש בהיסטוריה לחיזוי

7. **\*\*במוד קשה\*\*** - שני אויבים רודפים אחריך!

**## 📚 תיעוד נוסף**

- `A\_Star\_Algorithm\_Explanation.md` - הסבר מפורט על אלגוריתם A\*

- `MazeRun\_Presentation.md` - מצגת על הפרויקט

- `SRS.docx` - מסמך דרישות המערכת

**## 🎯 אלמנטי המשחק**

- 👤 **\*\*שחקן\*\*** - אתה!

- 👹 **\*\*אויב 1\*\*** - רודף אחריך עם בינה מלאכותית

- 😈 **\*\*אויב 2\*\*** - מופיע במוד קשה בלבד

- 🚪 **\*\*יציאה\*\*** - המטרה שלך

- 🪙 **\*\*מטבעות\*\*** - אסוף להעלאת הניקוד

- ⚡ **\*\*תור נוסף\*\*** - נותן לך תור נוסף

- 🔄 **\*\*רנדומיזר\*\*** - משנה את המבוך

- 🧱 **\*\*קירות\*\*** - מכשולים במבוך

**## 🔗 קישורים**

- 🎮 **\*\*[שחק עכשיו](**https://rotemrosenberg.github.io/MazeRun/html/index.html**)\*\*** - המשחק המלא

- 📂 **\*\*[GitHub Repository](**https://github.com/RotemRosenberg/MazeRun**)\*\*** - קוד המקור

---

**\*\*נוצר על ידי:\*\*** רותם רועי מיכאל ואלעד

**\*\*גרסה:\*\*** 2.0

**\*\*תאריך:\*\*** יולי 2025