

Table of Contents

[**1.** **Introduction** 5](#_Toc166938169)

[**2.** **Checklist Review** 5](#_Toc166938170)

[**3.** **Testing** 7](#_Toc166938171)

[**3Black Box Testing** 7](#_Toc166938172)

[**White Box Testing** 10](#_Toc166938173)

[**White Box Testing Table** 10](#_Toc166938174)

[**Conclusion** 22](#_Toc166938175)

[**4.Evaluation** 24](#_Toc166938176)

[Known bugs/weaknesses in the game 24](#_Toc166938177)

[What works well? 24](#_Toc166938178)

[Potential Improvements 24](#_Toc166938179)

[Additional Features 24](#_Toc166938180)

[Extensibility 24](#_Toc166938181)

[**5. Video Demonstration** 24](#_Toc166938182)

[**6. Conclusion** 24](#_Toc166938183)

[**7. References** 24](#_Toc166938184)

**Assignment Checklist**

|  |  |
| --- | --- |
| D+ to D- | |
| The start Game Button is hidden when clicking. |  |
| * The player cannot move until the start button has been clicked. |  |
| Maze wall collision with the player |  |
| * Multiple points of collision for the maze walls (top left and top right for the up direction) |  |
| Point element collision with the player. |  |
| * Points are hidden from the maze. |  |
| The score p tag is updated for every point the player collects. |  |
| A game-over message appears after collecting all the points in the maze. |  |
| Once the game has ended, the player can no longer move. |  |
| The game is over when the player collides with an enemy character. |  |
| * Display the death animation upon enemy collision (dead CSS class) |  |
| C+ to C- | |
| Randomise the position of enemies at the start of the game. |  |
| Prevent the enemies from being created outside of the maze. |  |
| * Prevent enemies from being created where there are walls. |  |
| Enemies randomly move around in the maze. |  |
| * Enemy movement has wall collision (cannot move through walls) |  |
| * The enemy does not stop upon collision with the wall instead it moves in a new direction. |  |
| Enemies stop moving when the game-over state has been reached. |  |
| Reset button instead of game over (resets game state) |  |
| Implement the arrow buttons. The player will continue moving in that direction when an arrow button is clicked. |  |
| * The Arrow GUI button movement does not impact the arrow key movement. |  |
| B+ to B- | |
| At the end of the game, ask the player to enter their name. |  |
| * Save the name and score using local storage. |  |
| Display the scores of all the players on the leaderboard. |  |
| The leaderboard should be organised in order from the highest score to the lowest score. |  |
| Add the lives through JavaScript (not the HTML) at the start of the game. |  |
| Remove a life instead of the Game Over/restart button when the player collides with an enemy. |  |
| * Add the hit class and prevent the player from moving for 1.5 seconds while the animation plays. |  |
| Display the game over/restart button when all three lives are lost. |  |
| A+ to A- | |
| Once all the points are collected, reconfigure the maze and create a new layout for each level. |  |
| * Implement increasing difficulty. As the game goes on, it should get more challenging. |  |
| Create an infinite number of levels (not premade mazes but randomly created) |  |
| * Prevent an impossible-to-solve maze. |  |
| Add two unique features to the game. The better the feature, the more marks |  |
| * Feature One (replace with the feature) |  |
| * Feature Two (replace with the feature) |  |

# **Introduction**

This project focuses on creating a web-based maze game where players navigate through randomly generated mazes, collect points, and avoid enemies. Inspired by classic arcade games, the game aims to provide a fun and challenging experience for all players.

**Key features of the game include:**

* **Random Maze Generation:** Each level has a unique maze layout.
* **Player Controls:** Players use keyboard arrow keys or on-screen buttons to move.
* Enemie**s:** Randomly moving enemies add difficulty. Players must avoid them to survive.
* **Points Collection:** Players collect points throughout the maze to increase their score.
* **Lives System:** Players have a limited number of lives, losing one upon collision with an enemy. The game ends when all lives are lost.
* **Leaderboard:** Player scores are recorded and displayed in a leaderboard to encourage competition.
* **Increasing Difficulty:** The game gets harder as players progress through levels, with more challenging mazes and additional enemies.

The game is developed using JavaScript mainly, but also HTML, and CSS. Various algorithms ensure the maze is solvable and the gameplay remains fair. This report covers the development process, testing methods, identified bugs, and suggestions for future improvements.

In the following sections, we will review the checklist, game features, testing approaches, performance evaluation, known bugs, and potential improvements.

# **Checklist Review**

This section provides an overview of the features implemented in the web-based maze game, along with those that are yet to be completed. The following features have been successfully implemented:

* **Start Game Button:** Hidden upon clicking.
* **Player Movement Control:** Player can only move after the start button is clicked.
* **Maze Wall Collision:** The player collides with maze walls accurately, with multiple points of collision detection.
* **Point Collection:** Points within the maze are hidden and collected by the player, updating the score accordingly.
* **Game Over Mechanics:** A game-over message is displayed when all points are collected or the player collides with an enemy. Player movement is disabled upon game over.
* **Enemy Positioning and Movement:** Enemies are randomly positioned within the maze, avoiding walls. They move randomly and collide with walls, changing direction upon collision. Enemy movement ceases when the game is over.
* **Lives System:** Lives are tracked and displayed via JavaScript. The player loses a life upon collision with an enemy, and the game ends when all lives are lost.
* **Score and Leaderboard:** Player scores are recorded, stored in local storage, and displayed on a leaderboard in descending order.
* **Player Controls:** Players can use keyboard arrow keys or on-screen buttons to navigate the maze.
* **Death Animation:** Displayed upon collision with an enemy.

However, the following features have not been fully implemented:

* **Increasing Difficulty: The game does not get progressively harder as players advance through levels.**
* **Infinite Levels: The game does not generate an infinite number of levels with randomly created mazes.**
* **Solvable Maze Guarantee: There is no mechanism to ensure that the generated maze is always solvable.**

**Additional Features**

* Dynamic Objectives: The game includes dynamically generated objectives based on the level.
* Colour Options: Players can change the colour of their character and enemies through the game interface.
* **Heart-Shaped Lives Indicator:** I implemented a heart-shaped design to represent the player's lives, enhancing the visual appeal and clearly indicating the number of remaining lives.

In summary, while many core features and functionalities have been successfully implemented, the game still requires improvements to meet the full project goals. The missing features, including increasing difficulty, infinite levels, and ensuring solvable mazes, need to be addressed to enhance the gameplay experience.

By addressing these areas, the game will provide a more engaging and challenging experience, keeping players motivated and ensuring fair play. Further testing and development are needed to achieve these objectives.

# **Testing**

### **Black Box Testing**

Black box testing focuses on testing the software functionality without knowing the internal code structure. Below is a summary of black box testing performed on the game.

#### **Black Box Testing Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test Case Description** | **Input** | **Expected Output** | **Actual Output** | **Pass/Fail** |
| TC-01 | Game Initialization | N/A | Game initializes, score is 0, no errors | As expected | Pass |
| TC-02 | Player Movement | Arrow keys (up, down, left, right) | Player moves accordingly, no wall collisions | As expected | Pass |
| TC-03 | Enemy Movement | N/A | Enemies move randomly | As expected | Pass |
| TC-04 | Start Game | Click Start Button | Game starts, player can move | As expected | Pass |
| TC-05 | Level Up | Collect all points | Level increases, new maze generated | As expected | Pass |
| TC-06 | Score Update | Collect points | Score increases by 1 for each point collected | As expected | Pass |
| TC-07 | Game Over | Lose all lives | Game over screen displayed, player cannot move | As expected | Pass |
| TC-08 | Name Submission | Enter name in input field | Name appears on leaderboard | As expected | Pass |
| TC-09 | Maze Solvability | Play levels 1-5 | All levels are solvable | As expected | Pass |
| TC-10 | Maze Solvability (Level 6+) | Play level 6+ | Maze becomes less solvable, points enclosed by obstacles | Points enclosed by obstacles | Fail |
| TC-11 | Checklist Updates upon Level Completion | Complete a level | All checklist items should update for the new level | Checklist items remain the same from the previous level | Fail |

#### **Conclusion of Black Box Testing**

The black box testing revealed that most of the game functionalities work as expected. However, there is an issue with the maze generation for levels 6 and above, where some points become enclosed by obstacles, making the maze unsolvable.

### **White Box Testing**

White box testing involves analyzing the internal code structure and logic. Below is a summary of white box testing performed on the game.

### **White Box Testing Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Test Case Description** | **Code Segment** | **Expected Output** | **Actual Output** | **Pass/Fail** |
| WT-01 | Initialization and Variables | javascript<br>let upPressed = false;<br>let downPressed = false;<br>...<br>const main = document.querySelector('main'); | Variables initialized correctly, no errors | As expected | Pass |
| WT-02 | generateMaze Function | javascript<br>function generateMaze(level, simplicity) {<br>...<br>}<br> | Function switches between maze generation algorithms | As expected | Pass |
| WT-03 | generateMazeInternal Function | javascript<br>function generateMazeInternal(dimensions, level, simplicity) {<br>...<br>}<br> | Maze initialized, walls, enemies, and points placed | As expected | Pass |
| WT-04 | isSolvable Function | javascript<br>function isSolvable(maze) {<br>...<br>}<br> | Correctly identifies solvability | As expected | Pass |
| WT-05 | isPointAccessible Function | javascript<br>function isPointAccessible(maze, y, x) {<br>...<br>}<br> | Ensures points are not enclosed | As expected | Pass |
| WT-06 | ensurePointsAccessible Function | javascript<br>function ensurePointsAccessible(maze) {<br>...<br>}<br> | Re-generates maze if points are inaccessible | As expected | Pass |
| WT-07 | Player Movement Key Event Handlers | javascript<br>function keyDown(event) {<br>...<br>}<br>function keyUp(event) {<br>...<br>}<br> | Key event handlers update movement flags correctly | As expected | Pass |
| WT-08 | Player-Wall Collision Detection | javascript<br>function checkWallCollisions(nextTop, nextBottom, nextLeft, nextRight) {<br>...<br>}<br> | Accurate detection of wall collisions | As expected | Pass |
| WT-09 | Player-Enemy Collision Detection | javascript<br>function checkEnemyCollision() {<br>...<br>}<br> | Correctly identifies collisions and updates game state | As expected | Pass |
| WT-10 | Level Complete Logic | javascript<br>function levelComplete() {<br>...<br>}<br> | Correctly transitions to the next level | As expected | Pass |
| WT-11 | Game Over Logic | javascript<br>function gameOver() {<br>...<br>}<br> | Correctly displays game over screen and stops movement | As expected | Pass |

#### **Test Plan**

1. **Initialization and Variables**
   1. **Objective**: Ensure variables and game initialization are correct.
   2. **Code**:

|  |
| --- |
| 1. **let** upPressed = **false**; 2. **let** downPressed = **false**; 3. **let** leftPressed = **false**; 4. **let** rightPressed = **false**; 5. **let** playerCanMove = **false**; 6. localStorage.removeItem('leaderboard'); 7. **let** score = 0; 8. **const** main = document.querySelector('main'); |

* 1. **Result**: Variables initialized correctly, no errors.

1. **Maze Generation Functions**
   1. **generateMaze Function**:
      1. **Objective**: Ensure maze generation works correctly for different levels.
      2. **Code**:

|  |
| --- |
| 1. **function** generateMaze(level, simplicity) { 2. **const** dimensions = Math.min(10 + Math.floor(level / 2), 13); 3. document.documentElement.style.setProperty('--dimensions', dimensions); 4. **let** maze; 5. **if** (level <= 5) { 6. **do** { 7. maze = generateMazeInternal(dimensions, level, simplicity); 8. } **while** (!isSolvable(maze)); 9. } **else** { 10. maze = generateNewMaze(dimensions, level); 11. } 12. console.log(`Level **${**level**}** maze:`); 13. console.log(maze); 14. **return** maze; 15. } |

* + 1. **Result**: Function switches between maze generation algorithms based on the level.
  1. **generateMazeInternal Function**:
     1. **Objective**: Ensure the internal maze generation logic is correct.
     2. **Code**:

|  |
| --- |
| 1. **function** generateMazeInternal(dimensions, level, simplicity) { 2. **let** maze = Array(dimensions).fill().map(() => Array(dimensions).fill(0)); 3. maze[1][1] = 2; 4. **for** (**let** i = 0; i < dimensions; i++) { 5. maze[i][0] = 1; 6. maze[i][dimensions - 1] = 1; 7. maze[0][i] = 1; 8. maze[dimensions - 1][i] = 1; 9. } 10. **const** wallProbability = 0.2 + (level \* 0.05) - (simplicity \* 0.05); 11. **for** (**let** y = 2; y < dimensions - 2; y++) { 12. **for** (**let** x = 2; x < dimensions - 2; x++) { 13. **if** (Math.random() < wallProbability && maze[y][x] !== 2) { 14. maze[y][x] = 1; 15. } 16. } 17. } 18. **let** numEnemies = 3; 19. **for** (**let** i = 0; i < numEnemies; i++) { 20. **let** enemyY, enemyX; 21. **do** { 22. enemyY = Math.floor(Math.random() \* (dimensions - 2)) + 1; 23. enemyX = Math.floor(Math.random() \* (dimensions - 2)) + 1; 24. } **while** (maze[enemyY][enemyX] !== 0); 25. maze[enemyY][enemyX] = 3; 26. } 27. **const** numPoints = Math.floor((dimensions - 2) \* (dimensions - 2) \* (1 - wallProbability)); 28. **for** (**let** i = 0; i < numPoints; i++) { 29. **let** pointY, pointX; 30. **do** { 31. pointY = Math.floor(Math.random() \* (dimensions - 2)) + 1; 32. pointX = Math.floor(Math.random() \* (dimensions - 2)) + 1; 33. } **while** (maze[pointY][pointX] !== 0); 34. maze[pointY][pointX] = 0; 35. } 36. **return** maze; 37. } |

* + 1. **Result**: Maze initialized correctly, walls, enemies, and points placed randomly.
  1. **isSolvable Function**:
     1. **Objective**: Ensure the maze is solvable.
     2. **Code**:

|  |
| --- |
| 1. **function** isSolvable(maze) { 2. **for** (**let** y = 1; y < dimensions - 1; y++) { 3. **for** (**let** x = 1; x < dimensions - 1; x++) { 4. **if** (maze[y][x] == 0 || maze[y][x] == 3) { 5. **if** ( 6. maze[y - 1][x] === 1 && 7. maze[y + 1][x] === 1 && 8. maze[y][x - 1] === 1 && 9. maze[y][x + 1] === 1 10. ) { 11. console.log(`Unsolvable maze detected at ({y}, {x})`); 12. **return** **false**; 13. } 14. **let** accessible = **false**; 15. **for** (**let** dy = -1; dy <= 1; dy++) { 16. **for** (**let** dx = -1; dx <= 1; dx++) { 17. **if** ( 18. y + dy >= 0 && 19. y + dy < dimensions && 20. x + dx >= 0 && 21. x + dx < dimensions && 22. maze[y + dy][x + dx] !== 1 && 23. maze[y + dy][x + dx] !== 3 24. ) { 25. accessible = **true**; 26. **break**; 27. } 28. } 29. **if** (accessible) **break**; 30. } 31. **if** (!accessible) { 32. console.log(`Inaccessible point or enemy detected at ({y}, {x})`); 33. **return** **false**; 34. } 35. **if** ( 36. maze[y - 1][x - 1] === 1 && 37. maze[y - 1][x] === 1 && 38. maze[y - 1][x + 1] === 1 && 39. maze[y][x - 1] === 1 && 40. maze[y][x + 1] === 1 && 41. maze[y + 1][x - 1] === 1 && 42. maze[y + 1][x] === 1 && 43. maze[y + 1][x + 1] === 1 44. ) { 45. console.log(`Point inside an obstacle block detected at ({y}, {x})`); 46. **return** **false**; 47. } 48. } 49. } 50. } 51. **return** **true**; 52. } |

* + 1. **Result**: Function correctly identifies solvability by checking if points or enemies are completely enclosed.
  1. **isPointAccessible Function**:
     1. **Objective**: Ensure points are accessible.
     2. **Code**:  
        ```javascript  
        function isPointAccessible(maze, y, x) {

|  |
| --- |
| 1. const visited = new Set(); 2. const queue = [[y, x]]; 3. visited.add(`{y},{x}`); 4. while (queue.length > 0) { 5. const [currentY, currentX] = queue.shift(); 6. if (currentY === 0 || currentX === 0 || currentY === maze.length - 1 || currentX === maze[0].length - 1) { 7. return true; 8. } 9. const neighbors = [ 10. [currentY - 1, currentX], 11. [currentY + 1, currentX], 12. [currentY, currentX - 1], 13. [currentY, currentX + 1], 14. ]; 15. for (const [neighborY, neighborX] of neighbors) { 16. if ( 17. neighborY >= 0 && 18. neighborY < maze.length && 19. neighborX >= 0 && 20. neighborX < maze[0].length && 21. maze[neighborY][neighborX] === 0 && 22. !visited.has(`{neighborY},{neighborX}`) 23. ) { 24. visited.add(`{neighborY},{neighborX}`); 25. Queue.push 26. ([neighborY, neighborX]); |

|  |
| --- |
| }  }  }  return false;  }  ```  - \*\*Result\*\*: Ensures points are not enclosed by checking if there's a path to the edge of the maze.  - \*\*ensurePointsAccessible Function\*\*:  - \*\*Objective\*\*: Re-generate maze if points are inaccessible.  - \*\*Code\*\*:  ```javascript  function ensurePointsAccessible(maze) {  for (let y = 0; y < maze.length; y++) {  for (let x = 0; x < maze[y].length; x++) {  if (maze[y][x] === 0 && !isPointAccessible(maze, y, x)) {  console.log(`Inaccessible point detected at ({y}, {x})`);  return generateMaze(level, simplicity);  }  }  }  return maze;  }  ```  - \*\*Result\*\*: Re-generates the maze if any points are inaccessible. |

1. **Player Movement Logic**
   1. **Key Event Handlers**:
      1. **Objective**: Ensure key event handlers for player movement work correctly.
      2. **Code**:

|  |
| --- |
| 1. **function** keyDown(event) { 2. **if** (playerCanMove) { 3. event.preventDefault(); 4. **switch** (event.key) { 5. **case** 'ArrowUp': 6. upPressed = **true**; 7. **break**; 8. **case** 'ArrowDown': 9. downPressed = **true**; 10. **break**; 11. **case** 'ArrowLeft': 12. leftPressed = **true**; 13. **break**; 14. **case** 'ArrowRight': 15. rightPressed = **true**; 16. **break**; 17. } 18. } 19. } 20. **function** keyUp(event) { 21. **if** (playerCanMove) { 22. **switch** (event.key) { 23. **case** 'ArrowUp': 24. upPressed = **false**; 25. **break**; 26. **case** 'ArrowDown': 27. downPressed = **false**; 28. **break**; 29. **case** 'ArrowLeft': 30. leftPressed = **false**; 31. **break**; 32. **case** 'ArrowRight': 33. rightPressed = **false**; 34. **break**; 35. } 36. } 37. } 38. document.addEventListener('keydown', keyDown); 39. document.addEventListener('keyup', keyUp); |

* + 1. **Result**: Key event handlers correctly update movement flags.

1. **Player and Enemy Collision**
   1. **Player-Wall Collision**:
      1. **Objective**: Ensure player movement checks for wall collisions.
      2. **Code**:

|  |
| --- |
| 1. **function** checkWallCollisions(nextTop, nextBottom, nextLeft, nextRight) { 2. **let** topLeftElement = document.elementFromPoint(nextLeft, nextTop); 3. **let** topRightElement = document.elementFromPoint(nextRight, nextTop); 4. **let** bottomLeftElement = document.elementFromPoint(nextLeft, nextBottom); 5. **let** bottomRightElement = document.elementFromPoint(nextRight, nextBottom); 6. **if** (downPressed) { 7. **if** (bottomLeftElement.classList.contains('wall') || bottomRightElement.classList.contains('wall')) { 8. **return** **false**; 9. } **else** { 10. **return** **true**; 11. } 12. } **else** **if** (upPressed) { 13. **if** (topLeftElement.classList.contains('wall') || topRightElement.classList.contains('wall')) { 14. **return** **false**; 15. } **else** { 16. **return** **true**; 17. } 18. } **else** **if** (leftPressed) { 19. **if** (topLeftElement.classList.contains('wall') || bottomLeftElement.classList.contains('wall')) { 20. **return** **false**; 21. } **else** { 22. **return** **true**; 23. } 24. } **else** **if** (rightPressed) { 25. **if** (bottomRightElement.classList.contains('wall') || topRightElement.classList.contains('wall')) { 26. **return** **false**; 27. } **else** { 28. **return** **true**; 29. } 30. } 31. } |

* + 1. **Result**: Accurate detection of wall collisions based on player movement direction.
  1. **Player-Enemy Collision**:
     1. **Objective**: Ensure player-enemy collisions are detected and handled.
     2. **Code**:

|  |
| --- |
| 1. **function** checkEnemyCollision() { 2. **if** (invulnerable) **return**; 3. **const** playerRect = player.getBoundingClientRect(); 4. **let** enemies = document.querySelectorAll('.enemy'); 5. **for** (**let** enemy **of** enemies) { 6. **const** enemyRect = enemy.getBoundingClientRect(); 7. **if** (playerRect.left < enemyRect.right && playerRect.right > enemyRect.left && 8. playerRect.top < enemyRect.bottom && playerRect.bottom > enemyRect.top) { 9. player.classList.add('hit'); 10. setTimeout(() => player.classList.remove('hit'), 1500); 11. lives--; 12. updateLivesDisplay(); 13. makePlayerInvulnerable(); 14. **if** (lives === 0) { 15. gameOver(); 16. player.classList.remove('hit'); 17. player.classList.add('dead'); 18. } 19. **break**; 20. } 21. } 22. } |

* + 1. **Result**: Correctly identifies collisions and updates game state (e.g., lives, game over).

1. **Level Transition and Game Over**
   1. **Level Complete**:
      1. **Objective**: Ensure level completion logic works correctly.
      2. **Code**:

|  |
| --- |
| 1. **function** levelComplete() { 2. playerCanMove = **false**; 3. showNextLevelScreen(); 4. playerMouth.style.display = 'none'; 5. } |

* + 1. **Result**: Correctly transitions to the next level.
  1. **Game Over**:
     1. **Objective**: Ensure game over logic works correctly.
     2. **Code**:

|  |
| --- |
| 1. **function** gameOver() { 2. playerCanMove = **false**; 3. player.classList.add('dead'); 4. playerMouth.style.display = 'none'; 5. showGameOverScreen(); 6. } |

* + 1. **Result**: Correctly displays game over screen and stops player movement.

### **Conclusion**

**Black Box Testing** revealed that the game functions as expected, except for the maze generation for levels 6 and above, where points can become enclosed by obstacles.

**White Box Testing** focused on ensuring the accuracy and functionality of the code, including initialization, maze generation, player movement, and collision detection. Here are the findings:

* **Initialization and Variables**: No bugs were found.
* **Maze Generation**:
  + From level 6 onwards, the maze generation algorithm results in mazes that are less solvable, with points enclosed by obstacles.
  + The generateMazeInternal function should be revisited to ensure proper placement of walls, enemies, and points.
  + The isSolvable and ensurePointsAccessible functions may not be checking all edge cases, leading to unsolvable mazes occasionally.
* **Player Movement and Collision Detection**:
  + No specific bugs were identified, but the checkWallCollisions function might need further stress testing at different game speeds to ensure accuracy.
  + Random enemy movement and collision logic work correctly, but enhancements could improve gameplay smoothness.
* **Level Complete and Game Over Logic**: No bugs were found.
* **Checklist Updates**: The checklist items do not update correctly after completing a level. They should reset and update at the end of each level.

**Identified Bugs/Weaknesses**:

* **Maze Generation**:
  + For levels 6 and above, mazes can become unsolvable with points enclosed by obstacles.
  + The maze generation algorithm needs refinement to prevent points from being enclosed.
  + The functions generateMazeInternal, isSolvable, and ensurePointsAccessible need improvements to handle edge cases more effectively.
* **Checklist Updates**: Checklist items do not update correctly after level completion, requiring a fix to reset and update the checklist at the end of each level.

To address these issues, especially with maze generation for higher levels, the algorithm needs refinement to ensure solvability. Further testing should be conducted after updates to confirm the effectiveness of the fixes. Overall, the game logic and functionality work well, providing a solid foundation for further development and improvement.

# **4.Evaluation**

Black Box Testing revealed that the game functions as expected, except for the maze generation for levels 6 and above, where points can become enclosed by obstacles.

White Box Testing focused on ensuring the accuracy and functionality of the code, including initialization, maze generation, player movement, and collision detection. Here are the findings:

**Initialization and Variables:**

* No bugs were found.

**Maze Generation:**

* From level 6 onwards, the maze generation algorithm results in mazes that are less solvable, with points enclosed by obstacles.
* The generateMazeInternal function should be revisited to ensure proper placement of walls, enemies, and points.
* The isSolvable and ensurePointsAccessible functions may not be checking all edge cases, leading to unsolvable mazes occasionally.

**Player Movement and Collision Detection:**

* No specific bugs were identified, but the checkWallCollisions function might need further stress testing at different game speeds to ensure accuracy.
* Random enemy movement and collision logic work correctly, but enhancements could improve gameplay smoothness.

**Level Complete and Game Over Logic:**

* No bugs were found.

**Checklist Updates:**

* The checklist items do not update correctly after completing a level. They should reset and update at the end of each level.

To address these issues, especially with maze generation for higher levels, the algorithm needs refinement to ensure solvability. Further testing should be conducted after updates to confirm the effectiveness of the fixes. Overall, the game logic and functionality work well, providing a solid foundation for further development and improvement.

### **Known Bugs/Weaknesses in the Game**

1. **Maze Generation for Levels 6+:**
   1. Points can become enclosed by obstacles, making the maze unsolvable.
2. **Checklist Updates:**
   1. Checklist items do not reset and update correctly after completing a level.
3. **Player Movement and Collision Detection:**
   1. Potential need for further stress testing of the checkWallCollisions function under different game speeds.

### **What Works Well?**

* Basic gameplay mechanics function as expected.
* Player movement and collision detection with walls and enemies work correctly.
* Score updates and level progression are functioning as intended.
* Random enemy movement logic operates as expected.

### **Potential Improvements**

1. **Refine Maze Generation Algorithm:**
   1. Ensure points are not enclosed by obstacles for higher levels.
   2. Improve the generateMazeInternal function to place walls, enemies, and points correctly.
2. **Enhance Checklist Functionality:**
   1. Update the checklist items correctly at the end of each level.
3. **Optimize Collision Detection:**
   1. Further test the checkWallCollisions function for accuracy at different game speeds.
4. **Improve Enemy Movement:**
   1. Enhance enemy movement logic for smoother gameplay.

### **Additional Features**

If more time were available, the following features could be added:

1. **Power-ups:**
   1. Temporary invincibility, ability to pass through walls, enemy freezing.
2. **Scoring System:**
   1. High score leaderboard, bonus points for completing levels quickly, combo system for collecting multiple points in succession.
3. **Lives and Checkpoints:**
   1. Lives system with extra lives as rewards, checkpoints within levels.
4. **Visual Enhancements:**
   1. Animations for character movements and interactions, dynamic lighting effects, particle effects for collectibles and power-ups.
5. **Sound and Music:**
   1. Sound effects for character actions and events, background music that changes based on the level or situation.

### Future Considerations

**How easy would it be to extend the game to add more functionality?**

* The game is designed with modular functions, making it relatively easy to extend and add more functionality. The leveling system and game logic are already structured to accommodate new features with some modifications.

### **Extensibility**

**If you had to build a similar game in the future, what would you do differently and why?**

1. **Use of Advanced Pathfinding Algorithms:**
   1. Implement advanced pathfinding algorithms like A\* with heuristic optimizations for better enemy movement and maze generation.
2. **Modular Design:**
   1. Ensure the game is designed with modular components from the beginning, allowing for easier updates and addition of new features.
3. **Enhanced Testing:**
   1. Incorporate both automated and manual testing extensively to cover more edge cases, especially for maze generation and collision detection.
4. **User Feedback:**
   1. Implement feedback mechanisms to collect user insights and improve gameplay experience iteratively.

# **5. Video Demonstration**

# **6. Conclusion**

### **Conclusion**

This project aimed to develop a web-based maze game where players navigate through randomly generated mazes, collect points, and avoid enemies. Inspired by classic arcade games, the project successfully implemented many core functionalities, providing a solid foundation for a fun and engaging gameplay experience.

#### **Strengths**

1. **Core Functionality:**
   1. The game initializes correctly, allowing players to start the game, move through the maze, collect points, and encounter enemies. The collision detection for both walls and enemies works as intended.
2. **Player Interaction:**
   1. Player movement, enemy movement, and interaction logic are well-implemented. The game accurately updates scores and progresses through levels, enhancing the player's experience.
3. **Leaderboard and Lives System:**
   1. The game includes a functional leaderboard, recording and displaying scores accurately. The lives system works correctly, providing players with multiple attempts before the game ends.
4. **Random Maze Generation:**
   1. The game successfully generates random mazes for each level, adding variety and replayability.

#### **Weaknesses**

1. **Maze Generation for Higher Levels:**
   1. From level 6 onwards, the maze generation algorithm has issues, leading to unsolvable mazes where points are enclosed by obstacles. This significantly affects the playability and fairness of the game at higher levels.
2. **Checklist Updates:**
   1. The checklist items do not update correctly after completing a level, causing confusion for players and impacting the game's usability.
3. **Collision Detection Stress Testing:**
   1. While collision detection generally works well, further stress testing under different game speeds is needed to ensure robustness and accuracy.

#### **Areas for Improvement**

1. **Refine Maze Generation Algorithm:**
   1. Ensure that points are not enclosed by obstacles in higher levels. Improve the generateMazeInternal, isSolvable, and ensurePointsAccessible functions to handle edge cases more effectively.
2. **Enhance Checklist Functionality:**
   1. Ensure that checklist items reset and update correctly at the end of each level to provide clear objectives for players.
3. **Optimize Collision Detection:**
   1. Conduct further stress testing of the checkWallCollisions function under various conditions to enhance accuracy.
4. **Improve Enemy Movement:**
   1. Refine the enemy movement logic to ensure smoother and more predictable behavior, improving gameplay fluidity.

#### **Additional Features and Future Considerations**

Given more time, several additional features could enhance the game:

* **Power-ups:** Implement temporary invincibility, ability to pass through walls, and enemy freezing to add variety and strategic depth.
* **Enhanced Scoring System:** Introduce a high score leaderboard, bonus points for quick level completions, and a combo system for collecting multiple points in succession.
* **Visual and Audio Enhancements:** Add animations, dynamic lighting effects, and sound effects to enrich the gameplay experience.

For future projects, adopting a modular design approach from the outset would facilitate easier updates and additions. Incorporating advanced pathfinding algorithms like A\* could improve enemy movement and maze generation. Additionally, implementing user feedback mechanisms would help iteratively improve the game based on player experiences.

### **Overall Assessment**

The assignment was well-executed in many respects, successfully implementing core functionalities and providing a solid foundation for further development. However, there are notable areas for improvement, particularly in maze generation for higher levels and checklist functionality. Addressing these issues and incorporating suggested improvements would significantly enhance the game, providing a more engaging, fair, and enjoyable experience for players. Overall, the project demonstrates strong potential and a good understanding of game development principles.

# **7. References**

1. JIT.codes. (n.d.). *JIT.codes — Convert text to code with AI*. [online] Available at: https://jit.codes/ [Accessed 9 May 2024].  
  
2. Stromberg, D. (n.d.). *Tutorial - Random Maze Generation Algorithm in Javascript*. [online] dstromberg. Available at: https://dstromberg.com/2013/07/tutorial-random-maze-generation-algorithm-in-javascript/ [Accessed 9 May 2024].  
  
3. openai.com. (n.d.). *ChatGPT*. [online] Available at: https://openai.com/chatgpt/.  
  
4. Adminko (2024). *LLama 3 on Groq*. [online] Llama 2. Available at: https://llama-2.ai/llama-3-on-groq/ [Accessed 9 May 2024].  
  
5. Claude.ai. (2023). *Claude*. [online] Available at: https://claude.ai/.  
  
6. servetgulnaroglu (2024). *servetgulnaroglu/pacman-js*. [online] GitHub. Available at: https://github.com/servetgulnaroglu/pacman-js [Accessed 9 May 2024].

7. Canva (2024). *Canva*. [online] Canva. Available at: https://www.canva.com/.

8. www.the-art-of-web.com. (n.d.). *Playable Maze Game Generator < JavaScript | The Art of Web*. [online] Available at: https://www.the-art-of-web.com/javascript/playable-maze-generator/ [Accessed 9 May 2024].

9. Anthony (n.d.). *JavaScript Maze Game*. [online] CodePen. Available at: https://codepen.io/anthropoco/pen/ZRmYvZ [Accessed 17 Apr. 2024].

10. biophorm.com. (n.d.). Biophorm - Maze Generator and Solver by Robert Hankin. [online] Available at: https://biophorm.com/ [Accessed 9 May 2024].

11. mazegenerator.net. (n.d.). Maze Generator. [online] Available at: https://mazegenerator.net/.