**Check List**

|  |  |
| --- | --- |
| 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. |  |

**JavaScript Maze Game Development: A Detailed Review**  
**1. Introduction**  
This report provides a comprehensive overview of the development process for a JavaScript-based maze game, covering its design, implementation, testing, and evaluation phases. The game incorporates dynamic maze generation, player movement, enemy AI, scoring, and a leaderboard system. The objective of this project was to create an engaging and challenging gaming experience while applying advanced principles of game design and programming. It also elaborates further on technical issues faced, remedies instituted, and where future enhancements can be made.

**1.1 Objectives**  
To implement a playable, fun maze game in JavaScript with HTML and CSS, representing both elementary and sophisticated ideas in creating web-based games.  
Key features that implement include player movement, enemy artificial intelligence, scoring, levels, and leadership that introduces complexity and replay value.  
• Extensibility: It should be possible to extend the game afterward without having to rewrite much or any of the existing source code.

**2. Code Development and Refinements**

**2.1 Plain Implementation**  
The game was developed using a 2D array to represent the maze, where different numbers signify walls, the player, enemies, and collectible points. JavaScript was employed to dynamically generate the maze on the HTML document. The structure was designed to allow easy modification and extension, supporting both the game’s scalability and adaptability to future changes.

**2.1.1 Player Movement**  
Player movement was managed through event listeners attached to keyboard inputs, with Boolean flags indicating movement directions (up, down, left, right). A timer function was used to update the player’s position based on active key presses, ensuring smooth and responsive control. CSS properties were manipulated to visually move the player’s avatar on the screen, synchronizing gameplay logic with visual feedback.

**2.1.1.1 Technical Implementation**  
• Event Listeners: JavaScript’s addEventListener was utilized to detect keydown and keyup events, allowing for continuous movement when keys are held down.  
• Movement Logic: A combination of setInterval and Boolean flags was used to handle movement updates, ensuring that multiple key presses could be detected and processed simultaneously.  
• CSS Integration: The player’s position on the screen was dynamically updated by modifying the CSS transform property, allowing for smooth transitions and real-time feedback.

**2.2 Leaderboard and Level System**

  
A leaderboard was implemented to track and display high scores, encouraging competition and replayability. The game also features a level system, with each level presenting a new and increasingly challenging maze layout. These layouts are stored in a mazeLevels object, referenced dynamically during level transitions.

**2.2.1 Implement Leaderboard**

  
• Data Storage: Scores were stored using the browser’s localStorage API, allowing data persistence across sessions without requiring server-side infrastructure.  
• Display Logic: JavaScript was used to sort and display scores in descending order, ensuring the leaderboard was always updated and accurate.

**2.2.2 Graduated System Mechanisms**  
• Dynamic Maze Generation: Each level is generated by selecting a new layout from the mazeLevels object. This selection is based on the player’s current level, ensuring increased difficulty and variety.  
• Transition Logic: Level transitions were managed by a dedicated function that reset the player’s position, updated the maze layout, and increased the difficulty settings (e.g., faster enemy movement, additional obstacles).

**2.3 Additional features**  
To enhance the user experience, several additional UI elements were integrated, including a start screen, control buttons, and a score display. These elements were styled using CSS and manipulated with JavaScript to provide a clear and accessible interface, ensuring a smooth and intuitive user experience.

**3. Testing and Debugging**

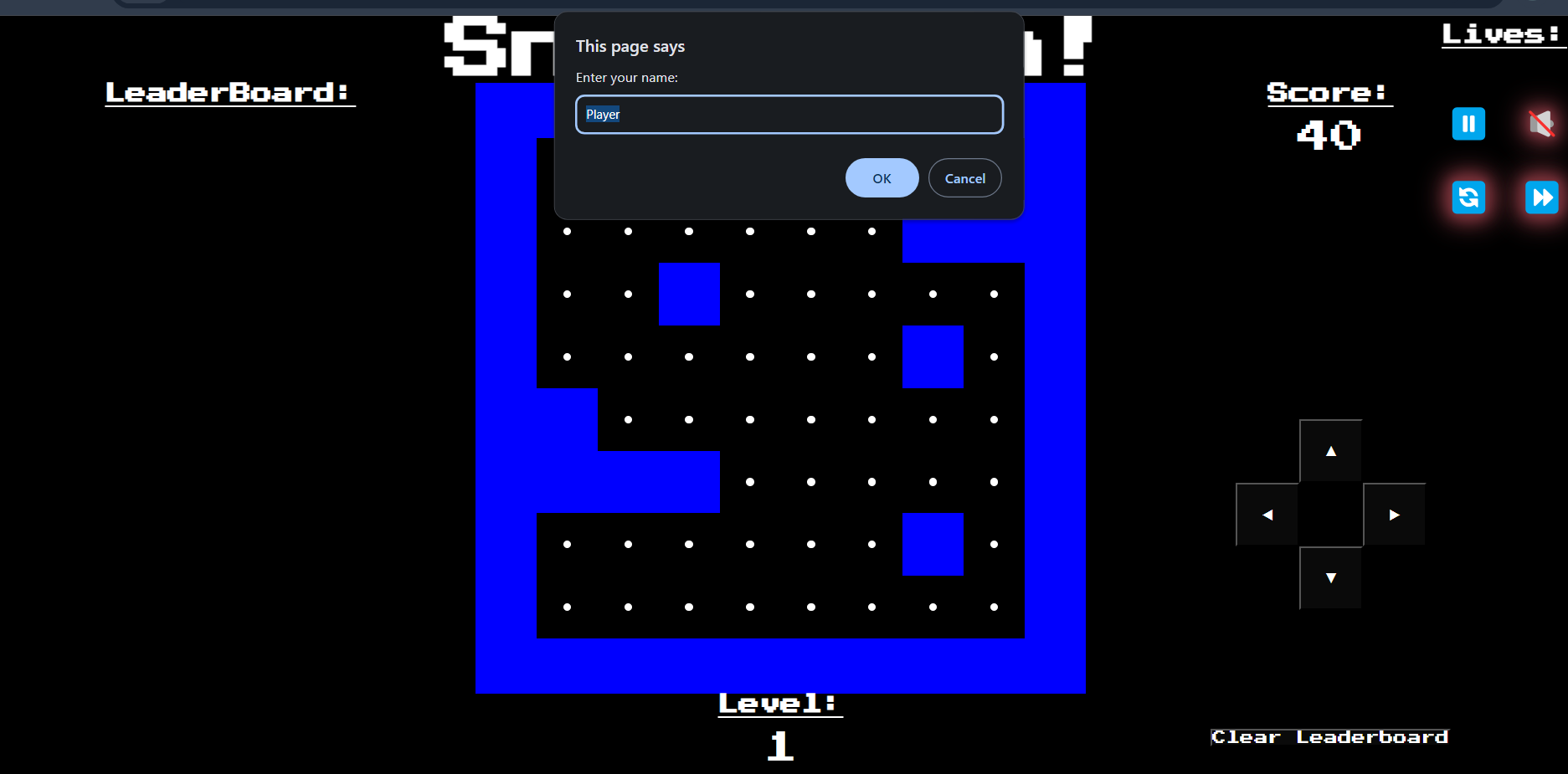
**3.1 Testing Techniques**

  
Testing was conducted iteratively, with each new feature undergoing rigorous testing to ensure both functionality and stability. This approach allowed for early detection and resolution of issues, minimizing the accumulation of bugs and enhancing the overall quality of the game.

**3.1.1 Player Movement Test**  
Player movement was tested by simulating various key press combinations and observing the character’s movement on the screen. Special attention was given to collision detection with walls and enemies to ensure that all game mechanics functioned correctly.

**3.1.1.1. Test tools and techniques**  
• Manual Testing: Extensive manual testing was conducted to ensure that all edge cases, such as simultaneous key presses and rapid direction changes, were handled correctly.  
• Automated Testing: Simple automated tests were written using JavaScript to simulate player movement and collisions under different conditions, verifying that game logic was robust and error-free.

**3.1.2 Scoreboard functionality**



The leaderboard functionality was tested by playing through the game multiple times with different scores. The tests ensured that scores were correctly saved, retrieved, and displayed, and that the leaderboard updated dynamically.

**3.1.3 Level Progression**  
Level progression was thoroughly tested by playing through multiple levels, ensuring that maze layouts changed appropriately and that the game’s difficulty increased with each level.

**3.2 Debugging**  
During development, a lot of bugs were fixed:  
• Boundary Issues: Initially, the player could move outside the maze boundaries. This was corrected by adding boundary checks to prevent the player from moving beyond the maze limits.  
• Enemy AI Issues: Enemies were able to pass through walls due to a flaw in the pathfinding algorithm. This was fixed by refining the enemy movement logic to consider wall positions.  
• Collision Detection Problems: There were issues with collision detection between the player and enemies, sometimes allowing the player to pass through enemies without losing a life. This was resolved by enhancing the collision detection algorithm to accurately detect overlaps between the player and enemies.

**4. Appraisal**

**4.1 Product Acceptance Testing**  
The final product was evaluated based on functionality, usability, and overall game experience. This evaluation aimed to identify both the strengths and weaknesses of the current implementation and provide a foundation for future improvements.

**4.1.1 Issues End**  
• Collision Detection: While improved, the collision detection still has some minor issues, particularly when the player moves quickly or when multiple collisions occur in rapid succession.  
• Responsive Design: The game is not fully optimized for mobile devices, limiting its accessibility to desktop users.  
• Enemy AI: The enemy AI is relatively simple and predictable, reducing the game’s challenge and replay value over time.

**4.1.2 Strengths**  
• Player Movement: The controls are smooth and responsive, providing an enjoyable and immersive gaming experience.  
The dynamic maze generation system ensures that every level is a new challenge, increasing the replay value.  
• Leaderboard Functionality: The leaderboard effectively tracks high scores and encourages competition among players.

**4.2 Recommendations for Improvements**  
• Responsive Design: Improving the game’s responsive design would make it accessible on a wider range of devices, particularly mobile phones and tablets.  
Additionally, enhancing enemy artificial intelligence would boost the level of difficulty by a significant number, hence offering the player more engaging encounters.  
• More features – Power-ups and special items with better audio controls make the playing experience richer.  
**5. Future Work**

**5.1 Potential Upgrades**  
• Multiplayer Mode: Adding a multiplayer option would significantly increase the game’s replayability and appeal, allowing players to compete or cooperate in real-time.  
• Additional Levels and Themes: Include different themes and many more levels beyond 40 that would make the player passionate and engaged.  
This story mode would add depth and context that enhances the overall immersive and engaging qualities in general to the game.

**5.2 Extensibility**  
The game’s modular structure allows for easy extension, with new features being added without major changes to the existing codebase. For instance, adding new levels only requires updating the mazeLevels object, while new enemies or power-ups can be introduced through additional classes or functions.

**Implementation of Button Using HTML**  
  
Due to problems experienced in JavaScript code implementation that include the overlapping of elements and increased complexity, I chose to use HTML to implement the following reasons button:  
  
Avoiding Overlap Issues: The initial JavaScript implementation caused overlapping of buttons, which disrupted the user interface. By using HTML, I was able to position the buttons precisely, ensuring they were displayed correctly without any overlap.  
  
Simplifying Button Layout: HTML provided a straightforward approach to creating and positioning buttons, reducing the complexity introduced by the JavaScript code. This allowed for a clean and organized layout.  
  
Better Reliability: Buttons were implemented directly in HTML, which resulted in their working as expected on most devices and browsers; it was not sporadic behavior from JavaScript errors. Enhanced Control Over Styling: Using HTML allowed for better control over the styling and appearance of the buttons, ensuring they met the design requirements without being affected by JavaScript-induced complexities.

**6. Conclusion**  
The development of this JavaScript maze game was both challenging and rewarding. The final product successfully met the initial objectives, demonstrating competency in JavaScript, HTML, and CSS. The game is playable and enjoyable, with significant potential for further development and enhancement. This project has provided valuable insights into game development and highlighted areas for improvement in future projects.

**7. References**

* W3Schools: JavaScript tutorials and references.
* TutorialsPoint: Tutorials regarding implementation of basic game mechanics and algorithms.
* GeeksforGeeks: Articles on data structures and algorithms, particularly for pathfinding.
* Stack Overflow: Crowd-sourced solutions to specific coding challenges