Research Process Document: Maze Navigation Game Development

This document outlines the problem-solving process followed for developing and refining the **Maze Navigation Game** using **Three.js**, including the tasks undertaken, methodologies employed, and the decision-making involved.

Project Overview:

The objective was to develop a **Maze Navigation Game** that allows a player (represented by a ball) to navigate through a dynamically generated maze. The game needed to include the following key features:

1. Dynamic maze generation using algorithms.
2. Player movement and controls using **WASD** keys.
3. Collision detection to prevent the player from moving through walls.
4. A top-down view to display the entire maze.

Problem-Solving Process:

1. Dynamic Maze Generation:

Objective: Programmatically generate a maze structure, with walls and paths, using an algorithm.

- **Methodology**: I researched maze generation algorithms and settled on the **Recursive Backtracking Algorithm** because it is widely used for generating random mazes with a single solution. This algorithm generates a perfect maze by creating paths in a 2D grid.
- **Implementation**: I used a 2D array to represent the maze, with 1 representing walls and 0 representing paths. The recursive backtracking algorithm randomly selects directions and recursively carves out paths.

References:

- Maze Generation Algorithm (https://www.geeksforgeeks.org/backtracking-algorithms/)
- Three.js documentation for BoxGeometry (BoxGeometry – three.js docs (threejs.org))

2. Player Movement and Controls

Objective: Allow the player to move around the maze using the **WASD** keys.

- **Methodology**: I implemented player controls using **JavaScript event listeners** to detect key presses and update the player's position accordingly.
- **Implementation**: The player (represented by a sphere) moves based on key inputs. I used a vector to calculate the movement in the X and Z axes.

3. Collision Detection

Objective: Ensure that the player cannot move through the walls of the maze.

- **Methodology**: I used **bounding boxes** (THREE.Box3) to detect collisions between the player and the walls. Each wall is represented by a THREE.BoxGeometry, and a bounding box is created for each wall and the player.
- **Implementation**: Before moving the player, the new position is calculated and checked for potential collisions with the walls. If a collision is detected, the movement is prevented.

  References:

- Three.js Box3 Collision Detection(https://discourse.threejs.org/t/collision-response-intersection-via-box3-intersectsbox/38272)

4. Top-Down Camera View

Objective: Provide a top-down view of the entire maze, as requested by my instructor.

- **Methodology**: Instead of the default third-person perspective, I positioned the camera above the maze and directed it to look straight down. This gives a full view of the maze layout.
- **Implementation**: I set the camera's position to a height (MAZE_SIZE * CELL_SIZE) that allows the entire maze to be visible and used camera.lookAt() to ensure it is pointed at the center of the maze.

Reference Game:

- http://joppiesaus.github.io/mazegame/
- https://github.com/joppiesaus/mazegame/blob/master/js/main.js
- https://aryanab.medium.com/maze-generation-recursive-backtracking-5981bc5cc766

Conclusion

By researching algorithms, camera manipulation, and collision detection, I created a functioning game with a dynamic maze, responsive player controls, and a top-down view. This organized approach allowed for iterative improvements and a focus on user experience, demonstrating my ability to learn and solve complex problems in a structured way.