# **🧠 Task: "AI Maze Solver & Visualizer"**

⏳ Estimated Time: 3 Hours  
🎯 Goal: Create a React-based Maze Solver where a user generates a random maze, selects a start & end point, and runs an algorithm (BFS/DFS/A)\* to find the shortest path.

## **📌 Task Requirements**

✅ Generate a random grid-based maze (Walls & Open Paths)  
✅ Allow users to select Start & End points  
✅ Implement a maze-solving algorithm (BFS, DFS, or A)\*  
✅ Visualize the algorithm's execution step-by-step  
✅ Show shortest path in the UI after solving  
✅ Ensure performance optimizations for large grids

## **🔹 How It Works (User Flow)**

1️⃣ User clicks "Generate Maze" → Creates a 10x10 grid with walls & open paths randomly.  
2️⃣ User selects a Start & End point by clicking on two grid cells.  
3️⃣ User clicks "Solve Maze" → The app runs BFS, DFS, or A\* to find the shortest path.  
4️⃣ Algorithm runs step by step (visualized with animations).  
5️⃣ Final shortest path is highlighted in a different color.  
6️⃣ User can regenerate the maze and try again.

## **🔹 Implementation Details**

### **1️⃣ Grid Generation**

* Create a 10x10 (or larger) grid with walls (#) and open paths (.).
* Randomly place walls while ensuring there’s at least one valid path.
* Example Maze Representation:

. . # . . . # . . .

. # # . # . # . # .

. . . . # . . . # .

. # # # . # # . . .

. . . . . . . # # .

### **2️⃣ User Input (Start & End Point Selection)**

* Click on a grid cell to mark Start (🟢) and End (🔴) points.
* Ensure Start & End are not walls.

### 3️⃣ Maze Solving Algorithms (BFS, DFS, or A)\*

#### **🔹 Breadth-First Search (BFS) (Recommended)**

* Finds the shortest path in an unweighted grid.
* Algorithm:
  1. Start at (startX, startY), add it to a queue.
  2. Visit adjacent open cells (.), mark them as visited.
  3. If End (🔴) is reached, stop.
  4. Trace back the shortest path.

#### **🔹 Depth-First Search (DFS) (Alternative)**

* Not guaranteed to find the shortest path but works.

### **4️⃣ Visualization of Algorithm Execution**

* Step-by-step rendering: Show nodes being explored in real-time.
* Color Coding:
  + Unvisited nodes → Light grey
  + Visited nodes → Blue
  + Walls → Black
  + Final Path → Green