

**T.C.**

**MARMARA UNIVERSITY**

**FACULTY of ENGINEERING**

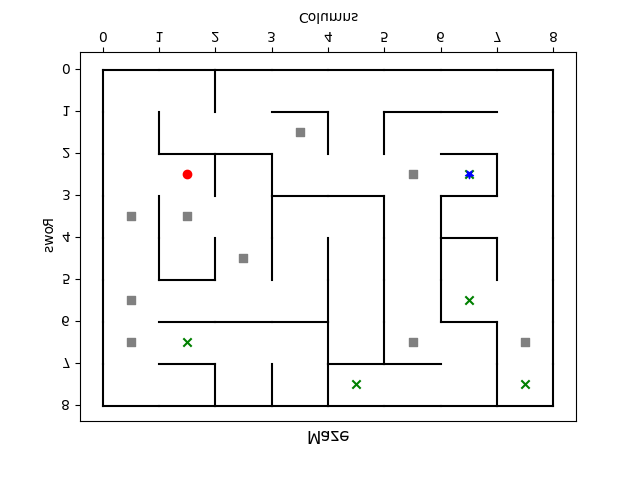
**COMPUTER ENGINEERING DEPARTMENT**

**AI**

**ASSIGNMENT 2**

**Ömercan Sabun-150119555**

**Senanur Yılmaz-150119801**

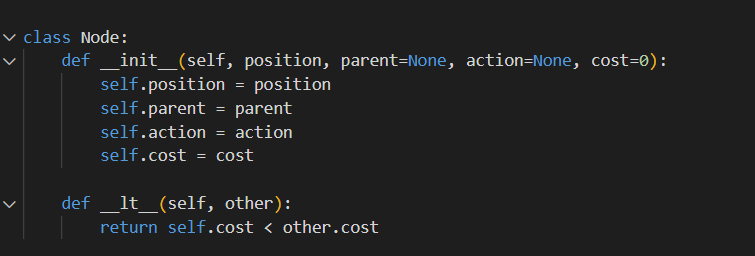


There is the maze we created above. The green x sign indicates Goals. Gray squares indicate traps. The red circles are the starting point.

**Design Document:**

**Classes: Node, Maze, MazeSolver,**

1. **Node:**



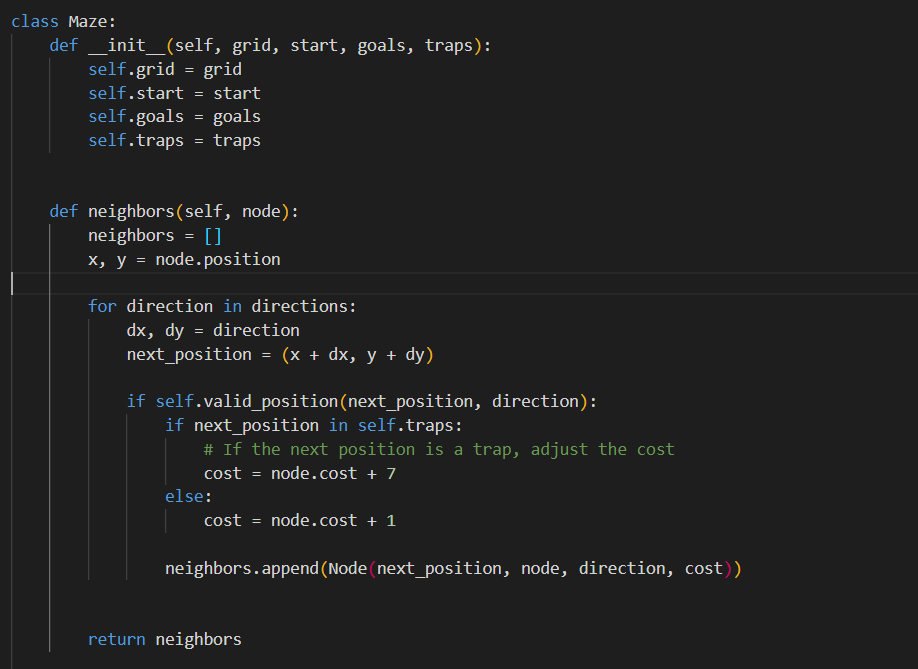
Fields:

* + - **position**: Tuple representing the position of the node in the maze.
    - **parent**: Reference to the parent node.
    - **action**: Action taken to reach this node from the parent node.
    - **cost**: Cost of reaching this node from the start node.

Methods:

* + - **\_\_init\_\_(self, position, parent=None, action=None, cost=0)**: Initializes a new Node instance.
    - **\_\_lt\_\_(self, other)**: Implements less than comparison for nodes based on their cost.

1. **Maze:**

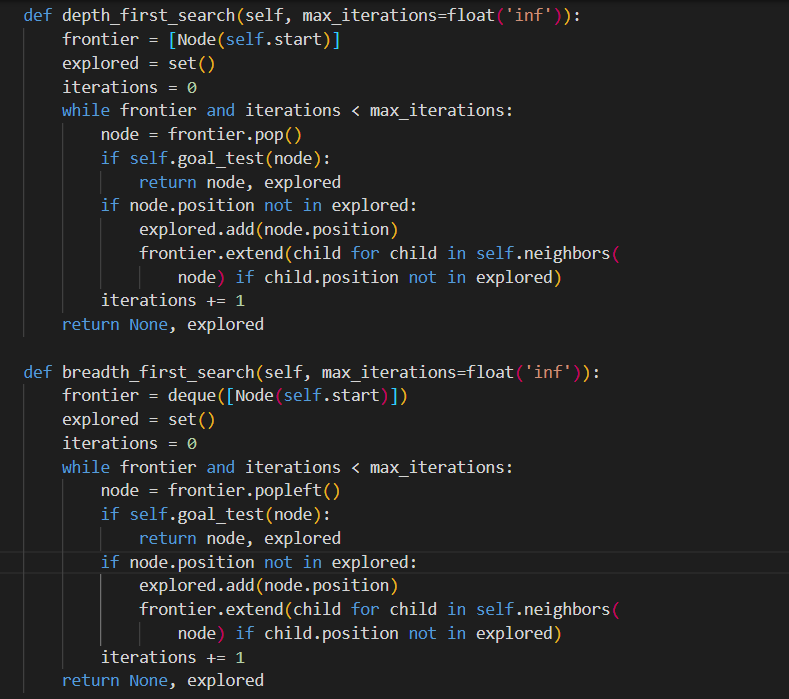


Fields:

* + - **grid**: 2D list representing the maze layout.
    - **start**: Tuple representing the starting position in the maze.
    - **goals**: List of tuples representing the goal positions.
    - **traps**: List of tuples representing the trap positions.

Methods:

* + - **\_\_init\_\_(self, grid, start, goals, traps)**: Initializes a new Maze instance.
    - **neighbors(self, node)**: Returns a list of neighboring nodes for a given node.
    - **valid\_position(self, position, direction)**: Checks if a given position in a certain direction is valid.
    - **get\_direction\_key(self, dx, dy)**: Returns the key for a given direction.
    - **goal\_test(self, node)**: Checks if a node's position is one of the goal positions.



* + - Search methods: **solve**, **depth\_first\_search**, **breadth\_first\_search**, **iterative\_deepening\_search**, **uniform\_cost\_search**, **greedy\_best\_first\_search**, **a\_star\_search**.
    - **heuristic(self, node)**: Calculates the heuristic value for a given node.
    - Helper methods: **simulate\_agent**, **print\_maze\_with\_agent**, **print\_maze**, **draw\_maze\_with\_agent**.

1. **MazeSolver:**

Fields:

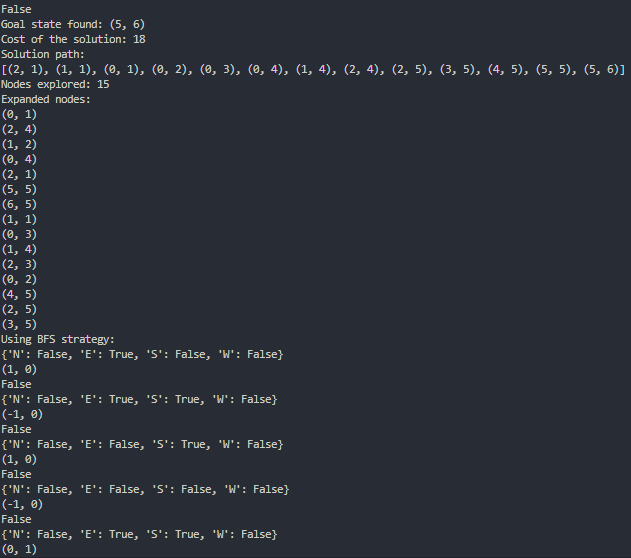
* + - None (This class may not be explicitly defined in the provided code).

Methods:

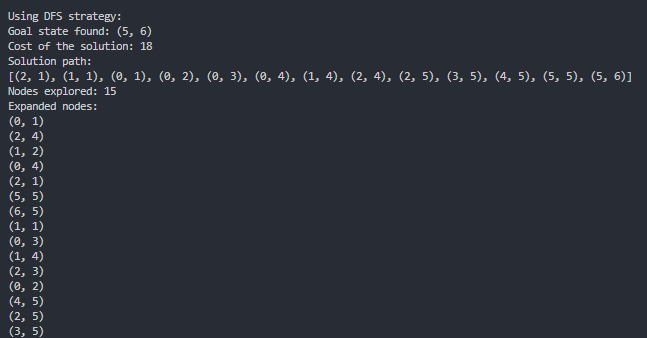
* + - **solve\_maze(maze, strategy)**: Prints the solution for a given maze and search strategy.
    - **print\_path(solution)**: Prints the path from the start to the goal.
    - **print\_explored\_nodes(explored)**: Prints the nodes explored during the search.
    - **read\_maze(filename)**: Reads maze details from a file and returns a Maze instance.
    - **draw\_maze\_with\_agent(maze, agent\_position)**: Draws the maze with the agent's position.

**Output Example:**

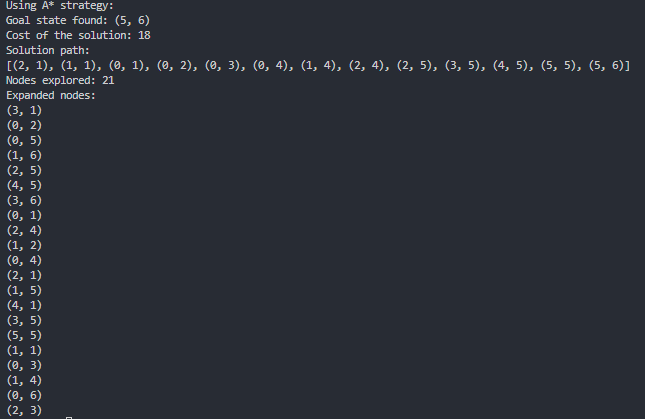
**BFS**



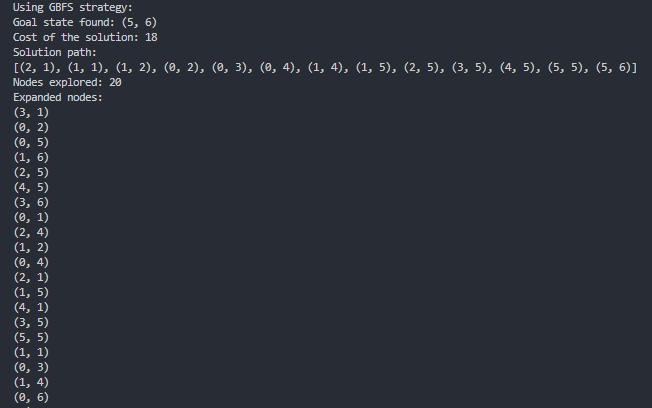
**DFS**



**A\***



**GBFS**



**UCS**

