Code: solve_maze.py

Explanation:
This algorithm solves a maze using recursion and is divided into three steps.
It outputs one valid path from the start to the goal as a log.

1. Initialize the data.
seen keeps track of the visited coordinates, and path keeps track of the current path.

2. Search using depth-first search.
The algorithm checks the current position, records it, checks whether it is the goal,
and then recursively explores the four directions.
If all directions fail, it backtracks to the previous position.

3. Print the result.
Finally, the solution path is printed.

The execution log is shown on the next page.

```python
        def dfs(r, c):
            if dfs(r, c-1):
                return True
            if dfs(r-1, c):
                return True

            path.pop()
            return False    Generate code: ⌥⇧Enter

        return path if dfs(0, 0) else None

    ans_path = solve(maze)

    # --- Print ---
    Zencoder
    def print_maze(maze, path):
        print("--- Print Path(*): ---")
        path = set(path) if path else set()
        for r in range(len(maze)):
            for c in range(len(maze[0])):
                if (r, c) in path:
                    print("*", end=" ")
                elif maze[r][c] == 1:
                    print("#", end=" ")
                else:
                    print(".", end=" ")
            print()
        print("----------------------")
    print_maze(maze, ans_path)
```

問題　出力　デバッグ コンソール　ターミナル　ポート

```
seigo@takadaseigonoMacBook-Pro abc443 % python solve_maze.py
--- Print Path(*): ---
* * # . . . # . . .
# * # . # . # . # .
# * * * # . . . # .
# # # * # # # . # .
. . . * * * # . . .
. # # # # * # # # .
. # . . . * * * # .
. # . # # # # * # .
. . . # . . . * * *
# # . # . # # # # .
----------------------
seigo@takadaseigonoMacBook-Pro abc443 %
```

Code

solve_maze.py

Log