solve1.py

```python
# solve1.py
# define the maze as an array
# 2 is the start
# 3 is the end
# 1 is a path
# 0 is blocked

#10x10 list of lists
maze = [
    [2, 0, 3, 1, 1, 0, 1, 1, 1, 1],
    [1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 1, 0, 0, 1, 1, 1],
    [1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
    [1, 0, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1, 0, 1],
    [1, 1, 1, 1, 0, 1, 0, 1, 0, 1],
    [1, 0, 0, 0, 0, 1, 0, 1, 1, 1],
    [1, 0, 1, 0, 0, 1, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
]


def solve(maze):

    # get the number of rows and columns
    rows = len(maze)
    cols = len(maze[0]) if rows > 0 else 0

    # find the start position
    start = None
    for i in range(len(maze)):
        for j in range(len(maze[i])):
            if maze[i][j] == 2:
                start = (i, j) # this is called a tuple - because of the brackets
                break
        # we found start so stop looking
        if start:
            break

    # find the end position
    end = None
    for i in range(len(maze)):
        for j in range(len(maze[i])):
            if maze[i][j] == 3:
                end = (i, j) # this is called a tuple - because of the brackets
                break
        # we found end so stop looking
        if end:
            break

    if not start or not end:
```

```
52          return None
53
54      # initialize the path
55      path = []
56
57      #create and set visited to false
58      visited = [[False]*cols for _ in range(rows)]
59
60      def check(r, c):
61          if not (0 <= r < rows and 0 <= c < cols):
62              return False
63          if maze[r][c] == 0 or visited[r][c]:
64              return False
65
66          visited[r][c] = True
67          #print(f"Visiting: ({r}, {c})")
68
69          path.append((r, c))
70
71          if (r, c) == end:
72              return True
73
74          # Check all four directions recursively
75          if (check(r+1, c) or check(r-1, c) or check(r, c+1) or check(r, c-1)):
76              return True
77
78          # If none of the directions lead to a solution, backtrack
79          path.pop()
80          return False
81
82
83
84      if check(*start): # unpacking the tuple *start
85          return path
86      else:
87          return None
88
89  # Test
90  path = solve(maze)
91  if path:
92      print("Path found:")
93      for step in path:
94          print(step)
95  else:
96      print("No path found.")
97
98
```