ANSWERS

A2.10 Project: out of the dead end

Recursion and computational thinking · A level · OCR H446 2.1.1, AQA 7517 4.1.1.15, Eduqas A500QS 1.3 · about 30 min

BugBotLab

What this lesson is about

Model a maze, solve it by recursive backtracking, and drive the robot out along the route.

- The model
- Plan before code
- How the search runs

Questions

5 marks in all

1. In the maze project, what is the base case that ends a call of solve with success? [1 mark]

- ☐ **A** The cell is the goal
- ☐ **B** The cell is a block
- ☐ **C** The cell is off the grid
- ☐ **D** All four directions have been tried

> **Answer:** A. Reaching G returns True. Blocked, off-grid and visited cells are base cases that return False.

2. What does the visited set do in the backtracking search? [1 mark]

- ☐ **A** Stops the search going round in circles by never exploring a cell twice
- ☐ **B** Stores the final route
- ☐ **C** Holds the return addresses
- ☐ **D** Makes the path the shortest possible

> **Answer:** A. It records cells already explored, a kind of cache. The route is kept on the path list.

3. What does this program print? [1 mark]

```
MAZE = ["S.#",
        "#.#",
        "..G"]
visited = set()
path = []

def solve(r, c):
    if r < 0 or r > 2 or c < 0 or c > 2:
        return False
    if MAZE[r][c] == "#" or (r, c) in visited:
        return False
    visited.add((r, c))
    path.append((r, c))
    if MAZE[r][c] == "G":
        return True
    for dr, dc in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
        if solve(r + dr, c + dc):
            return True
    path.pop()
    return False

solve(0, 0)
print(path)
```

Answer:

```
[(0, 0), (0, 1), (1, 1), (2, 1), (2, 2)]
```

Trying right, down, left, up: right to (0, 1), down to (1, 1), down to (2, 1), then right to G at (2, 2). No dead end is met.

4. Which details did the maze model leave out, to be handled by the driving code? [1 mark]

Tick every answer that is true.

- ☐ **A** The thickness of the blocks
- ☐ **B** The size of the robot
- ☐ **C** The robot's drift as it drives
- ☐ **D** Which cells are open

Answer: A, B, C. The model keeps only which cells are open; the driving code corrects for reality with position().

5. Why does a backtracking search not always find the shortest route? [1 mark]

- ☐ **A** It returns the first route it finds, which depends on the order the directions are tried
- ☐ **B** It never visits the goal
- ☐ **C** It explores every route before choosing
- ☐ **D** It uses a heuristic

Answer: A. Depth-first backtracking stops at the first success. Breadth-first search finds a shortest route in an unweighted grid.

The task: solve the maze model

Write a recursive function `solve(row, col)` for the maze above, following the five rules, with `MAZE` as the list of five strings, a set `visited` and a list `path`. - `row` and `col` are whole numbers; the grid runs from 0 to 4 in each. - `solve` returns `True` if the goal can be reached from `(row, col)` without revisiting a cell, and `False` otherwise. - A cell that is off the grid, a block or already visited returns `False` **without** printing anything. - An open cell that fails in all four directions prints `back from (<row>, <col>)`, for example `back from (0, 2)`, after taking itself off `path`. - Try the directions in the order up, right, down, left. Call `solve(4, 0)`, then print `path:` followed by every cell on `path`, each written `(row, col)` and separated by single spaces, starting `path: (4, 0) (4, 1)`.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

MAZE = ["...#G",
        ".###.",
        ".#...",
        ".#.#.",
        "S...."]
visited = set()
path = []

def solve(row, col):
    return False

solve(4, 0)
```

The hint students can ask for: Write the checks that end a call straight away first: off the grid, a block, or a cell already visited. Then mark the cell and add it to the path. If it is the goal you are done; otherwise try the four directions in the order given, and stop as soon as one works. Only when all four fail do you take the cell back off the path and report it.

A solution

```
from bugbot import *
connect()

MAZE = ["...#G",
        ".###.",
        ".#...",
        ".#.#.",
        "S...."]
visited = set()
path = []

def solve(row, col):
    if row < 0 or row > 4 or col < 0 or col > 4:
        return False
    if MAZE[row][col] == "#" or (row, col) in visited:
        return False
    visited.add((row, col))
    path.append((row, col))
    if MAZE[row][col] == "G":
        return True
    for d_row, d_col in [(-1, 0), (0, 1), (1, 0), (0, -1)]:
        if solve(row + d_row, col + d_col):
```

```
            return True
    path.pop()
    print(f"back from ({row}, {col})")
    return False

solve(4, 0)
print("path:", " ".join(f"({r}, {c})" for r, c in path))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.