Model Based Agent for Object Collection

Submitted to:	Submitted by:
Amit Kumar Mondal Associate Professor Computer Science & Engineering Discipline Khulna University, Khulna	Tahmid Hasan Tasfi Student ID: 210218 3rd Year, 1st Term Computer Science and Engineering Discipline Khulna University Khulna

Course No: CSE 3202

Course Title: Artificial Intelligence

Date of Submission: October 22, 2024

1. Theoretical Analysis

The provided code implements a simulation where a robot navigates a randomly generated room to collect objects represented by "#" symbols. Below is a detailed theoretical analysis of the key components, design choices, and potential areas for improvement.

1. Structure and Initialization
Initialization:

- `create_room()`: Generates an 8x8 grid filled with random elements (0s for empty spaces, 'H' for hurdles, and '#' for objects). The randomness introduces variability in each run.
- `start()`: Randomly assigns a starting position for the robot within the room boundaries.

2. Room Representation

- The room is represented as a 2D list. The robot's current position is marked with "R", and the path taken is marked with "." in the printed representation. The rest of the cells show their respective contents.

3. Movement Logic

- **Direction Handling**: The robot moves in four possible directions: up, down, left, and right. The code includes logic to prevent the robot from reversing its last move, which simulates more deliberate navigation.
- **Boundary and Hurdle Checks**: The robot checks for boundaries and obstacles ('H') before attempting a move, ensuring that it does not go out of bounds or collide with hurdles.

4. Object Collection

- When the robot encounters an object ("#"), it collects it (removes the object and increments the count). The simulation tracks both the total number of objects collected and the actions taken.

5. Simulation Steps

- The `collect_object()` method governs a single iteration of the robot's operation. It
  manages movement, object collection, and prints the room's state at each step. The method loops until the robot either collects an object, hits a boundary, or becomes blocked by hurdles.

6. Performance Metrics

- After each step, the code calculates and prints performance metrics: the total number of objects collected, the number of moves, and the performance ratio (actions per object collected). This metric provides insight into the robot's efficiency.

7. Main Function and Execution

- The `main()` function orchestrates multiple iterations of the robot's operations. It aggregates results across five runs and computes final performance metrics

Theoretical Considerations

Complexity Analysis

- **Time Complexity**: Each step involves checking adjacent cells and potentially iterating over directions, leading to a complexity of O(1) per move attempt. However, since the robot can move multiple times, the overall complexity can scale with the number of steps taken until a boundary or blockage occurs.

- **Space Complexity**: The primary space usage comes from the room representation (O(n^2) where n is the room size) and the path list, which can grow with the number of moves.

Randomness and Exploration

- The use of random elements in room creation and starting positions introduces uncertainty and variability in the robot's performance. This randomness can be beneficial for testing the robustness of the robot's navigation strategy.

2. Data Structure

The primary data structure used is a 2D list, which represents the room layout. Here's a detailed breakdown of the data structures and their components:

1. 2D List (Room Representation)

- **Structure**:
  - The room is represented as an 8x8 grid using a list of lists in Python. Each inner list corresponds to a row in the grid.
  - The overall structure can be visualized as:

```
room = [
    [0, 0, 'H', '#', 0, 'H', 0, 0],
    [0, '#', 0, 0, 0, 0, '#', 0],
    ['H', 0, 0, 'H', 0, '#', 0, 'H'],
  ...
```

]

- **Cell Values**:
  - ` 0 ` : Represents an empty space where the robot can move.
  - ` # ` : Represents an object that the robot can collect.
  - ` H ` : Represents a hurdle that blocks the robot's movement.

2. Robot Position

- The robot's current position is tracked using two integer variables:
  - `self.x`: Represents the row index of the robot's current position.
  - `self.y`: Represents the column index of the robot's current position.

3. Path Tracking

- List of Tuples: The robot's path is stored in a list called `self.path`, which contains tuples representing the coordinates of the positions the robot has visited.

4. Performance Metrics

- The robot tracks several integer counters to measure performance:
  - `total_moves`: Counts the total number of moves the robot has made.
  - `total_objects_collected`: Counts the total number of objects collected.
  - `total_actions`: Counts the total actions taken by the robot, including collecting objects.

5. Other Variables

- Previous Direction:
  - `prev_direction`: A string that records the last direction the robot moved in, helping to prevent the robot from reversing direction immediately.

- **Implications of the Data Structure Choice**
  - The use of a 2D list is suitable for representing grid-based environments, allowing easy access to neighboring cells for movement and checks for obstacles.
  - Storing the path as a list of tuples facilitates easy tracking and visualization of the robot's movements.
  - The choice of simple integer counters for performance metrics allows for straightforward updates and calculations during the simulation.

This structure effectively supports the simulation's requirements while remaining efficient and easy to manipulate.

3. Algorithm to Function/Method Representation

Initialize room (8x8 grid)

Set robot's starting position

For each step in range(5):

    Initialize total_moves, total_objects_collected

    While True:

        If current position has object:

            Collect object

        Add current position to path

        Print room state

        If at boundary:

            Break

Determine valid move directions

If valid directions:

Choose direction randomly and move

Else:

Break

Compute and print performance metrics

Print final summary of total moves, objects collected, and performance

4. Input Test Ca s e s & Output

Test case	Grid	Initial Grid	Starting Location	Final Grid
1	8*8	0 # H H H H 0 0 # 0 0 0 # H 0 # # H # # # 0 0 H 0 # # # H H # # H # # # R 0 0 0 0 # # 0 # 0 H # 0 0 0 H # # H # 0 # 0 0 0 # 0 #	(5, 5)	0 # H H H H 0 0 # 0 0 0 # H 0 # # H # # # 0 0 H 0 # # # H H # # H # # # . 0 0 0 0 # # 0 . . H # 0 0 0 H . . H # 0 # 0 0 R # 0 #
2	8*8	# 0 # 0 # 0 H H 0 0 # 0 0 H # # # # H # H H # H H # H H 0 0 0 0 H # R # # # H # 0 # 0 # 0 H # 0 0 # 0 # H # 0 H 0 H # 0 # # 0 H	(3, 5)	# 0 # 0 R 0 H H 0 . . . . H # # # . H # H H # H H . H H 0 0 0 0 H . . . # # H # 0 # . . 0 H # 0 0 # 0 # H # 0 H 0 H # 0 # # 0 H
		# H 0 H # 0 # H		# H R H # 0 # H

3	8 * 8	H 0 0 0 H # # # 0 0 # 0 0 0 # # # H H # 0 # 0 0 0 # 0 R 0 # 0 H # H 0 H # # 0 0 # H # 0 0 H 0 H # # H H H 0 0 H	(4, 5)	H . . . H # # # 0 . . . 0 0 # # # H H . 0 # 0 0 0 # 0 . 0 # 0 H # H 0 H # # 0 0 # H # 0 0 H 0 H # # H H H 0 0 H