A* Algorithm Implementation for Wumpus World

Introduction

In this report, I briefly describe how I implemented the A* algorithm to navigate the Wumpus world. The A* algorithm is a popular pathfinding and graph traversal algorithm used to find the shortest path between nodes in a weighted graph. The Wumpus World is a grid-based environment with specific obstacles and goals, such as finding gold while avoiding pits and the Wumpus.

Methods Tested

To find a good pathfinding solution, I decided to use the A* algorithm. Here is a simple pseudocode representation of what I implemented:

Pseudocode for A* Algorithm:

---

function AStar(startX, startY):

openSet = priority queue with start node

closedSet = empty set

cameFrom = empty map

gScore = map with default infinity, set gScore[start] to 0

fScore = map with default infinity, set fScore[start] to heuristic(start, goal)

while openSet is not empty:

current = node in openSet with lowest fScore

if current is goal:

return reconstructPath(cameFrom, current)

remove current from openSet

add current to closedSet

```
    for neighbor of current:
        if neighbor in closedSet:
            continue
        tentative_gScore = gScore[current] + moveCost(current, neighbor)

        if neighbor not in openSet or tentative_gScore < gScore[neighbor]:
            cameFrom[neighbor] = current
            gScore[neighbor] = tentative_gScore
            fScore[neighbor] = gScore[neighbor] + heuristic(neighbor, goal)
            if neighbor not in openSet:
                add neighbor to openSet
    return null
function reconstructPath(cameFrom, current):
    path = empty list
    while current in cameFrom:
        add current to path
        current = cameFrom[current]
    reverse(path)
    return path
```

Note that other functions such heuristic or moveCost normally defined in the real code.

First Order Logic in Agent's Reasoning

In the Wumpus World, the agent needs to use logical reasoning to avoid hazards such as the Wumpus, pits, and to find the gold. The agent's reasoning can be described using first-order logic (FOL):

1. Percepts and Actions:

- Percept(Breeze, Stench, Glitter) indicates the presence of a pit, Wumpus, or gold in the vicinity.

- Actions include MoveForward, TurnLeft, TurnRight, Grab, Shoot, and Climb.(These are four human play gameplay)

2. Knowledge Base:
- The agent maintains a knowledge base (KB) that stores known facts about the world.
- Example FOL statements:
- Breeze(x, y) -> Pit(x', y') (A breeze implies a neighboring pit)
- Stench(x, y) -> Wumpus(x', y') (A stench implies a neighboring Wumpus)

3. Reasoning:
- The agent infers safe and unsafe locations using logical rules.

Problems Encountered

1. Not Finding the Correct Path:
- Initially, the algorithm sometimes failed to find the shortest path due to incorrect gScore updates.

2. Entering Cells with Pits and Wumpus:
- The agent occasionally moved into cells with pits or the Wumpus, leading to failures. This was due to incomplete or incorrect logical inferences about the environment.Also , I couldn't use to tell-ask interface well.

3. Bound Errors:
- During execution, the algorithm occasionally tried to access cells outside the grid boundaries, causing errors. These were fixed by adding boundary checks.

4. Path Reconstruction Issues:
- During the path reconstruction phase, ensuring the correct sequence of nodes was crucial. Initially, the path was constructed in reverse order, and a final reversal was necessary to get the correct path from start to goal.

5. Handling Multiple Paths in Priority Queue:

At first, nodes were added to the open set multiple times, causing unnecessary checks.I fixed this by updating nodes in the open set only if a lower score is found.

6.Implementing Human Playing My program allows you to play as either human or AI, so it was difficult to implement both gameplay modes at first, then I realised the algorithm so that it now works for both.

Conclusion

Using the A* algorithm in the Wumpus World requires careful planning and logical thinking to avoid dangers. The algorithm uses a priority list and keeps track of checked nodes to find the shortest path quickly. Basic logic helps the agent decide which areas are safe or unsafe. Although it was challenging at first, the final version works well, showing how strong and effective the A* algorithm is