Alex Lu 03/02/2025

ESE 561 Artificial Intelligence
Assignment 1

Question 1: PEAS and Environments

Alexa is a virtual assistant from Amazon. Alexa is capable of voice interaction and question answering and can perform a set of functions such as telling the weather, playing music, searching the web, creating lists, ordering take-out food, or calling a taxi. In this question, we will assume Alexa is not connected to other devices.

1. Perform a PEAS analysis of Alexa.
   Performance measure:
   Accuracy of voice recognition, Accuracy of language comprehension, Efficiency in generating responses, Context Awareness, Power consumption, Successful task execution, Privacy, and Security.
   Environment:
   Home, Room, Office, places with different level of background noise, Hotel, In car, Speakers with different languages, different accents, Multiple speakers.
   Actuators:
   Speakers, Internet connection.
   Sensors:
   Microphones.

2. Describe Alexa's environment using the properties seen in class.
   Partially observable, Stochastic, Sequential, Dynamic, Continuous, Single agent, Unknown.

Question 2: Search Space Formulation

We would like to formulate the following search problem:
Problem Statement: An alien starting from planet Mars wants to collect samples of Earth soil from five different continents: North America, Africa, Europe, Asia, and Australia. Since he has a very fast flying saucer, he can visit the continents in any order. After completing his mission on Earth, he will proceed to any other planet in some other galaxy.

1. What is the initial state?
   $S_{initial} = (current\ location, visited\ continents)$
   $S_{initial} = (Mars, 0)$

2. What are the possible states?
   $S = (Earth, 0)$
   $S = (Earth, [North\ America])$

$S = (Earth, [Asia, North\ America, Australia])$
$S = (Earth, [Europe, Africa, Asia, North\ America]) \cdots$
$S = (Another\ planet, [Africa, Europe, Asia, Australia, North\ America])$

3. What are the possible actions?
   (1.) Travel from Mars to Earth. (have not yet collected any soil sample)
   (2.) Move between continents on Earth.
   (3.) Leave Earth to another planet.(After collected all continent's soil samples)

4. What is the transition model?
   $S_{initail} = (Mars, 0)$
   If the alien is on Mars and travels to Earth, the state transitions to: $S = (Earth, 0)$
   If the alien moves to a continent C on Earth, the new state becomes:
   $S = (Earth, visited\ continents\ \cup\ \{C\})$
   If the alien completes all collections and leaves Earth, the final transition is:
   $S = (Another\ plenat, [North\ America, Africa, Europe, Asia, Australia])$

5. What is the goal test?
   $S = (Another\ plenat, [North\ America, Africa, Europe, Asia, Australia]$

Question 3: Search Strategies

Consider the following graph. Edges between nodes may only be traversed in the direction indicated by the arrow.
We will search the graph with the algorithms we learned, keeping a full explored set as we go. Let A be the start node and G be the goal node. As usual, where an arbitrary choice has to be made, assume that nodes are visited in lexicographical order. The table provides the heuristic values for each node.

1. What is the order of visit and path obtained with BFS? Show the queue.
   Order of visit: A, E, F, G
   Path obtained: $A \to E \to G$
   Queue: A, E, F, G, C, H (Finally, only C and H remain in the queue.)

2. What is the order of visit and path obtained with DFS? Show the stack.
   Order of visit: A, E, G
   Path obtained: $A \to E \to G$
   Stack: A, F, E, G (Finally, only F remains in the stack.)

3. What is the order of visit and path obtained with UCS? Show the priority queue. If some keys change, show the change.
   Order of visit: A, E, F, G
   Path obtained: $A \to E \to G$
   Priority queue: A, E, F, G, C, H (Finally, only C and H remain in the priority queue.)

4. What is the order of visit and path obtained with A*? Show the priority queue. If some keys change, show the change.
   Order of visit: A, E, G
   Path obtain: $A \to E \to G$
   Priority queue: A, E, G, F (Finally, only F remains in the priority queue.)

Programming Task

Read Carefully:
You must name your file `Assignment_your_name.py`.

Objective

The goal of this assignment is to implement different search algorithms (BFS, DFS, UCS, and A*) to solve a search problem. The problem will involve navigating through a simple state space, and you will be required to apply these algorithms to find the optimal path or solve the problem.

Problem Statement

Consider a generic graph search problem. You are given a state space represented as an undirected graph where each node represents a state, and each edge represents an action that leads to a neighboring state. Your task is to implement the following search algorithms to navigate through this state space and find the shortest path from a start node to a goal node.

Search Algorithms to Implement

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Uniform Cost Search (UCS)
- A* Search (Informed Search)

Examples

- A robot needs to navigate through a 5×5 grid
- Pathfinding in a Maze
- Game State Search
- City Road Navigation
- 8-Puzzle Problem
- Sudoku Solver

- N-Queens Problem
- Tic-Tac-Toe game

N-Queens Problem

Code: *Click here*

Analysis:
This program allows users to input the number of queens they want to solve for and specify the starting position for placement, as figure(1) shows. The program will place queens from top to bottom until it finds a solution or informs the user that no solution is possible.

I used 8 Queens and started from the first row, first column as an example. After implementing BFS, DFS, UCS, and A*, I visualized the results, which are shown in the figure(2)(3)(4)(5), and figure(10) shows the final result. Among them, the green Q represents the placed position, the red Q represents the visited position, and the blue Q represents the expanded position but not visited.

The results indicate that using DFS is the most efficient approach for solving this task, as it finds the solution quickly while using minimal memory, The time consumption of each search method provide in figure(6)(7)(8)(9). And through the color differentiation, the memory usage can also be observed. I incorporated an *is_valid()* function in the program to determine whether a node can be expanded. As a result, all considered nodes are valid queen placements. Given that the depth is known (equal to the number of rows on the board), DFS provides the optimal solution in this scenario.

However, this *is_valid()* function also limits the performance of the A* search. The heuristic function in A* is defined as the number of remaining queens to be placed plus the number of conflicting queen pairs. Since the *is_valid()* function ensures that no conflicts exist, it essentially negates the impact of the heuristic, restricting A*'s effectiveness. Nevertheless, I implemented A* to demonstrate my understanding of its logic for potential future modifications.
Lastly, for the UCS search method, I set the cost of placing each queen to 1. This aligns with a real-world logical approach, but since every step has an equal cost of 1, UCS performs similarly to BFS, leading to the same results.

Future work:
Modify the node expansion rule so that expansion can occur regardless of Queens conflicts, making every position on the board a node. In this way, the advantage of A* can be more easily highlighted.

```
Please Enter the number of Queen you have:8
Please Enter the first Queen's position(1~8): 1
Start putting Queen from the top of board
```

Figure 1: User Input Interface

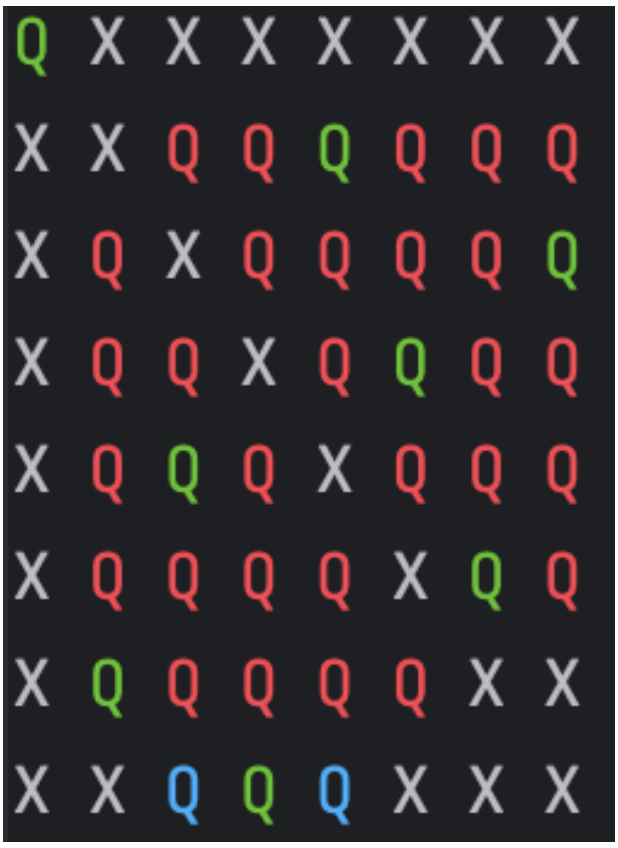

Figure 2: BFS

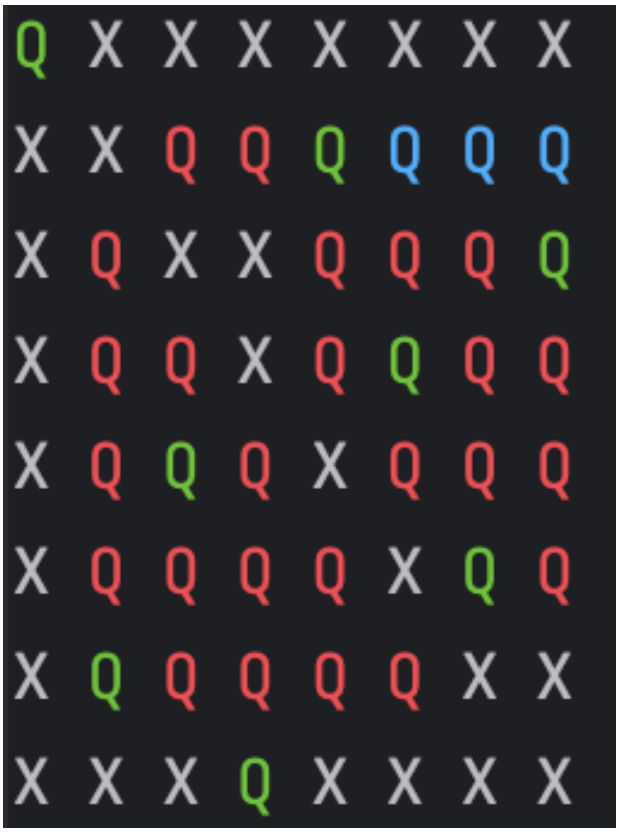

Figure 3: DFS

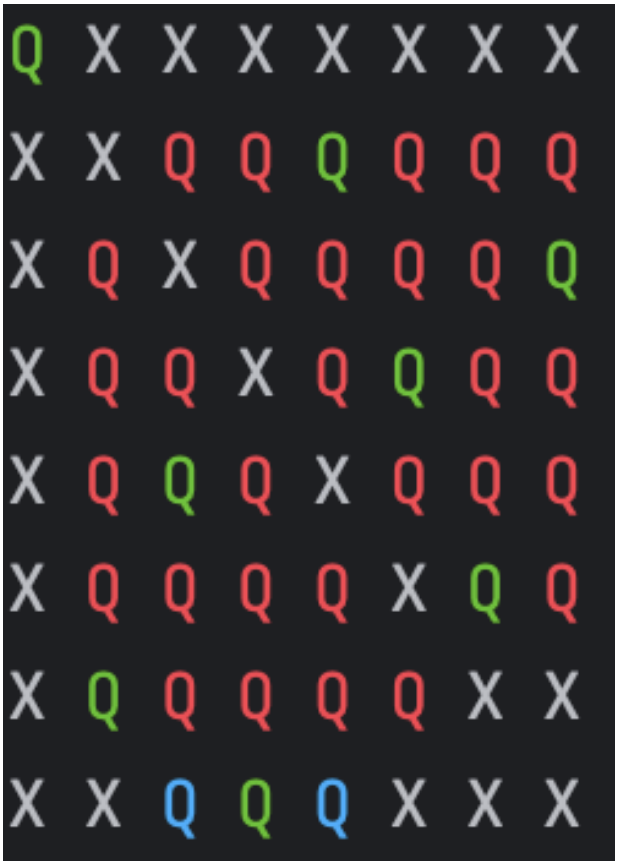

Figure 4: UCS

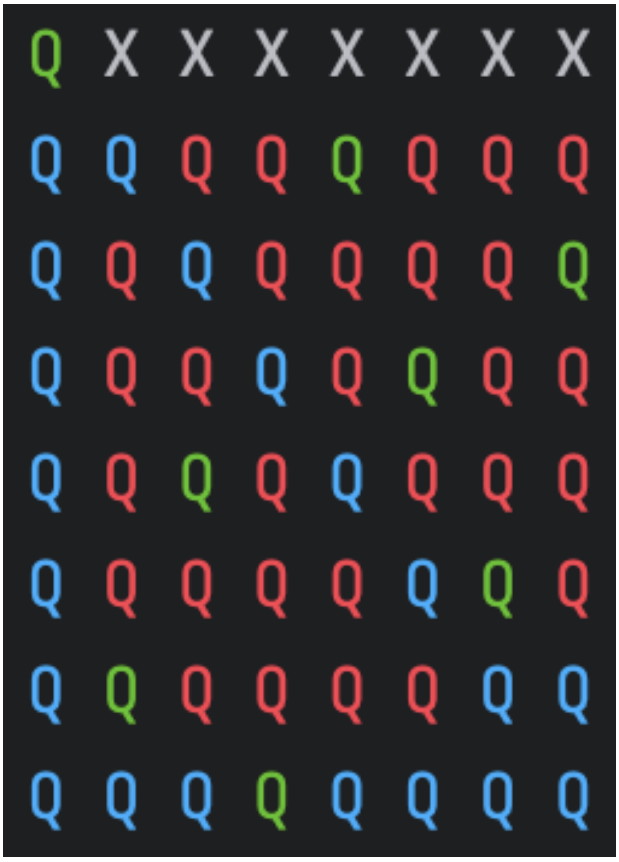

Figure 5: A*

```
node_explored num:224
Searching time: 0.019459s
How many nodes have tried: 224
```

Figure 6: BFS time consumption

```
node_explored num:113
Searching time: 0.007691s
How many nodes have tried: 113
```

Figure 7: DFS time consumption

```
node_explored num:224
Searching time: 0.031958s
How many nodes have tried: 224
```

Figure 8: UCS time consumption

```
Searching time: 0.108206s
How many nodes have tried: 224
```

Figure 9: A* time consumption

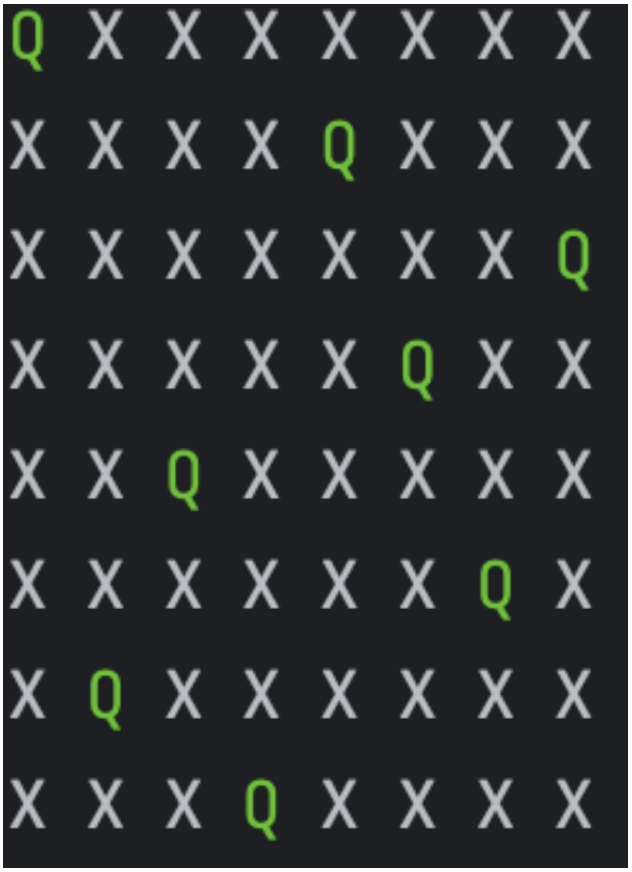

Figure 10: Final result