EAST WEST UNIVERSITY

Department of CSE

ASSIGNMENT-01

(Spring'24)

Topic: Enhanced Dynamic Robot Movement Simulation

Submitted to:

Dr. Mohammad Rifat Ahmmad Rashid

Assistant Professor
Department of Computer Science & Engineering

Course name: Artificial Intelligence

Course code:CSE366

Submitted by-

Raiyan Islam

ID:2021-3-60-219

Section:03

Department of CSE

Enhanced Dynamic Robot Movement Simulation

Introduction

In this project, we implemented two popular pathfinding algorithms, Uniform Cost Search (UCS) and A* Search, to find the optimal path from a start point to a goal point in a grid-based environment. Additionally, we introduced the concept of charging points to address the limited battery life of an agent traversing the grid. The objective was to find the shortest path while ensuring that the agent can recharge its battery when necessary.

Problem Statement

The problem involved navigating a grid environment with obstacles from a start point to a goal point. The agent had limited battery life and needed to recharge at specific charging points to reach the goal.

Grid Generation

We randomly generated a grid environment of size **grid_size** with a specified probability of obstacles (**obstacle_probability**). The start and goal points were placed at opposite corners of the grid.

Pathfinding Algorithms

We implemented two pathfinding algorithms:

- **Uniform Cost Search (UCS)**: Explores paths in order of increasing cost, ensuring that the shortest path is found.
- ***A Search**:** Utilizes a heuristic function in addition to path cost to guide the search towards the goal, often resulting in faster convergence compared to UCS.

Charging Points

To manage the agent's battery life, we introduced charging points along the path. The agent recharges its battery when it reaches a charging point if the battery level falls below a certain threshold.

Implementation

The project was implemented in Python using object-oriented programming. The key classes implemented were:

Node: Represents a state in the search space with information such as state, parent, action, and path cost.

Priority Queue: A data structure for storing nodes with priorities, used for managing the frontier in the search algorithms.

Environment: Defines the grid environment, including obstacles, start, and goal points, as well as actions and results.

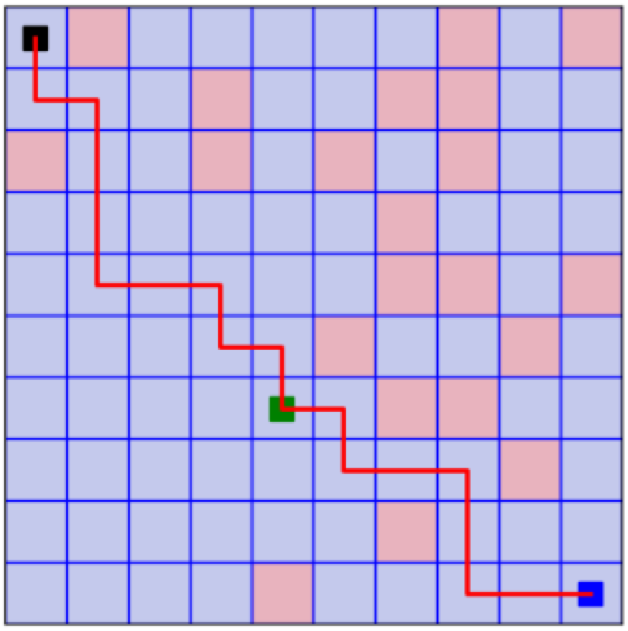
Agent: Implements the pathfinding algorithms (UCS and A*), manages the agent's battery, and determines charging points.

Results

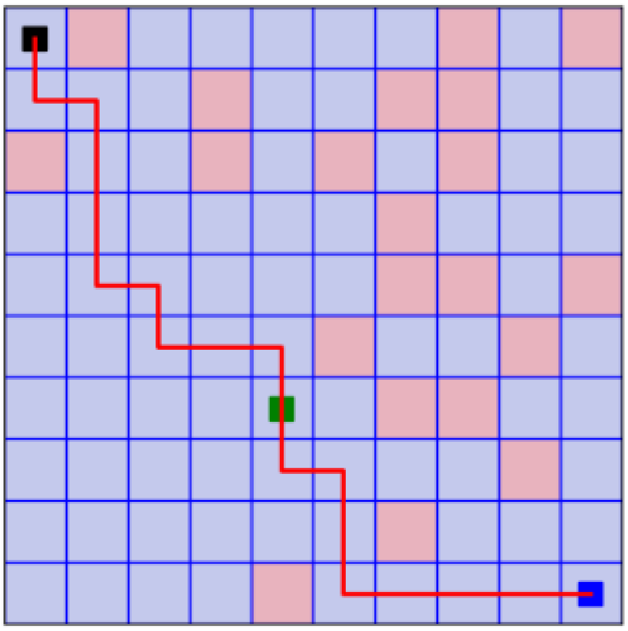
We successfully obtained optimal paths using both UCS and A* algorithms while considering the agent's battery life. The charging points were strategically placed to ensure the agent could reach the goal without running out of battery.

Output:

```
This is the solution using Uniform Cost Search algorithm
(0, 0) = 100
(0, 0) = 100
(1, 0) = 90
(1, 1) = 80
(2, 1) = 70
(3, 1) = 60
(4, 1) = 50
(4, 2) = 40
(4, 3) = 30
(5, 3) = 20
(5, 4) = 10
(6, 4) = 100
(6, 5) = 90
(7, 5) = 80
(7, 6) = 70
(7, 7) = 60
(8, 7) = 50
(9, 7) = 40
(9, 8) = 30
(9, 9) = 20
Charging Points: [(6, 4)]
```



```
This is the solution using A* Search algorithm
(0, 0) = 100
(0, 0) = 100
(1, 0) = 90
(1, 1) = 80
(2, 1) = 70
(3, 1) = 60
(4, 1) = 50
(4, 2) = 40
(5, 2) = 30
(5, 3) = 20
(5, 4) = 10
(6, 4) = 100
(7, 4) = 90
(7, 5) = 80
(8, 5) = 70
(9, 5) = 60
(9, 6) = 50
(9, 7) = 40
(9, 8) = 30
(9, 9) = 20
Charging Points: [(6, 4)]
```



Why A* is better than UCS in python?

Ans-A* Search is often considered better than Uniform Cost Search (UCS) in certain scenarios due to its ability to guide the search more efficiently towards the goal node. Here are some reasons why A* may be preferred over UCS in Python or any other programming language:

1. **Heuristic Function**: A* utilizes a heuristic function that estimates the cost from the current node to the goal node. This heuristic information helps A* prioritize exploring paths that are more likely to lead to the goal, resulting in faster convergence towards the optimal solution. UCS, on the other hand, does not use any heuristic information and explores paths solely based on their cost.

2. **Optimality**: A* guarantees optimality when an admissible heuristic is used. If the heuristic function never overestimates the actual cost to reach the goal, A* will always find the shortest path. UCS also guarantees optimality, but it may explore unnecessary nodes in its search space, leading to slower performance in certain cases.
3. **Time Complexity**: In many cases, A* can achieve better time complexity than UCS due to its ability to prune unpromising paths early in the search process. By prioritizing nodes with lower estimated costs (based on the heuristic), A* can avoid exploring unnecessary parts of the search space, leading to faster search times.
4. **Space Complexity**: A* typically requires more memory compared to UCS because it maintains additional data structures (such as a priority queue) to store and prioritize nodes based on their estimated costs. However, the trade-off in space complexity is often justified by the improved efficiency and faster convergence of A*.
5. **Application in Real-World Problems**: A* is commonly used in real-world applications such as pathfinding in video games, robotics, and route planning systems. Its ability to find optimal paths efficiently makes it well-suited for scenarios where time and resource constraints are critical factors.