EAST WEST UNIVERSITY

Assignment Report

Course Code: CSE366

Course Title: Artificial Intelligence

Section: 4

Assignment No. 2

Submitted to:

Dr. Mohammad Rifat Ahmmad Rashid

Assistant Professor, Department of Computer Science and Engineering

Submitted by:

Mahjabin Tasnim Samiha

ID: 2021-3-60-271

Date of Submission: 4 April, 2024

Assignment: Enhanced Dynamic Robot Movement Simulation

Objective: Create and deploy a sophisticated simulation environment to help a robot navigate a dynamic grid. The goal of this project is to increase knowledge of object-oriented programming (OOP), task optimization, safety, energy management techniques, and algorithms for navigation and pathfinding.

Initial: We are working with two popular algorithms which are Uniform Cost Search and A-star search.

Environment: A grid-based environment is navigated by an agent using the environment class as a simulation framework. Enabling the agent to travel from an initial point to a designated destination while avoiding obstacles is the main objective: to generate a representation of the agent's surroundings. An basic grid pattern is used to set up the environment, with 1 signifying obstacles and 0 open areas. Crucial elements are the starting point (start) and the goal location (goal).

The class provides techniques to help the agent move and make decisions. The actions technique takes grid borders and impediments into account when determining what possible actions the agent can perform from a given state. These movements include 'UP, DOWN, LEFT, and RIGHT.' The agent can switch between states while exploring thanks to the result method, which computes the new state that arises from doing a certain action.

One important feature is the is_goal method, which compares the current state with the predefined goal to determine whether the agent has arrived at its destination. This technique helps identify when the agent has finished its navigation task successfully.

All things considered, the code creates an adaptable setting for grid-based pathfinding issues, providing a basis for creating and evaluating algorithms for agent navigation, obstacle avoidance, and goal accomplishment in a grid-world simulation.

Priority Queue: The PriorityQueue class provides a priority queue data structure by acting as a wrapper around the heapq module in Python. To effectively manage items with related priority, this data structure is necessary. Elements can be added to the queue with a priority value, and they are retrieved in ascending priority order. The class has methods to insert an element with a given priority, retrieve the element with the greatest priority, and determine whether the queue is empty.

The Node class, which is frequently employed in algorithms like A* or Dijkstra's, depicts a state inside a search tree. Information about the agent's current state (where it is in the grid), its parent node in the search tree, the steps it took to get there, the path cost from the start node to this node, and the battery level (shown as a percentage) are all contained in each node.
A unique comparison function (__lt__) is also included in the Node class to help in sorting nodes in the priority queue. By comparing nodes according to their path costs, this technique allows them to be added to the priority queue in ascending cost order.

Agent with Uniform Cost Search: a weighted graph's minimum-cost path from a start node to a goal node can be found using a variation of Dijkstra's approach called uniform cost search. To effectively control the exploration process, the method makes use of a priority queue in addition to various data structures.
The first step in the process is to initialize a priority queue with the start node and its initialized zero cost. To hold data about each node, three dictionaries are set up: one for the starting node's initial value, one for the parent node, and one for the cost to reach the node.
The node with the lowest cost is then repeatedly removed from the priority queue by the algorithm. When the dequeued node is the intended node, the algorithm provides the battery and path.

Agent with A-star Search: A star search is an informed best-first search algorithm that efficiently determines the lowest cost path between any two nodes in a directed weighted graph with non-negative edge weights123

The code defines a class Agent_aStar that takes an environment as an argument and implements the a_star_search method. The method takes an optional heuristic function as an argument and returns a path and a dictionary of battery levels for each node in the path. The method also keeps track of the number of times the battery was recharged.

The code uses a priority queue to store the nodes in the open list, where the priority is determined by the sum of the path cost and the heuristic value. The code also uses dictionaries to store the parent, the cost, and the battery level of each node. The code iterates through the open list until it finds the goal node or the list is empty. For each node, it generates its successors and updates their values according to the A star search algorithm. The code also checks the battery level of each node and recharges it if it falls below a threshold. The code allows for a maximum number of retries if no valid path is found. The code also defines a helper method reconstruct_path that returns the path from the start node to the current node by following the parent pointers.