East West University

Department of Computer Science and Engineering

Semester: Spring 2024

Assignment 1

Course Code: 366

Course Title: Artificial Intelligence

Section: 4

Submitted By:

MD. Iftakher Alam

2020-2-60-003

Submitted To:

Dr. Mohammad Rifat Ahmmad Rashid

Assistant Professor

Department of Computer and Science Engineering

East West University

Date of Submission: March 7, 2024

Assignment: Enhanced Dynamic Robot Movement Simulation

Objective: Design and implement an advanced simulation environment for a robot navigating through a dynamically created grid. This project aims to deepen understanding of basic programming concepts, object-oriented programming (OOP), algorithms for navigation and pathfinding, task optimization, safety, and energy management strategies.

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

Environment: The environment class provides a grid-based environment navigation framework for an agent. The main objective is to model the environment around the agent such that it can travel from a starting point to a designated destination while dodging impediments. A grid structure is used to start the environment, with 1 signifying obstacles and 0 open spaces. The two most important positions are the beginning (start) and goal (goal) positions.
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 outcome method determines the altered state that follows a particular action, enabling the

Priority Queue: The Priority Queue 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. Every node contains data on the agent's present state (where it is in a grid), its parent node in the search tree, the steps it took to get there, and the

Agent with Uniform Cost Search: A Uniform Cost Search version of Dijkstra's algorithm for determining the least expensive route between a start node and a goal node in a weighted graph. The exploration process is effectively managed by the algorithm through the use of a priority queue and additional data structures.
First, a priority queue is initialized with the start node and its initialized zero cost. The parent node, the distance to the node, and the battery level (one serving as the starting value for the start node) are the three dictionaries that are set up to record data about each node.

Subsequently, the algorithm repeatedly removes the node from the priority queue that has the lowest cost. The algorithm returns the path and battery if the dequeued node is the goal node.

In case no valid path is found, the algorithm prints a message and returns an empty list and an empty dictionary. The algorithm includes a helper function to reconstruct the path from the parent dictionary, starting from the goal node and working backward to the start node. The resulting path is then reversed and returned.

In essence, the algorithm efficiently explores the graph, adjusting costs and battery levels as needed, and provides a solution by returning the optimal path and associated battery levels from the start node to the goal node.

Agent with A-star Search: In a directed weighted network with non-negative edge weights, a star search is an educated best-first search technique that effectively finds the lowest cost path between any two nodes123.
The code defines a class called Agent_aStar, which implements the a_star_search method and accepts an environment as an argument. The method returns a path and a dictionary of battery levels for each node along the path, and it accepts an optional heuristic function as an input. The procedure also logs how many times the battery was recharged.

The nodes in the open list are stored by the code using a priority queue, where the priority is established by adding the path cost and the heuristic value. Dictionary storage is also used by the code to hold each node's cost, parent, and battery level. Until the goal node is found or the list is empty, the function iterates through the open list. It creates successors for every node and modifies their values based on the A star search method. Every node's battery level is likewise monitored by the algorithm, which recharges it if it drops below a predetermined level. If a valid path cannot be found, the code provides for a maximum number of retries.
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.