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:

Mohammad Tahmid Noor

ID: 2021-3-60-026

Date of Submission: 4 April, 2024

Assignment: Enhanced Dynamic Robot Movement Simulation

Objective: Construct and put into use a sophisticated simulation environment enabling a robot to navigate a dynamic grid. Through this project, basic programming ideas, object-oriented programming (OOP), navigation and pathfinding algorithms, task optimization, safety, and energy management techniques will all be better understood.
.

Initial: we are using two well-known algorithms: A-star search and uniform cost 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 agent can switch between states while exploring thanks to the result method, which computes the new state that arises from doing a certain action.

The is goal method, which compares the current state with the predefined goal to determine whether the agent has arrived at its destination, is a crucial component. When the agent has effectively finished its navigation mission, it may be determined with the use of this procedure.

Overall, the code establishes a flexible environment for grid-based pathfinding problems, offering a foundation for developing and testing algorithms related to agent navigation, obstacle avoidance, and goal achievement in a simulated grid world.

Priority Queue: The Priority Queue class provides a priority queue data structure by acting as a wrapper for Python's heap module. 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 Priority Queue class provides a priority queue data structure by acting as a wrapper for Python's heap module. 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.

Agent with Uniform Cost Search: a version of the Dijkstra method called Uniform Cost Search that determines 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 traces back through the parent nodes to return the path and battery levels if the dequeued node is the goal node. If not, it determines a new cost by adding one to the existing cost and a new battery level by deducting 0.1 from the current level for each neighbor of the current node (indicating a 10% reduction for each move).

The algorithm recharges the battery to one and increases the count of recharges if the battery level falls to or equals 0.1. Next, if the neighbor's cost is not included in the cost dictionary or if the new neighbor's cost, priority, parent, and battery level
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.

Agent with A-star Search : In a directed weighted graph 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 nodes.123

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. A helper method is also defined in the code.