MARS ROVER NAVIGATOR

Monalisha Ojha
Department of Mathematics
Birla Institute of Technology
Mesra, 835215
monalishaojha974@gmail.com

July 23, 2020

ABSTRACT

Mars Rover Navigator is an AI agent which will help the mars rover to find the shortest path between two points while avoiding obstacles on the way.This project uses 6 different pathfinding algorithms i.e., Dijkstra's,A* Search,Greedy Best-first Search,Bidirectional A* Algorithm,Breadth-first Search and Depth-first Search.Furthermore, to enhance the efficiency, i have added 6 different types of mazes algorithm. This is a part of Microsoft Codess Mentorship Program 2020

***K*eywords** A* Search · Dijkstra · Breadth First Search · Depth-first Search · Recursive Diversion

1 Introduction

Mars Rover Navigator is a simple path finder visualizer web application.It will act as an AI agent in the future devices to determine the path.

Artificial intelligence is defined as a study of rational agents. A rational agent could be anything which makes decisions, as a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance).An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

This project uses 6 types of pathfinding algorithm in order to route mars rover from your origin to it's destination. This is simply a tool that visualizes how these types of pathfinding algorithms work. With this app, we can visualize how the Breadth First Search, Depth First Search, A* ,bidirectional A* and Dijkstra pathfinding algorithms operate upon custom drawn graphs.In addition to this , this application uses mazes and walls as the predefined obstacles to make the application more challenging.The maze algorithm that is used in this application include Recursive Diversion (both vertical and horizontal) maze,stair,weight and simple maze.

2 Shortest Path Algorithms

This sections explains the basics of the algorithms used in this application.

2.1 Dijkstra's

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors. The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

2.2 A* search

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

2.3 Breadth-first search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.It uses the opposite strategy as depth-first search, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes.

2.4 Depth-first search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

2.5 Bidirectional search

Bidirectional search is a graph search algorithm that finds a shortest path from an initial vertex to a goal vertex in a directed graph. It runs two simultaneous searches: one forward from the initial state, and one backward from the goal, stopping when the two meet. The reason for this approach is that in many cases it is faster: for instance, in a simplified model of search problem complexity in which both searches expand a tree with branching factor b, and the distance from start to goal is d, each of the two searches has complexity O(bd/2) (in Big O notation), and the sum of these two search times is much less than the O(bd) complexity that would result from a single search from the beginning to the goal.

3 Maze Algorithms

This section describes the mechanics of maze used in this application

3.1 Recursive Division

Mazes can be created with recursive division, an algorithm which works as follows: Begin with the maze's space with no walls. Call this a chamber. Divide the chamber with a randomly positioned wall (or multiple walls) where each wall contains a randomly positioned passage opening within it. Then recursively repeat the process on the subchambers until all chambers are minimum sized. This method results in mazes with long straight walls crossing their space, making it easier to see which areas to avoid.

3.2 Basic Algorithms

Here, the mazes are created using stair demonstration loop and other basic looping and recursion techniques.

4 Conclusions and Future work

This has been a great platform to learn the basics of graph theory and AI agents.I have learned a lot during the four weeks of this program through the mentors and lectures provided the engage 2020 team.