ARTIFICIAL INTELLIGENCE

**Path Planning using A\* Final Report**

*Omar F.*

*Perlanie P.*

*Cole S.*



WILFRID LAURIER UNIVERISTY

*Dr. K.*

**ABSTRACT**

**(do we need one?)**

1. INTRODUCTION

Using Artificial Intelligence, we present a solution a solution to a grid based maze with multiple robots

and a rendezvous point. Given a text file with a x by y grid maze, robot(s) coordinate, and the coordinates of the rendezvous point we must find a path from each robot start point to the final rendezvous coordinate. Using A\* heuristic algorithm we are able to create a solution for finding the shortest path for each robot.

1. PROBLEM FORMULATION
   1. INPUT

Input is taken through a text file with the following information:

8 10                 // the room has dimensions 8 by 10

2                 // there are N = 2 robots

2 1                 // 1st robot initial position: point (2,1)

8 2                 // 2nd robot initial position: point (8,2)

4 7                 // the rendezvous point R has coordinates (4,7)

1000000001           // room points (0,7), (1,7), ... (9,7)

1100000011

0000000000

1000110001

1001111001

0001111000

0000110000           // room points (0,1), (1,1), ... (9,1)

1100000011           // room points (0,0), (1,0), ... (9,0)

* 1. INITIAL STATE

Robot’s starting position given by an (x,y) coordinate. In the example above there are two robots, one starts at (2,1) and the other starts at (8,2)

* 1. ACTIONS

Robots can move one position left, right, up and down from its current position.

* 1. TRANSITION MODEL

Depending on action, and the starting position the robot coordinates will change. Robots cannot make a certain move if the move will result to its position being on an obstacle.

* 1. GOAL TEST

Robot checks if it is at the rendezvous point.

* 1. PATH COST

f(n)=h(n)+g(n). Where f(n) is the total path cost to move to the next node, h(n) is the line distance from the current node to the goal node, g(n) is the distance from the robot’s starting position.

1. A\* SEARCH

The A\* algorithm is a popular algorithm for pathfinding and graph traversal. It avoids expanding all possible paths which can be expensive. It uses the total estimated solution cost to determine each move towards the end node. The estimated solution cost is defined as:

***f(n)=g(n)+h(n)***

Where ***g(n)*** is the cost to reach the current node from the start node, ***h(n)*** is the estimated cost to get from the current node to the end node and ***f(n)*** is the estimated total cost of the cheapest solution.

* 1. HEURISTICS
  2. ADMISSIBILITY

1. IMPLEMENTATION

For our artificial intelligence final project, we decided to implement a program to help a robot find a path to get to its desired destination. We have decided to use the A\* algorithm and the language we will use to implement the project is Java. Currently we have 4 different classes.

* 1. THE NODE CLASS

The node class contains all the deliverables for each coordinate on the puzzle. For each node object instance the object will have, a straight line distance for how far the said node is from the starting point point (g-value), an actual distance for how far the said node is to the rendezvous point (h-value), the position of the node (row,col), a boolean value for if the node is a object or not, and lastly a parent node so if there is a solution it can easily be traced back to return a full path.

/\*===============================================================  
Node Class  
----------  
  
This class create a instance for each node (position on the maze) which includes  
the specific nodes function values (g,h and f) its row and column position (x,y),  
if the node is a object, and each parent node it has.  
  
================================================================\*/

public class Node{  
 private double g=0;  
 private double h;  
 private double f;  
 private int row;  
 private int col;  
 private boolean isObject;  
 private Node parent;  
  
 public Node(int row, int col, boolean isObject){  
 this.row=row;  
 this.col=col;  
 this.isObject=isObject;  
 }

* 1. THE ROBOT CLASS

This class defines the robot and its current position. It also contains an array list of nodes that lead the

/\*===============================================================  
Robot Class  
----------  
This class create a instance for each robot which includes the position  
of the robot.  
================================================================\*/  
  
public class Robot{  
 int row;  
 int col;  
 private ArrayList<Node> path= new ArrayList<Node>();  
 public Robot(int row, int col){  
 this.row=row;  
 this.col=col;  
 }  
  
 public void addToPath(Node newPathNode){  
 this.path.add(newPathNode);  
 }  
  
 public ArrayList<Node> getRobotPath(){  
 return this.path;  
 }  
  
 /\*===============================================================  
 getRow: gets the row value of the robot  
 ================================================================\*/  
 public int getRow(){  
 return this.row;  
 }  
   
 /\*===============================================================  
 getCol: gets the col value of the robot  
 ================================================================\*/  
 public int getCol(){  
 return this.col;  
 }

* 1. THE MAZE CLASS
  2. THE PATHFINDING CLASS
  3. MAKING IT FASTER

When we first tested our program, we saw that the robots could make it through the maze if there was a solution. However, we noticed running a 1000 by 1000 maze would take about 15 minutes. We then tried to increase the speed by adding a priority queue instead of an array list. Running through the maze again we observed the runtime to be 9 minutes. We then found out that there was a lot of time spent confirming whether nodes have already been visited or not. Initially we used a linked list to hold the visited nodes which is very inefficient because if there is a possibility of 1000000 different nodes to check it will take a long time to check 1000000 nodes. As a result we switched from using a linked list to using a hash set because will allow us to find the node if it was visited we are looking for in an instance.

1. CONCLUSION
   1. OUTPUT  
      X represents the rendezvous point, S is the robot’s starting point and \* are the nodes marked to be part of the Robot’s path.

1000X00001

1100000011

0000000000

1000110001

1001111001

0001111000

0000110000

1100000011

====================================================================

Destination: (4,7)

====================================================================

Robot Starting Position: (2,1)

Getting Path...

Path:

[2,1] => [2,2] => [2,3] => [2,4] => [2,5] => [2,6] => [3,6] => [4,6] => [4,7]

Visual Path:

1000X00001

11\*\*\*00011

00\*0000000

10\*0110001

10\*1111001

00\*1111000

00S0110000

1100000011

====================================================================

Robot Starting Position: (8,2)

Getting Path...

Path:

[8,2] => [8,3] => [7,3] => [7,4] => [6,4] => [6,5] => [5,5] => [4,5] => [4,6] => [4,7]

Visual Path:

1000X00001

1100\*00011

0000\*\*\*000

100011\*\*01

1001111\*\*1

00011110S0

0000110000

1100000011

* 1. STUFF