CS 4710 - Artificial Intelligence
HW2 - Path Finding

Justin Ingram (jci5kb)
Megan Bishop (mgb5db)

Robot Search Strategies

HAL9000 - A* for non-probabilistic searching

The basic implementation uses a priority queue of all of the neighbor points with scores allocated based on their distance to target. It begin by removing the best point from the queue (at first this will the start position, the only point in the queue at the time) and putting it in the set of points that have been considered. The point is pinged, and if it can be moved to all of its neighbors will be checked in turn. If the neighbor is out of bounds it will be ignored, if it already in the set of points that have been considered it mean that there is already a shorter path to them and they will also be ignored. Remaining neighbors are given a score based on their distance to the target. If the points are not in the queue then they will be placed in a map indicating where they are coming from (the point they are a neighbor to) and will be added to the queue. If the points are in the queue already and its new score is less than its previous then it will have its score adjusted and its path in the map altered to match. Once all neighbors have been considered it will then repeat, going back to the beginning and removing the the best point from the queue. If it exhausts all points in the queue (meaning it has searched all reachable parts of the map) and is not able to locate the target then there is no possible path.

This robot, while it does not handle uncertainty, will always find the optimal path with the minimum pings.

WallE - Repeated calls to A*

This robot uses the path finding algorithm of HAL9000, except that if the target can not be reached it will return the path of the best point it can get to. When it returns a path to a better position (or the target) it will attempt to follow it. Should it run into a wall or reach the end of its path without locating the target it will stop and return to path finding, using its new position as its starting point.

This is a simplistic way of working around the uncertainty problem. As the robot will continually go back looking for a path to the target assuming that it didn't get there because of uncertainty, it will never terminate on an impossible map. The uncertainty function provided leads to lots of walls being returned where there are none, this means that WallE will often not be able to make any large moves in a single pass and if he does it will not be an efficient path. As he does not track previous paths he can loop back on himself and it might take several tries to find his way around an obstacle. He will be able to solve most maps though as long as there aren't any overly large obstacles to overcome.

Rosie - Repeated calls to A* with adaptive depth of pings

This robot only attempts to plan a small distance into the map, starting at only pinging *n* spaces away. If it is able to successfully plan a path and follow it, then the distance will be

increased by 1 for the next round of planning. If the move fails then the distance will be decremented and the data structures will be cleared.

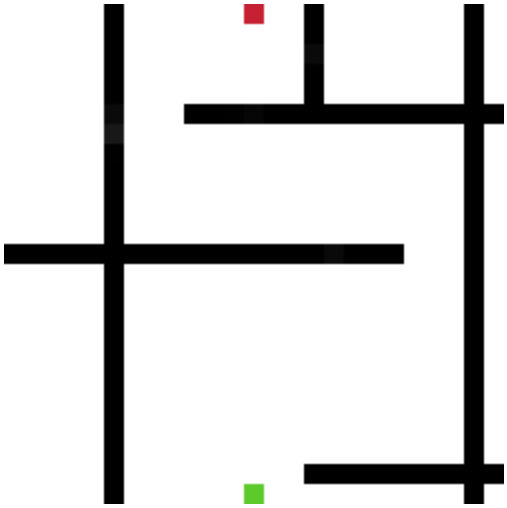
The paths found tend to be roughly a s efficient as those found by WallE, but Rosie will be able to do it in fewer pings. It had the similar downfall where it could only 'see' so far into the map and therefore had difficulty getting around large obstacles, this made it so that it was not able to solve the 'lines' map which WallE was able to. This is because the distance would be reduced to 1 at a part and it would be trapped in a small area and would never be able to see where to go.

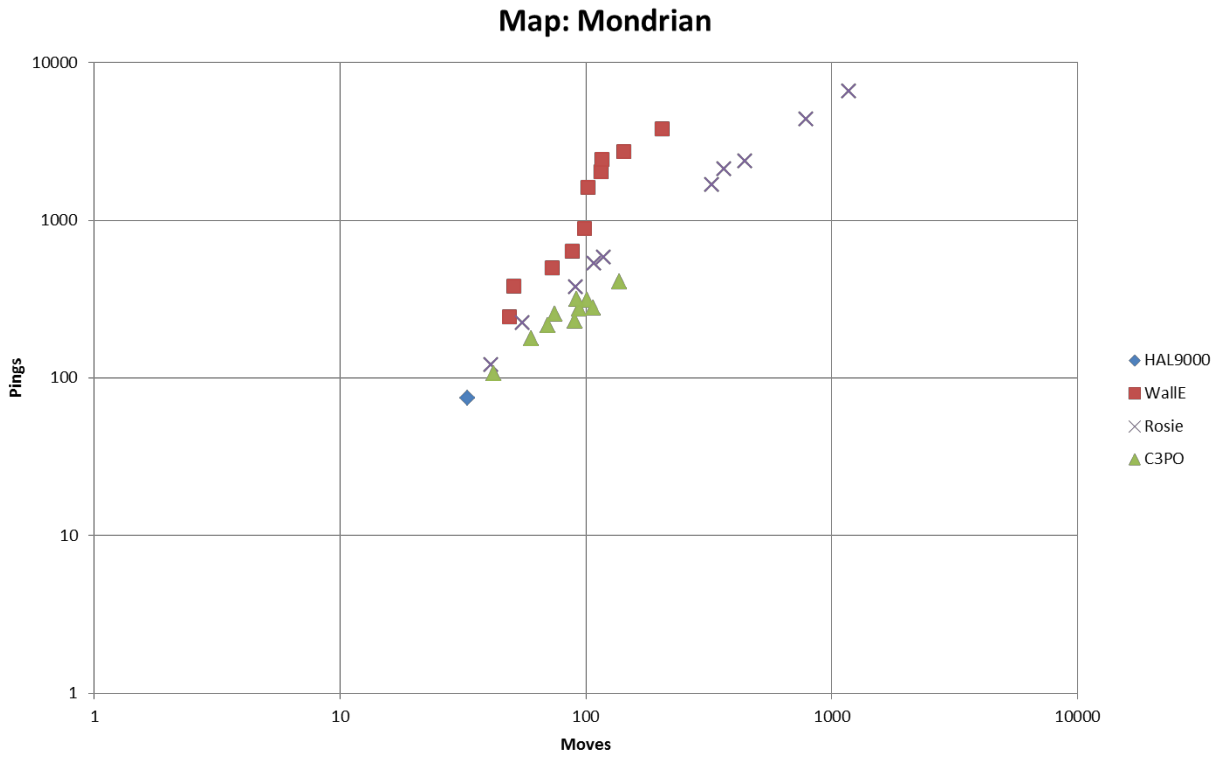
C3PO - Adaptive depth and accuracy map

Here there was the expansion of a knowledge base where it remembered what it thought the map looked like. If a point was pinged multiple times it would weight the accuracy of the data and store what the point most likely was (an opening or a wall).

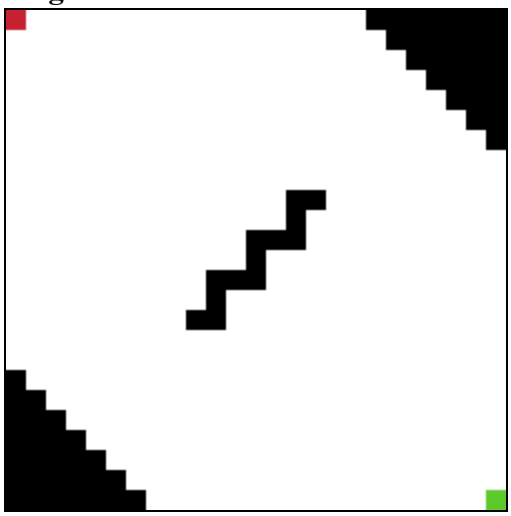
C3PO could do as well as HAL9000 under the right conditions (as seen on 'diagonal'). On all maps it proved to do the best for both moves and pings. It was not able to solve the spiral problem though, most likely because the uncertainty function was showing walls were there were none and it was therefore unable to find a path.

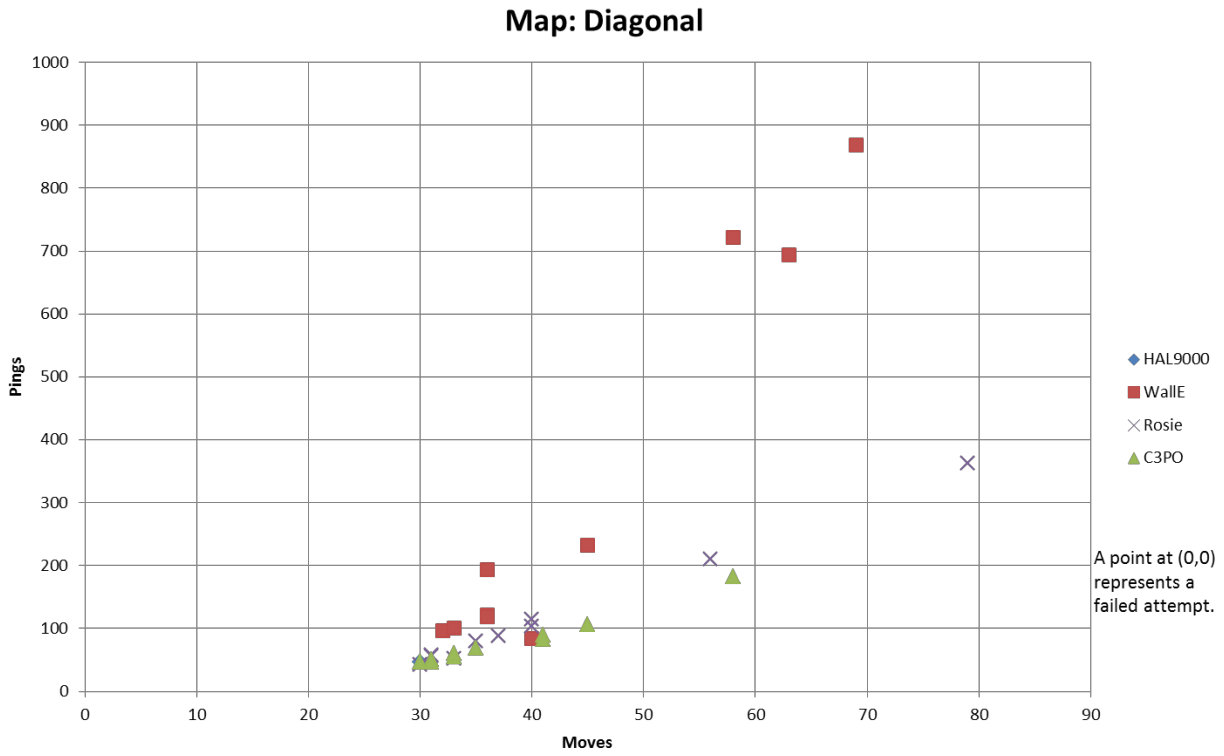
Mondrian



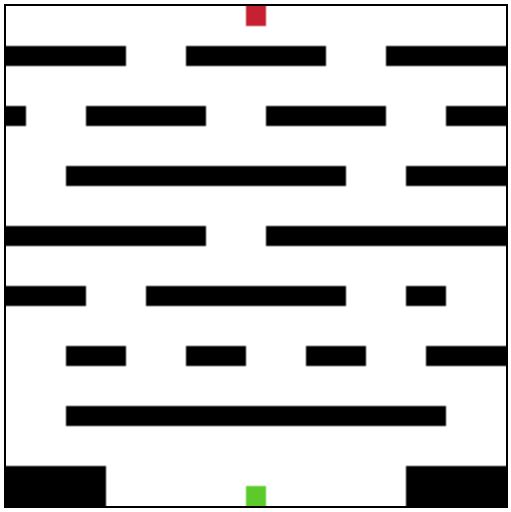

Map: Mondrian
10000
1000
100
10
1
Pings
1
10
100
1000
10000
Moves
HAL9000
WallE
Rosie
C3PO

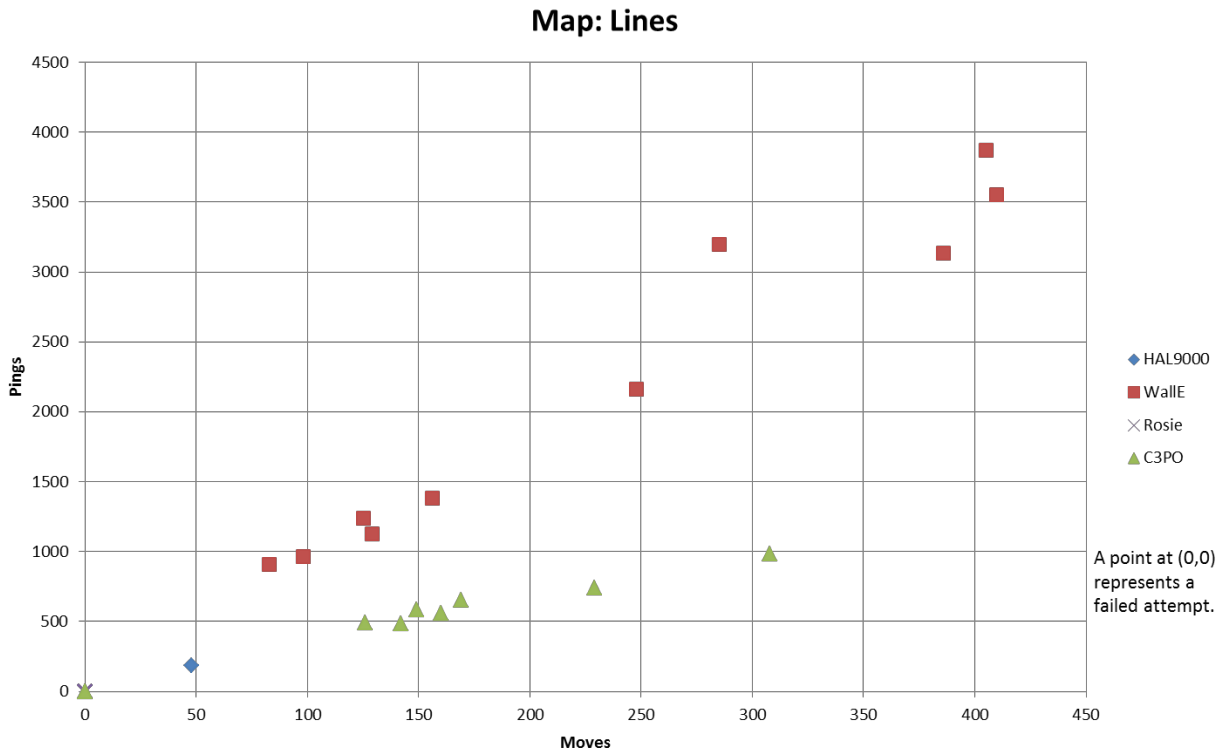
Diagonal



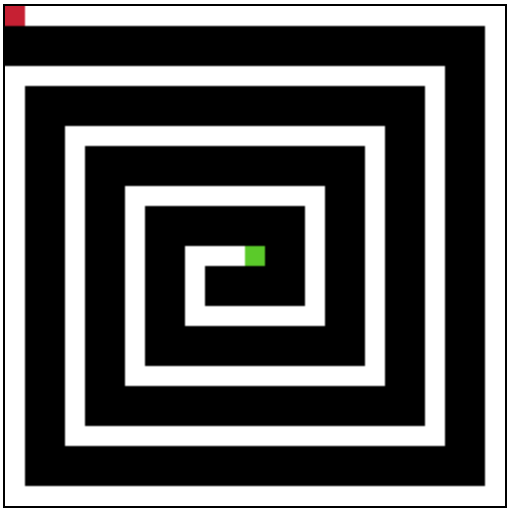

Map: Diagonal
Pings
Moves
0
100
200
300
400
500
600
700
800
900
1000
0
10
20
30
40
50
60
70
80
90
HAL9000
WallE
Rosie
C3PO
A point at (0,0) represents a failed attempt.

Lines

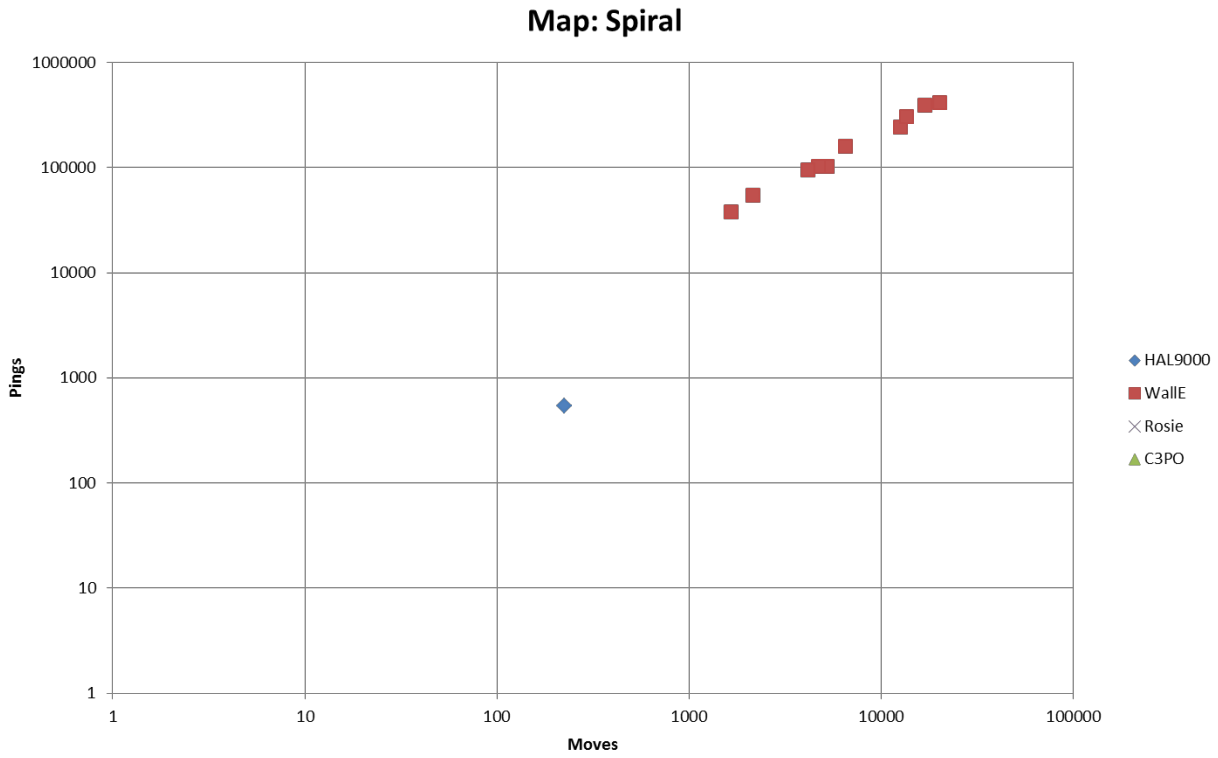




Spiral

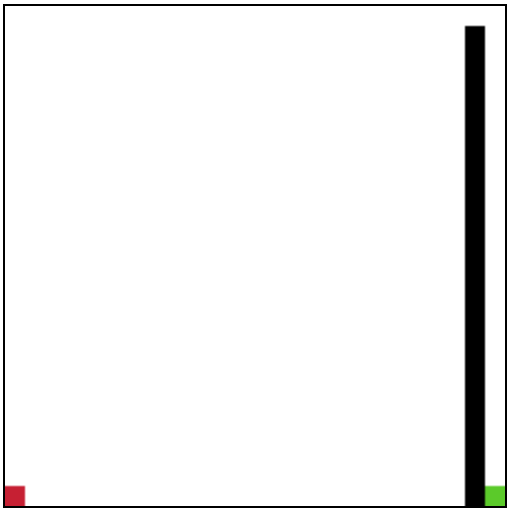


This map should be the worst case scenario for moves and pings. There is only one possible path to the target which make it nearly impossible to solve when using uncertainty. This is because the robot would see walls where there were none and would not be able to get through as there was no way around them.



Hitting the Wall

One area where none of our probabilistic robot implementations could succeed was the map that had a large "wall" separating the finish node from the rest of the map. This proved to be one of the worst case scenarios for all of the robots dealing with a probabilistic map. The robots fared poorly because the the algorithms always tried to push the robots toward the goal, and the depth at which the robots could successfully ping was never far enough to "see" around the wall.



Strategies such as trying to incentive new knowledge would help, but would never result in the robot solving the graph. Preventing the robot from revisiting points might also have helped, but would have resulted in failures in other areas such as a map like the following:

S O O O
X O X O
X O X O
X X X F

If the robot moved down the second column of nodes, it would need to revisit nodes it had already visited in order to back out of the corner. Given more time, we would like to worked on more ways to handle this situation, such as a better heuristic function and perhaps

an “escape” procedure, that would revert to something more akin to best first search until the robot was in a clearer section of the map start A* again.