Capstone Project
Plot and Navigate a Virtual Maze

Taylor Mitchell
October 24th 2017

I. Definition

Project Overview

This project is based off the Micromouse competition, where teams design a robot to navigate a maze from the corner to the center without any other knowledge of the structure of the maze aside from it can see around it.

For this project, a virtual agent is created to navigate a similar maze. The agent navigates the maze twice. During the first run, the agent must find it's way to the goal, but can also choose to continue searching the maze afterwards. During the second run, the agent can use the information gained from the first run to navigate it's way to the goal as fast as possible.

Problem Statement

The maze which the robot will be navigating is a 12x12, 14x14, or 16x16 grid. The starting square is the bottom left corner of the maze, and the goal is a 2x2 room in the center. The robot is considered at the center of the square it's located, and is facing one of four orthogonal directions (up, down, left, right). It is equipped with sensors which allow it to see 3 squares in front, to the left, and to the right of itself. Each time step, the robot first receives information from its sensors. Based on this information, the robot can rotate 90 degrees clockwise or counterclockwise, or not rotate, and then move up to 3 squares forward or backwards. Once the robot moves, this concludes the time step.

During the first run through the maze, the robot starts at the bottom left corner of the maze, facing the "up" direction. From here, the robot must navigate through the maze until it reaches the goal room. Once the robot chooses to finish the first run, it's relocated back to the start of the maze in its original orientation. The robot then begins its second run, using the information it gained from the first run to get to the goal room in the least time steps possible.

Specifically, the goal is to try to get the lowest final score as possible. The scores is calculated using the following equation:

Score = (time steps taken in first run)/30 + (time steps taken in second run)

The second run has a much larger effect on the score, so it may be worthwhile to spend extra time exploring the maze in the first run even after finding the goal, just in case there might be a

shorter path than the one found for getting there. Deciding when to stop exploring and whether it's worthwhile at all is an interesting problem. Because the maze is unknown, unless we explore the entire maze, we won't know whether or not we've found the optimal solution. So the question of when to decide to finish exploring the maze will be one aspect of the project.

Once finished the first run, the agent will have information about the parts of the maze it has explored. The next question will be how to determine the shortest path to the goal given this information. An exhaustive search would be an option since computation time doesn't affect the final score, but for practical purposes, an approximate solution will be used.

To solve this problem, we'll first consider the two objectives of the first run:

1) Find the goal room
2) Explore the maze

These two objectives will be used to design a decision making algorithm for the robot. We know that the goal room is at the center of the maze, and we know the size and shape of the maze grid. Given that, we should prioritize moving locations which are closer to the center. We also want to explore as much of the maze as possible, so we want to prioritize locations we haven't been to over the ones we have. The way I implemented this was first by assigning each square a utility value, where the goal room square have a utility of 0, and the utility of squares decreases as you move radially out from the center. As the robot explores the maze, whenever it lands on a square, we decrease the utility of that square by some fixed amount. Each time step, the robot will choose to move to the square with the highest utility. This algorithm will combine the objective of avoiding explored squares, and moving towards goal room.

The above algorithm should eventually get the robot to the goal room. Once in the goal room, we must decide whether to continue exploring the maze or not. I'll try creating another decision making algorithm which works it's way from the goal room back to the start while avoiding previously explored squares. This directed exploration will provide at least one other route from the start to the goal room, while also doing general exploration along the way which may turn out to be useful when determining the best path. I'll run through the mazes many times, with and without the added exploration, to determine experimentally whether or not extra exploration generally increases the final score.

Metrics

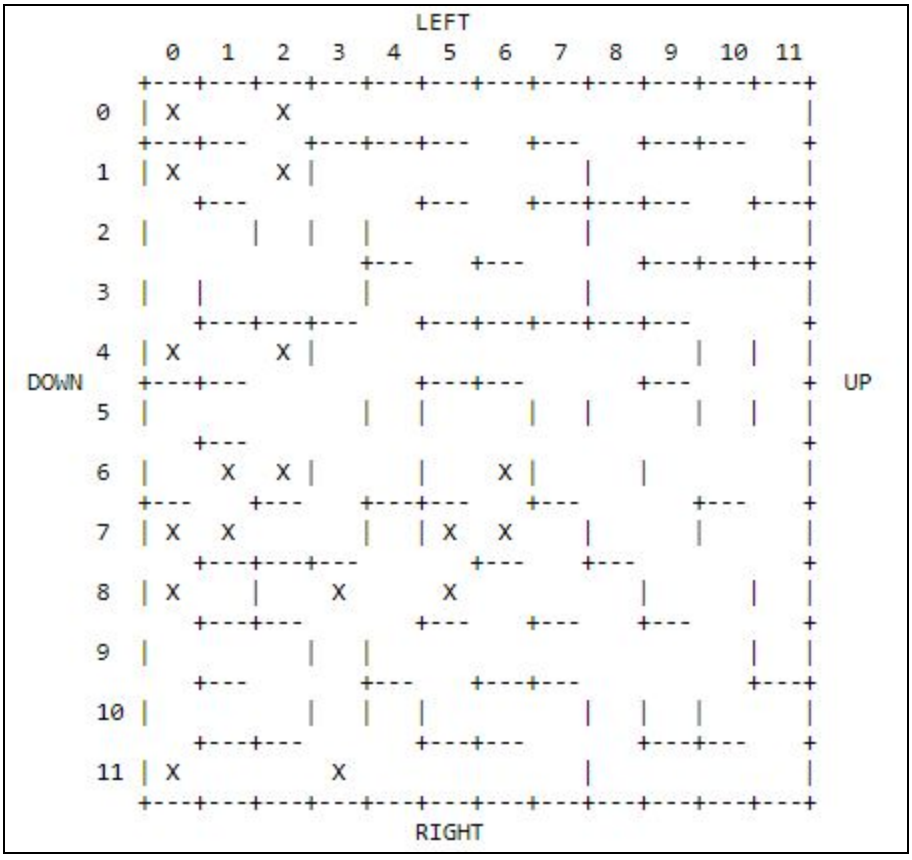
1. Time steps for first run
2. Time steps for second run
3. Final score

II. Analysis

Data Exploration and Visualization

One of the three mazes provided, test_maze_01, is shown below. The X's show the optimal path from the start to the goal room. Keeping in mind the solution I proposed above, this maze may be difficult for the robot to solve. The algorithm will prefers squares towards the center of the maze, but the optimal path goes along the outside edge of the maze before a path towards the center opens up. The robot will end up wasting a lot of time exploring around the center of the maze before finding it's way around to the side where moving towards the center will actually lead it towards the entrance to the goal room.

Test Maze 1 - Optimal Path



Algorithms and Techniques

Important Information Stored by Robot

Utility Table

As I mentioned above, the robot uses an estimate for the utility of a square to decide which one to move to. Initially, the utility of each square is the negative of the number of steps it would take to get from the square to the goal room (if there were no walls in the way).

```
[[-10.  -9.  -8.  -7.  -6.  -5.  -5.  -6.  -7.  -8.  -9. -10.]
 [ -9.  -8.  -7.  -6.  -5.  -4.  -4.  -5.  -6.  -7.  -8.  -9.]
 [ -8.  -7.  -6.  -5.  -4.  -3.  -3.  -4.  -5.  -6.  -7.  -8.]
 [ -7.  -6.  -5.  -4.  -3.  -2.  -2.  -3.  -4.  -5.  -6.  -7.]
 [ -6.  -5.  -4.  -3.  -2.  -1.  -1.  -2.  -3.  -4.  -5.  -6.]
 [ -5.  -4.  -3.  -2.  -1.   0.   0.  -1.  -2.  -3.  -4.  -5.]
 [ -5.  -4.  -3.  -2.  -1.   0.   0.  -1.  -2.  -3.  -4.  -5.]
 [ -6.  -5.  -4.  -3.  -2.  -1.  -1.  -2.  -3.  -4.  -5.  -6.]
 [ -7.  -6.  -5.  -4.  -3.  -2.  -2.  -3.  -4.  -5.  -6.  -7.]
 [ -8.  -7.  -6.  -5.  -4.  -3.  -3.  -4.  -5.  -6.  -7.  -8.]
 [ -9.  -8.  -7.  -6.  -5.  -4.  -4.  -5.  -6.  -7.  -8.  -9.]
 [-10.  -9.  -8.  -7.  -6.  -5.  -5.  -6.  -7.  -8.  -9. -10.]]
```

Next Locations Table

The next locations table is the record of the explored maze. It's a dictionary with a key for each of the locations in the maze. The list associate with each key is of the locations that are available to move to from the key location:

```
{(0, 0): [(0, 1), (0, 2), (0, 3)],
 (0, 1): [(0, 0), (0, 2)],
 (0, 2): [(1, 2), (2, 2), (3, 2), (0, 1), (0, 0), (0, 3), (0, 4), (0, 5)],
 (0, 3): [(0, 4), (0, 5), (0, 6), (0, 2), (0, 1), (0, 0)],
 (0, 4): [(0, 3), (0, 2), (0, 1), (0, 5), (0, 6), (0, 7)]
```

Robot Functions

Get Visible Next Locations:

This function takes the list of sensor information and returns a list of the squares which are currently visible to the robot.

Get Backward Locations

This function adds the locations behind the robot that it knows it can move to based on its last move. For example, if the robot moved forward 3 steps on time step t, then on time step t+1 it won't be able to see any of the squares behind it but it knows that all three of the squares behind it are valid locations to move to. The main purpose for creating this function is to prevent the robot from getting stuck in a dead end. Now when the robot reaches a dead end, and sees no locations it can move to based on its sensor information, it will still remember that it can move back to one of the locations behind it.

Remove Dead Ends

When at a location where there are no visible locations to move to, the robot must be at a dead end. This function adds the location to a list of dead ends and then removes the location from all the lists in the next locations table.

Update Next Locations Table

This function takes a list of the locations which the robot can see are available to move to, removes from this list any locations which are dead ends, and adds to this list any squares behind the robot which it knows it can move it to. It then adds these locations to the next locations table if they weren't already there.

Get Movements

This functions takes the location which the robot has chosen to move to as input and returns the rotation and movement required to get there.

The Explorers

The functions above are used to process the sensor information and eventually return a list of locations which the robot can move to (and aren't obviously bad, like dead ends). Next, we need a decision making algorithm to pick which of the available locations to go to. For this project, I've created four different decision making algorithms which will be tested and contrasted to one another. To aid with the discussion and comparison, I've given each algorithm a name, and will refer to them as if they were explorers, each with their own preferences and biases for navigating through the maze.

The Fool (FL)
Chooses randomly between the available locations.

The Explorer of the Unknown (EU)
From the list of available locations to move to, The Explorer of the Unknown chooses the one which is least explored.

The Directed Explorer (DE)
From the list of available locations to move to from it's current location, The Directed Explorer chooses the location which leads towards the center of the maze. It does this by choosing the location with the maximum utility (from the utility table mentioned above). The Directed Explorer also prefers moving to less explored locations. This is accomplished by adding a negative reward of -10 to a locations utility each time it's visited. The value of the reward was set at a value such that less explored locations are always preferable to explored ones.

The Curious Explorer (CE)
This explorer only takes over once the goal has been found on the first run. It chooses paths which lead in the direction towards the start from the goal, and avoids locations which have already been explored.

Estimating the best path - A*

The A* path search algorithm will be used to choose an estimate of the shortest path.

A* uses a heuristic to estimate how far a given location is from the goal: It assumes that the number of steps left to get to the goal is equal to however many single square steps it would take to reach the goal from the location if there were no walls. This is called the heuristic cost estimate.

As A* searches through the possible paths, it keeps track of, for each location, the length of the shortest path which leads to that location. This value is called the gScore.

Adding the gScore and the heuristic cost estimate for a specific location gives an estimate for the length of the shortest path from the start to the goal which passes through this location. This is called the fScore.

As A* searches through the branches of paths, it keeps track of the shortest paths it's seen for arriving at every one of the locations it's looked at so far. It iteratively branches out from whichever location currently has the smallest fScore (since it's estimated to be the shortest path to the goal). It keeps doing this until it's found the goal. Whatever is the shortest path that it's come across for getting to the goal is returned.

Benchmark

The results obtained by The Explorer of the Unknown will be used as a benchmark for comparing results to the chosen solution. To obtain the average performance of The Explorer of the Unknown, 100 trials will be conducted for each maze, and the average results will be used for comparison. The Explorer of the Unknown is a very simple decision making algorithm which will always find the goal room. By taking into account the location of the goal room and implementing some guided and bounded exploration, it should be possible to create another algorithm which has improved results over this benchmark.

III. Methodology

Data Preprocessing

No data preprocessing required. Code was provided which takes a maze and the robot's location and returns the robot sensor information in a form which it can readily use to make its navigation decisions.

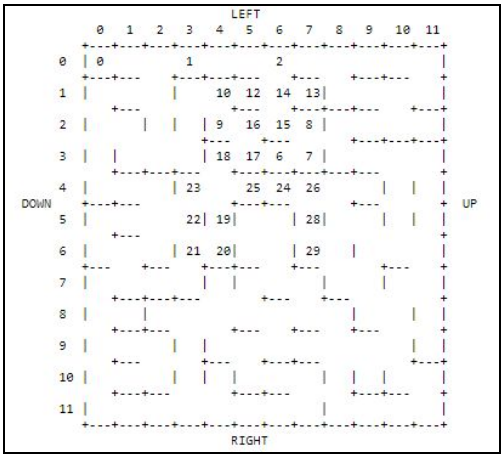
Implementation

Visualizing the robot in the maze

After some messing around trying to get the algorithms to run and to complete the maze, I created some scripts which could print a given maze, and also show the path taken by the robot through the maze. This helped me determine why the robot was getting stuck, where it was spending its time, how it was making certain decisions in certain locations, and general troubleshooting.

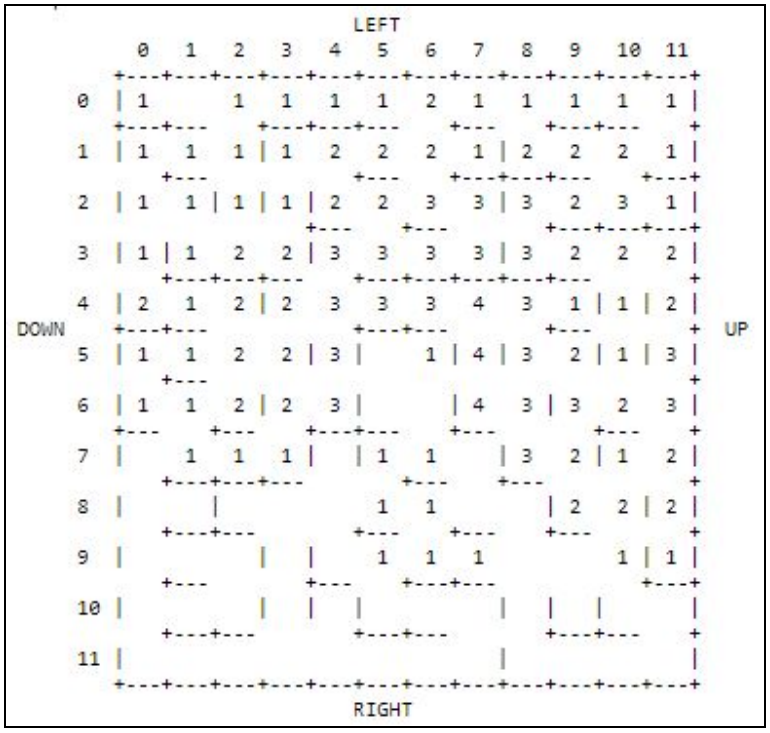
Step Count

This script takes a list of locations, the path taken by the robot, and displays at each location on the maze the step count the robot was at when it landed there.



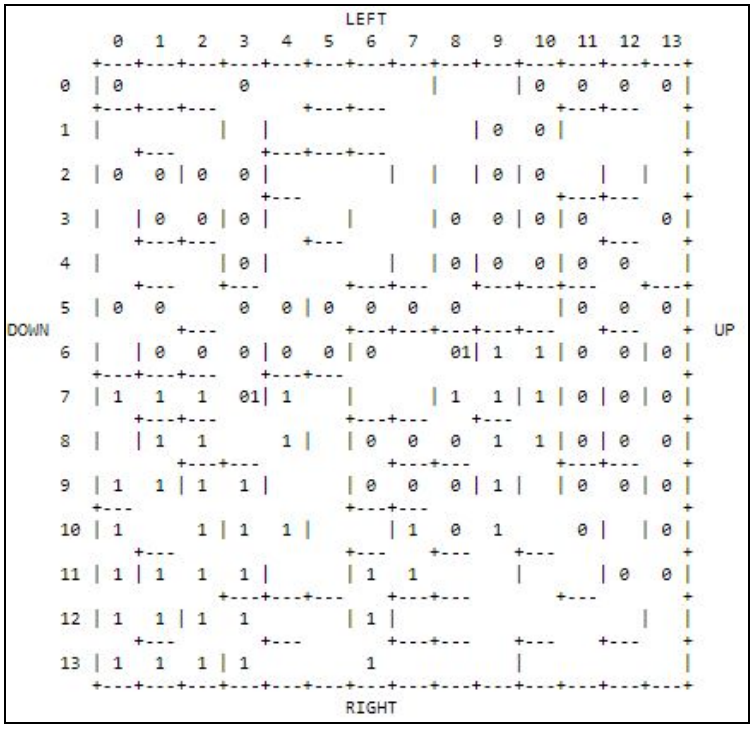
Times Visited

I also created a script which would display the number of times the robot visited each location



Multiple Paths

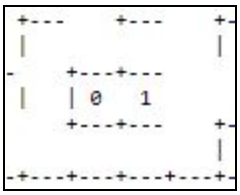
When I created and started testing The Curious Explorer, I found it useful to be able to see the path taken to reach the goal, and to see separately the path taken afterwards. The path taken to the goal is shown with 0s and the path taken afterwards with 1s.



Dealing with dead ends

The first big problem the above visualizations help me address was dead ends. I described earlier that I created the “get backwards location” function so the robot would remember the locations available to move to behind it, and that this should get the robot out of dead end scenarios. Turns out there’s a special case involving deep dead ends that this doesn’t deal with.

In the picture below there are two locations, 0 and 1:



Say the robot finds itself at location 1 and decides to step forwards to location 0. Now the robot won’t see any locations to move to, but it will remember that it can move backwards thanks to “get backward location”. It moves back to location 1. Now it’s right back where it started, and the only location it knows it can move to is 0. It’s stuck here forever!

There are two things I did to deal with this. First, I added the functionality to track and remove dead ends. Now, when the robot lands at 0, it records that this location is a dead end, and removes it from the list of locations available to move to. Now when the robot moves from 0 to 1, the location 0 in front of it will have been removed from the list of available locations by “remove dead ends”. If the robot finds itself in this situation, where there doesn’t appear to be any locations to move to, it chooses to move a single step backwards. On top of that, location 1 will be recorded as a dead end as well, since it appeared there was no where to go from there. This added functionality will get the robot out of all dead ends, and prevent the robot from ever reentering a path which only leads to a dead end.

Intermediate testing and results

At this point, all of the explorers were able to find their way to the goal, and A* was able to take the information learned from the first run and approximate the best path from the start to the goal. So I could successfully run the tester script to see the performance of each of the explorers on a single trial (when I refer to a trial, I’m referring to a completion of a first and second run through the maze).

Next, I created a script to run a single explorer through multiple trials of a maze, and record the results. This allowed me to get the average results of the explorer across many maze runs, to better understand how the explorers were performing in general.

The Directed Explorer was outperforming The Explorer of the Unknown on all mazes, but not by much. The Directed Explorer would complete the first run much faster than The Explorer of the Unknown, but the extra exploration done by The Explorer of the Unknown meant that it usually

found a shorter path for the second run. My hope was that the combination of The Directed Explorer and The Curious Explorer would achieve the best performance. The Directed Explorer would find the goal quickly, and The Curious Explorer would add some extra exploration, but in a directed fashion which would waste less time than the random exploration of The Explorer of the Unknown.

To see how the addition of The Curious Explorer was affecting the results of The Directed Explorer, I did the following: run through the maze with The Directed Explorer until the goal was found, save the path taken to get there, and use A* to find the best path from start to goal given the maze which has been explored so far. Then continue the run with The Curious Explorer until it reached it's end of exploration condition, and save the total path taken, and again use A* to find the best path given the now more explored maze. By finding the "best path" before implementing The Curious Explorer and afterwards, we can compare how the addition of The Curious Explorer affects the time taken during the first run, seconds run, and the final score.

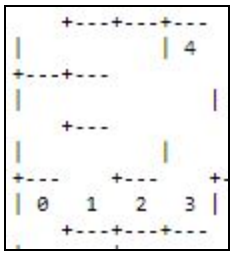
The results were mixed. On some mazes the addition of The Curious Explorer would improve results, but on others it would make it worse. The Directed Explorer + The Curious Explorer would still always outperform The Explorer of the Unknown, but it wasn't the improvement I was hoping for. In going through the trials where the addition of The Curious Explorer was reducing the performance of The Directed Explorer, I was able to find some areas for further refinement, which would hopefully improve these intermediate results.

Refinement

Improving next locations updates

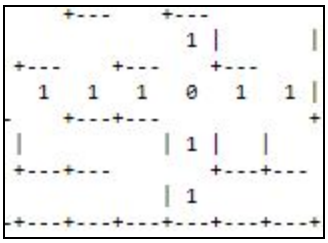
One thing I noticed when going through some of the trials of The Directed Explorer + The Curious Explorer was that sometimes The Curious Explorer would actually find the optimal path on it's way back to the start, but it was saving this information in a way that A* wasn't able to reconstruct the path.

When the explorer is moving forward through the maze, it's adding all the locations it sees around it which are available to move to to the next locations table. So even if it doesn't move to a particular location from the current one, it remembers that it can. When the explorer is moving "backwards" through the maze (ie. it's moving along a path from the goal to the start), it doesn't see the squares behind it, so they're not automatically being added to the next locations table. Let's say the robot has moved along the following path, from 0 to 4:



From 0, if the explorer moves directly to 3, “get backwards location” will add 0, 1, and 2 to the list of locations that the robot can move to from 3. But if the robot moves from 0 to 1, it only records that it can move to 0 from 1. Then it moves from 1 to 2, and records that it can also move from 2 to 1. Then it moves from 2 to 3, and records that it can also move from 3 to 2. Now when A* is passed the next locations table to work out the best path, it won’t see that it can move directly from 3 to 0, it will only know that the robot can move from 3 to 2, then 2 to 1, then 1 to 0. This clearly is not going to get us the shortest path.

To help explain the added functionality, consider the example below. The robot is located at the 0, and knows that all the locations marked with a 1 are available to move to.



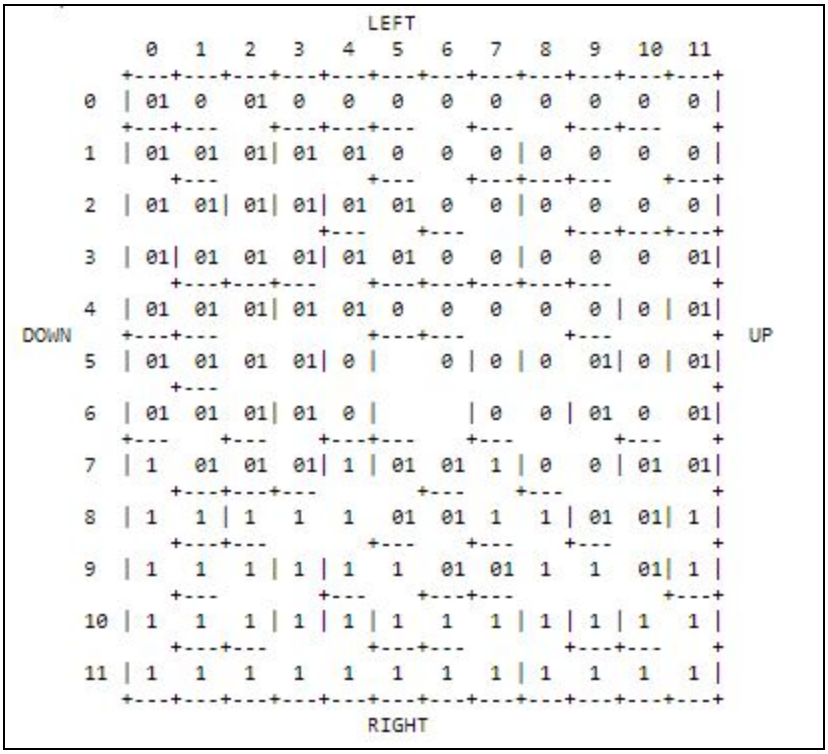
Before, only the current location (0) in the next locations table would have been updated with the (1) locations shown above. But we actually know that every location in a horizontal line can move to any of the other locations in the horizontal line (which are 3 or less squares away). The same is true for the vertical line. Now, when the “update next locations” function is called, it uses the information it has about which locations are available to move to, and the fact that the robot can move to any square from any other square along a line, to update all these locations in the next locations table.

The Curious Explorer Taking Too Long

Another thing I noticed was that, although adding The Curious Explorer was decreasing the length of the second run on average, but it was using a lot of steps to do so.

Here’s an example where The Curious Explorer helped find a shorter path but ultimately reduced the final score. By doing some extra exploration during the first run, The Curious Explorer was able to decrease the second run from 21 time steps to 19. But because the extra exploration increased the steps during the first run from 172 to 334, the final score actually increased from 26.7 to 30.1

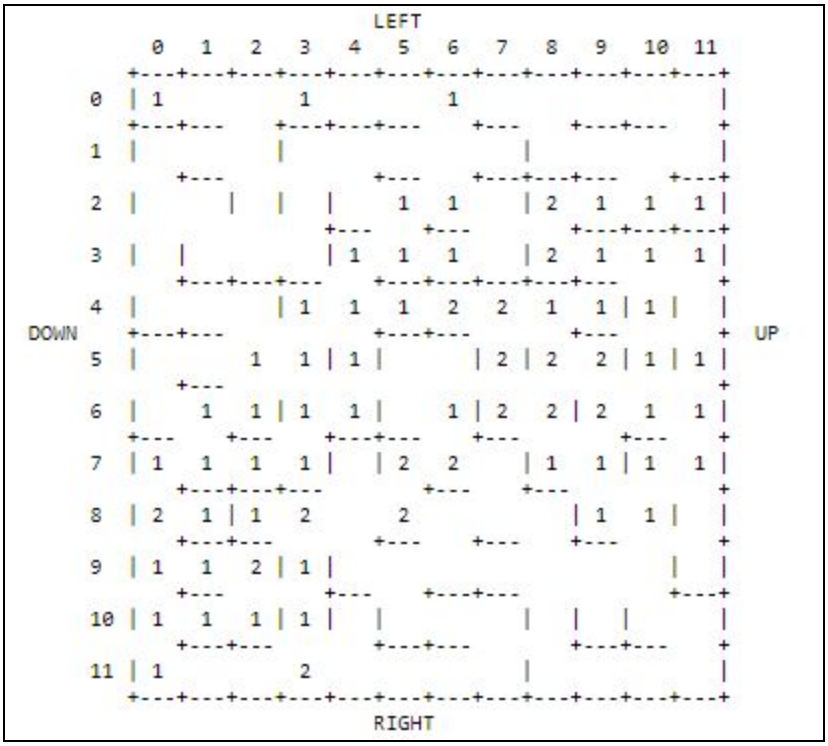
Let’s take a look at the areas explored by The Directed Explorer (the 0s) and The Curious Explorer (the 1s).

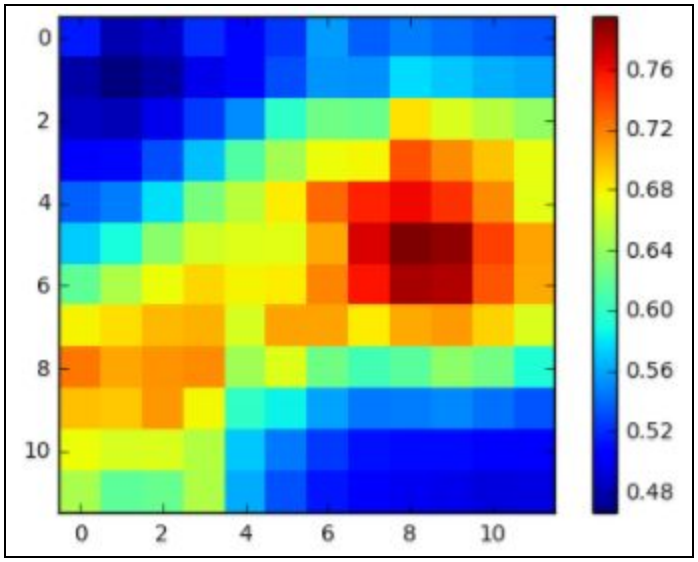


The Curious Explorer explores the lower section of the maze which was previously unexplored by The Directed Explorer. The lower left corner is the section that was needed to find the optimal path, so this is good. But we can see that a lot of time was used traversing parts of the maze which had already been explored (squares traversed by The Directed Explorer and The Curious Explorer shown with 01).

There are two changes I made to try to improve The Curious Explorer. First, I had The Curious Explorer find the area of the maze which has been explored the least at the time that the goal room is found. This is done by calculating what I called a loneliness value for each square of the maze. The loneliness value for a given square is calculated by adding 1 / (distance to visited location from square) for each location which has been explored. The lower the loneliness value, the lonelier that square is.

I found it useful to display the loneliness value for the entire maze as a heat map. This way I could check whether the areas that were coming up as “lonely” matched my intuition of which areas seem unexplored by the robot. The first image shows the maze and the number of times a square has been visited. The next one is a heat map of the loneliness values.





As we would expect, the top area and the bottom right area are considerably lonelier than the center.

Once we have this loneliness matrix, we can determine the loneliest square. This square is at the center (weighted by loneliness) of the unexplored area. The Curious Explorer resets the utility table such that it's driven towards the loneliest square. Once the loneliest square is in it's sights, it resets the utility table again, such that the robot is driven towards the start location. The goal here is to have The Curious Explorer veer towards the unexplored area, since this is

going to have new information about the maze that may be relevant. Once there, it heads back towards the start, this ensures that the area which we've now explored is connected to the start and the goal, so we know we've added at least one new path to the set of explored paths.

The second change was to add an end condition to The Curious Explorer which wasn't just finding it's way back to the start. As we saw above, making it's way all the way back to the start can have The Curious Explorer retrace paths which have already been explored. So I added an end condition to The Curious Explorer if it lands back on a square which has already been explored. This end condition only comes into effect after some number of steps from the goal, since The Curious Explorer can't avoid retracing some squares when leaving the goal room. Once away from the goal room, and since The Curious Explorer is moving towards the unexplored, it shouldn't run into an explored square for some time. As I said earlier, after exploring the unexplored area, The Curious Explorer heads back to the start to ensure this area connects back into the explored paths. If The Curious Explorer lands back on a square that was explored on the way to the goal room, we also know that we've connected back into a path which leads to the start and the goal room.

With these two changes, The Curious Explorer will always try to explore some area of the maze which it hasn't seen yet, and will always connect back into the areas of the maze which we have explored. This ensures that we're adding more paths to our set of explored paths. Now the question will be whether or not those paths end up being useful often enough to increase the final score… Let's find out!

IV. Results

Model Evaluation and Validation

For this section, I'll present the results for The Directed Explorer and for The Directed Explorer + The Curious Explorer over 100 trials for all 3 test mazes. I'll discuss the difference in the results and what they have to say about the value of taking time for extra exploration. These results will be used to select the final model which will be compared to the benchmark and further justified. The robustness of the model will be touched on during the Free-Form Visualization section.

Test Maze 01	**Average Over 100 Trials**		
The Explorers	**1st Run Steps**	**2nd Run Steps**	**Final Score**
The Directed Explorer	111.4	18.1	21.8
The Directed Explorer and The Curious Explorer	125.4	17.9	22.1
Change due to The Curious	14	-0.2	0.3

The addition of The Curious Explorer increased the number of steps taken in the first run, as it always will, but managed to decrease the average number of steps of the second run. The decrease in the second run wasn't enough to offset the increase of the first though, so the final score increased. The difference here is very small though. The problem seems to be that The Directed Explorer is exploring enough of the maze to find a path that's quite close to the optimal path most of the time. In a case where The Directed Explorer finds the optimal path, or something close, it's going to be hard or impossible to increase the score with extra exploration.

Test Maze 02	**Average Over 100 Trials**		
The Explorers	**1st Run Steps**	**2nd Run Steps**	**Final Score**
The Directed Explorer	137.1	28	32.5
The Directed Explorer and The Curious Explorer	161	25.3	30.7
Change due to The Curious	23.9	-2.7	-1.8

For the second test maze, the addition of The Curious Explorer was able to decrease the second run by a few steps without increasing the first run much at all. This maze is particularly difficult for The Directed Explorer because the only paths that get you to the goal room hug the outsides of the maze. This is exactly the area that The Directed Explorer is biased to move away from. It is then funneled in towards the center of the maze, often leaving a section of the maze unexplored.

Test Maze 03	**Average Over 100 Trials**		
The Explorers	**1st Run Steps**	**2nd Run Steps**	**Final Score**
The Directed Explorer	76.4	30.4	32.9
The Directed Explorer and The Curious Explorer	121.8	30.3	34.4
Change due to The Curious	45.4	-0.1	1.5

For the third test maze, there was only a single trial where adding The Curious Explorer was able to decrease the final score. Most of the time it just adding extra steps to the first run without actually finding a better path. With a maze this large, it seems like it becomes much less likely that any randomly selected subset of the maze will contain useful information for reducing the best path length.

Let's take a look at how the addition of The Curious affected the final score overall:
+0.3 -1.8 +1.5 = 0

No difference at all! My attempt to decide experimentally whether extra exploration of the maze is useful or not seems to have failed pretty spectacularly. What this does show is that the benefit or lack thereof is highly dependent on the structure of the maze.

Although adding up the scores across all mazes is equal, The Directed Explorer working alone was able to outperform on two of the three mazes, so I'll select this one as my final robot model.

Justification

Let's take a look at how our chosen explorer compares to the benchmark explorer. Once again, we'll take a look at the average performance over 100 trials for each maze.

Test Maze 01	**Average Over 100 Trials**		
The Explorers	**1st Run Steps**	**2nd Run Steps**	**Final Score**
The Directed Explorer	111.4	18.1	21.8
The Explorer of the Unknown	181.3	19	25
Difference	69.9	0.9	3.2

The directed explorer completes the first run and the second run faster on average. Like I mentioned in the section above, it looks like The Directed Explorer usually finds a path which is quite close to the optimal path length of 17. Even if the extra exploration that The Explorer of the Unknown resulted in the optimal path, it's unlikely that it'll do so quickly enough to make it worth it.

Test Maze 02	**Average Over 100 Trials**		
The Explorers	**1st Run Steps**	**2nd Run Steps**	**Final Score**
The Directed Explorer	137.1	28	32.5
The Explorer of the Unknown	270.4	24.2	33.2
Difference	133.3	-3.8	0.7

This one was close! Again, as I mentioned in the previous section, this maze is particularly hard for The Directed Explorer. It's biased towards the center but it needs to move towards the outside of the maze to get around a long wall before moving towards the center becomes a useful strategy. And the way it makes it's way to the center once it makes it around the wall leaves the lower section of the maze mostly unexplored, which turns out to contain some nice little short cuts. But even though The Explorer of the Unknown finishes the seconds run almost 4 steps quicker than The Directed Explorer on average, too many steps in the first run are needed to do so, leading to a higher score.

Test Maze 03	Average Over 100 Trials		
The Explorers	**1st Run Steps**	**2nd Run Steps**	**Final Score**
The Directed Explorer	76.4	30.4	32.9
The Explorer of the Unknown	224.9	27.7	35.2
Difference	148.5	-2.7	2.3

Once again, The Directed Explorer completes the first run so much quicker than The Explorer of the Unknown that a quicker second run can't make up the difference in the score.

In general, The Explorer of the Unknown learns more about the maze by taking more time to explore, and has a quicker second run because of it. But as I expected, biasing the explorers actions towards the center where the goal room leads to improved results. The speed at which it makes it to the goal room over random chance makes up for the poorer performance on the second run.

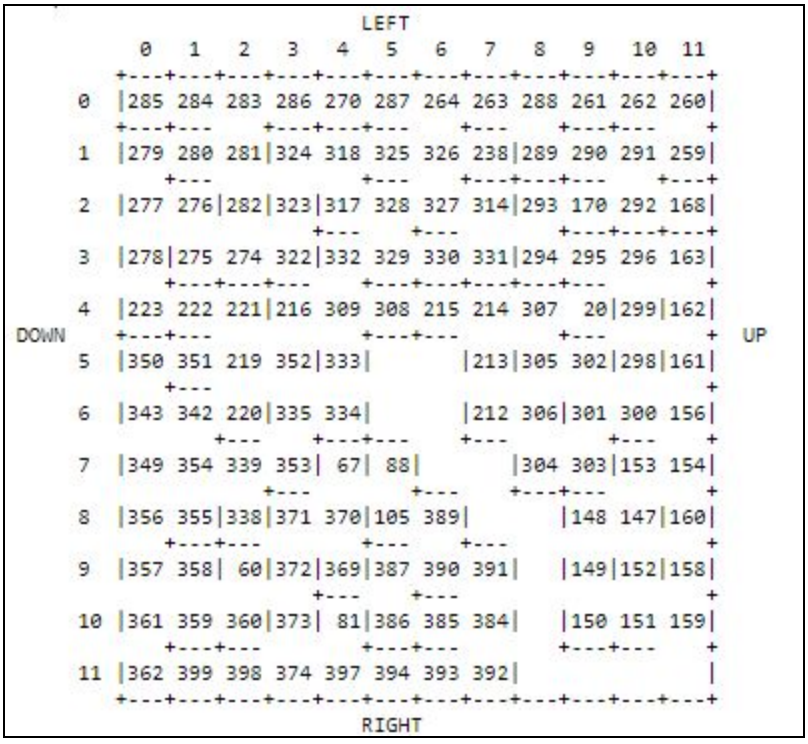
The chosen solution is able to find the goal room within the time constraint every single trial for each of the test mazes. It outperformed the benchmark algorithm, which, although it's simple, represents a fairly balanced trade-off between exploration and goal attainment. I believe this solution is significant enough to be deemed an adequate solution to this problem. With that said, there's definitely room for improvement, some of which I'll cover in the next section*.

V. Conclusion

Free-Form Visualization

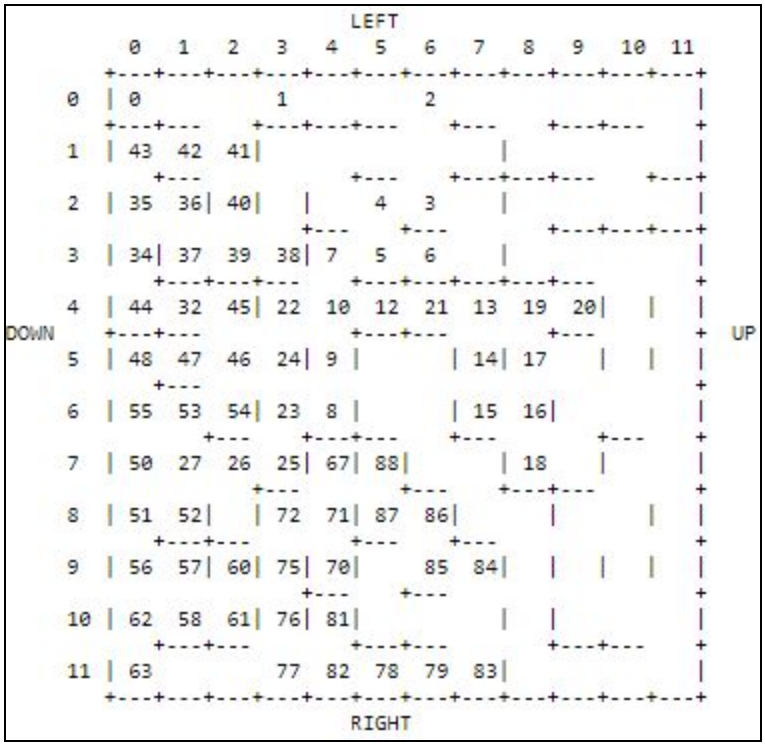
I've modified the first test maze to display a weakness of The Directed Explorer. It's still able to complete the maze in the specified time, but in some cases it takes significantly longer than the original.

Here's the maze during one of The Directed Explorer's first runs through. The number on the squares show the step count when the robot last visited the square. In this example, The Directed Explorer is on it's 400th step and still hasn't found the goal room.



At a certain point the robot should probably stop and consider what can be inferred from the information it's gotten from the maze so far. One obvious point here is that we can know which square leads to the goal room. And we also know which square leads to the square which leads to the goal room. In both cases there's only one option. We can keep working backwards until we get to the bottom right square. At this point we'll see that one of the squares we've been to is able to move to the bottom right square, which we know leads to the goal room. Finding this out on the 400th step isn't ideal though. A whole lot of time steps have already been wasted. So how far back could we have started this process of elimination?

By the 88th step, the robot had explored every square surrounding the goal room except for one. This is the point where we can begin the process of working back from the goal room to find the location furthest away from it which leads towards it:



We can work all the way back to the (8,8) square before we're uncertain about which square connects to this path. This gives us some useful information which we could start using to inform the navigation, rather than blindly following the heuristics which clearly weren't working for another 300 steps.

There are probably other ways to introduce forms of long distance planning into the robot. One might implement something similar to the loneliness matrix which I used to determine the initial direction of travel for The Curious Explorer. For example, if the robot is steeped in an area of the maze which is thoroughly explored, perhaps it should determine the area of the maze which is least explored, and switch to a temporary utility table which will drive it towards this unexplored area. Once there, it could switch back to it's original utility table and continue exploring as usual.

Reflection

The scripts I created for visualizing the maze and the path travelled by the robot, as well as saving intermediate results, like the utility table and the next locations table for the robot at each time step, incredibly useful. Between these intermediate results and the visualization, I was easily able to step through the maze with the robot to check its decision making logic and troubleshoot problems.

Just getting the code to work the way I wanted was a bit of a challenge for me. Python is a new language which I've been learning while doing this course, and my programming skill in general were very rusty.

What I found most interesting was how such simple algorithms are able to solve the mazes and how difficult it was to significantly outperform the benchmark results.

Improvement

Continuous Domain

Solving this problem in the continuous domain presents a few more problems which would need to be dealt with. First, the sensors wouldn't return nice round numbers anymore. If there were 3 squares in front of the robot and a wall, I would assume the sensors would return 3 (for squares) + 0.3 (for distance from edge of robot to edge of square it's on) - 0.05 (for wall thickness) = 3.25. My code would have to be modified to handle this. If the robot is still virtual and/or moves perfectly with noiseless sensor info, then you can still assume the robot always moves from center to center when it moves squares. If the robot is in the real world with imprecise motors and noisy sensor data, you'd have to assume the robot will never perfectly in the middle. Functionality would have to be added to use the sensors to gauge the robots distance from the wall and use this to determine the robot's location. If the walls in the continuous maze could partially cover the side of a square, but still leave enough room for a 0.4 unit robot to fit through, this would be a case where you could set up a maze which was solvable in the continuous domain but not the discrete.

Extending Project

In some ways this project was a lesson in the potential usefulness of machine learning, not by presenting a novel use of it, but by showing how time consuming it is to manually design and implement algorithms which cover all the edge cases of a problem. Although I haven't looked into it, I'd be curious if there's a way to implement a deep reinforcement learning agent for this problem. Obviously training on only the three provided mazes wouldn't generalize to unseen mazes, but if we could create a maze generator, then we could have as many training mazes as we want. At the very least, we could probably train an agent to take the sensor information as input, and output a rotation and movement which don't run the robot into walls.