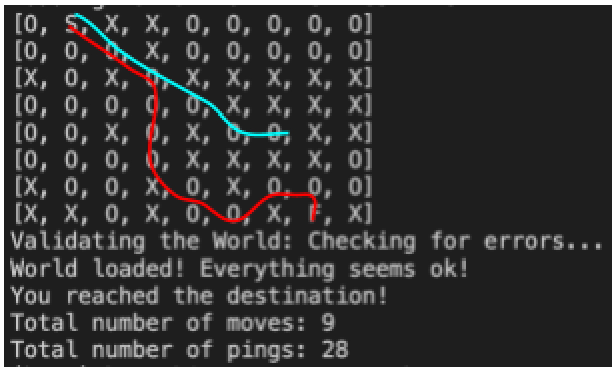
Bryan Kim bjk3yf
3/9/21
CS 4710 AI
Floryan

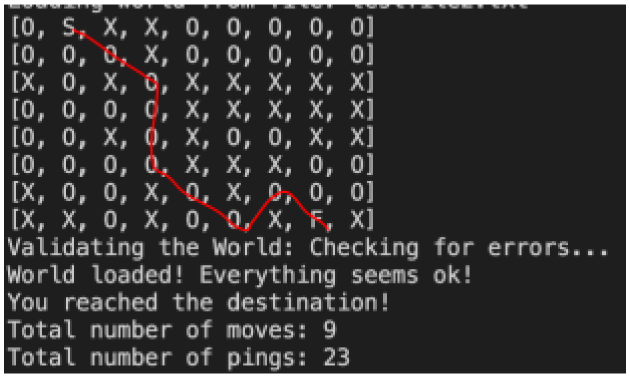
HW2 Writeup

The basic path-finding algorithm I had used in HW2 is the A* algorithm. Each position will be represented by a node containing the cost values and parent node. The algorithm contains a closed and open list with the closed list holding all the nodes that are fully searched and the open list containing nodes that have been visited but still need to be searched. My implementation also includes a wall list that will contain all the wall positions to save redundant pings on walls. The algorithm will start with the initial position in the open list and be taken out and add all the valid neighbor positions in the open list. A position will be valid if it is in the bounds of the map and is not a wall. Before a neighbor is being added to the open list, A* will check if the neighbor is already in the open list and see if the total cost is less than the one already in the open list. The same goes for the closed list and if the cost is greater in the list, the neighbor will be added back to the open list. If either check fails, the neighbor will be disregarded. Once the position reaches the goal, we will track the path by going through the parent nodes. Then the algorithm will move the robot in the final path.
The datasets that are more inefficient are the ones where there is a possible optimal path to the end that would be invalid since it would be blocked by a wall. This causes the algorithm to keep investigating the fake path, lowering the efficiency of the number of pings.
An example of the test case is shown with figure 1 and 2.



```
[O, S, X, X, O, O, O, O, O]
[O, O, O, X, O, O, O, O, O]
[X, O, X, O, X, X, X, X, X]
[O, O, O, O, O, X, X, X, X]
[O, O, X, O, X, O, O, X, X]
[O, O, O, O, X, X, X, X, O]
[X, O, O, X, O, X, O, O, O]
[X, X, O, X, O, O, X, F, X]
Validating the World: Checking for errors...
World loaded! Everything seems ok!
You reached the destination!
Total number of moves: 9
Total number of pings: 28
```

Figure 1



```
[O, S, X, X, O, O, O, O, O]
[O, O, O, X, O, O, O, O, O]
[X, O, X, O, X, X, X, X, X]
[O, O, O, O, X, X, X, X, X]
[O, O, X, O, X, O, O, X, X]
[O, O, O, O, X, X, X, O, O]
[X, O, O, X, O, X, O, O, O]
[X, X, O, X, O, O, X, F, X]
Validating the World: Checking for errors...
World loaded! Everything seems ok!
You reached the destination!
Total number of moves: 9
Total number of pings: 23
```

Figure 2

In both figures, the red line represents the most optimal path while in figure one, the blue line represents the “fake” path that fools the algorithm. We see that in figure one, A* will have to check if the blue route is possible before finding the actual solution and results in a higher number of pings. Thus we can conclude that the more there are of paths that look optimal, the more the algorithm will have to check and reduce efficiency.

The uncertain robot uses a similar algorithm to the regular robot with the only difference being that the uncertain robot will continue to loop the algorithm if the robot did not meet the destination. This is due to the fact that the robot can ping the incorrect value from a certain range and will throw off the algorithm from choosing the right path. So on the first iteration, the algorithm will obtain a path and try to move the robot. Since we know that the closer the location is to the bot the more accurate the ping will be, the robot will be able to move for a couple of spaces closer to the end point. There will be a point where the robot will try to move in a wall position, resulting in the bot to be stuck there since the move command will no longer work. This is when the algorithm will run again and move a couple spaces closer to the end and continue to loop till the bot completes its journey. This “fix” to the uncertainty of ping, however, is highly inefficient since for each iteration of the algorithm the robot has to ping for a new path. This will eventually lead to a high total count of pings but will still find the solution.

To handle the cases with no answers, I made an arraylist that stores all the points where the robot ends in the failed paths. If the robot ends on the same path as before, it will be assumed that there is no possible path, resulting in the end of the function. This will prevent the uncertain robot from being stuck in an infinite loop.

The uncertain robot does produce similar results to the certain robot when the test cases are no larger than around a map size of 5x5, but the values of the number of pings would still fluctuate. This fluctuation is also another major issue of the uncertain robot. Due to the problem being random, the algorithm will be fed a different problem every time a path is being requested. Thus I see cases from between 1000 to 12000 pings for larger test cases such as myInputFile4.txt that were given to us. Some additions to the solution I have thought of was to utilize a memory to store the values of each ping in order to save some ping calls, but the pings were too volatile to reliably use the previous pings without rechecking them. Maybe a future implementation would include using temporary robots to scout the terrain to debunk any possible routes, but again, would be very costly in the number of resources.