Intro to AI - Homework 1

Brandon Cheng 207009794
Arya Shetty 206009145
Damon Lin 208008630

Due Date: Feb. 27, 2024

Part 0

Build your 50 grid world environments with the above or a similar process of your preference and store them. You are encouraged to consider methods available online for maze generation. Provide a way to load and visualize the grid world environments you have generated.

We used JavaScript and HTML canvases to generate and display our mazes. Through the script, you will be able to see the process and the final path all the A* algorithms take.

To generate the mazes, we used a DFS/backtracking algorithm, which is illustrated on our display. The maze generation algorithm picks random directions, creating a path as it goes, until it hits a dead end. Once this happens, it backtracks until there are options to explore. This continues until every square has been visited. Due to the nature of the algorithm, our mazes tend to be long and windy, and the agent runs into walls very frequently.

Instead of storing the mazes, we decided to go for a different approach to ensure that tests are repeatable. When generating the maze, we initially used the Math.random() function in Javascript, but if we used a seeded randomizer then we could regenerate mazes and ensure that they generate the same way given the same seed. Since there is no seeded randomizer in Javascript, we used an implementation that we found online (here) which uses a cyrb128 hash to first convert seed strings into 128-bit hashes, then use the first 32 bits as the input to a splitmix32 pseudo-random number generator. With this approach, we can run our maze generation algorithms with the same seeds to get the same output. We found that this was an elegant solution ensuring that our tests are repeatable and comparable between different algorithms.

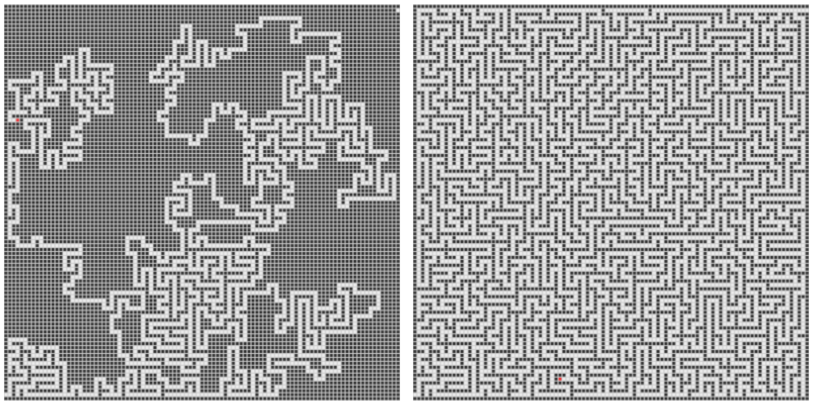

Figure 1: Generation of a 50x50 maze with the seed "asdf"

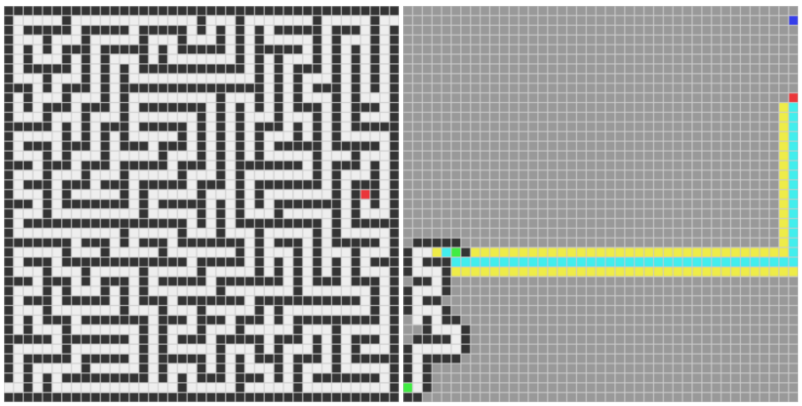

Figure 2: Using forward A* to search for a path in a 41x41 grid

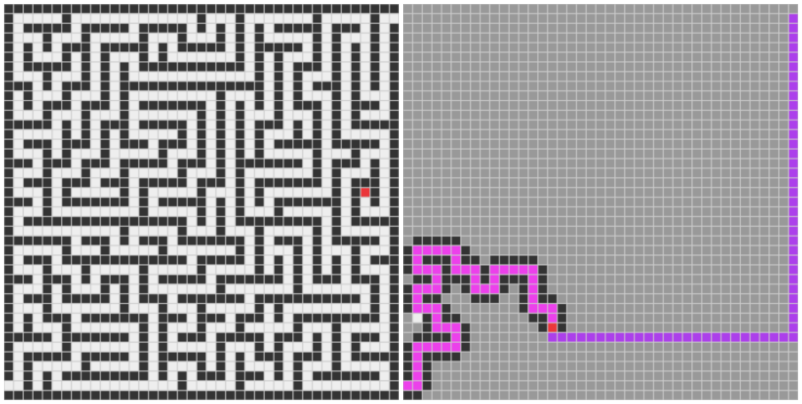

Figure 3: Following the path found by A*

Part 1

Read the chapter in your textbook on uninformed and informed (heuristic) search and then read the project description again. Make sure that you understand A* and the concepts of admissible and consistent h-values.

a Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north given that the agent does not know initially which cells are blocked.

Answer:

Since the agent does not initially know which cells are blocked in the maze, the agent performs the search algorithm on the adjacent cells. The order in which the agent inspects these cells corresponds with their f-values. Since each square has a g-value of 1, but the one on the right has an h-value of 2, while the others have a value of 4, this one has the lowest f-value. This process continues until a path is found, where the agent moves directly right. After moving and inspecting both cells above and to the right of the starting cell, the agent then concludes that the only possible move to reach its destination is the left.

b This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. Give a convincing argument that the agent in finite gridworlds indeed either reaches the target or discovers that this is impossible in finite time. Prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared.

Answer:

Let's set some assumptions, we know that A* uses an admissible heuristic which means it never overestimates the cost to reach the goal meaning the path it takes will be optimal. We can use a proof by contradiction and we assume that A* doesn't reach the target or it doesn't discover that the maze is impossible in finite time. This would occur if the robot is in an infinite loop, which would mean that the path taken is not optimal. This would lead to the heuristic not being admissible which contradicts what we know about A*. This infinite loop wouldn't be possible as it creates a new path every time it learns the position of a new wall based on its previous path. If the maze was impossible, then A* would first have to explore all the unblocked cells to check for a valid path. Eventually, all walls will be discovered by trying every path. The max moves for the algorithm to make would have a bound that is $O(n^2)$. In this case, n is the number of cells that the algorithm will explore to check everything. Then the times you can move to each of those cells would also be at most n, which means $n * n$ is the max number of moves before it deems a maze impossible which is in finite time.

Part 2

Repeated Forward A* needs to break ties to decide which cell to expand next if several cells have the same smallest f-value. It can either break ties in favor of cells with smaller g-values or in favor of cells with larger g-values. Implement and compare both versions of Repeated Forward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation.

[Hint: For the implementation part, priorities can be integers rather than pairs of integers. For example, you can use $c \times f(s) - g(s)$ as priorities to break ties in favor of cells with larger g-values, where c is a constant larger than the largest g-value of any generated cell. For the explanation part, consider which cells both versions of Repeated Forward A* expand for the example search problem from Figure 9.]

Answer:

When the Repeated Forward A* needs to break ties to decide which cell to expand, the algorithm should favor the larger g-values over smaller g-values. Since the g-value of each cell determines the length of the shortest path from the start state to state s, a larger g-value suggests that state s is farther from the start state, therefore, having a higher probability of moving closer to the goal, this is also backed by our results in Table. In other words, having a larger g-value would be likely to have a smaller h making it close to the end state.

In our observed data below (1), favoring low g values gives a much worse result, exploring more than 16 times as many states on average.

Results of Various A*					
Maze	Seed	F. high g	F. low g	Backward	F. adaptive
0	seed0	52,771	1,391,559	1,849,708	50,099
1	seed1	44,472	1,100,338	1,375,743	42,788
2	seed2	115,207	2,337,898	3,820,487	102,901
3	seed3	139,907	1,795,552	2,918,536	97,178
4	seed4	71,681	1,949,254	2,506,515	68,243
5	seed5	79,586	1,612,067	2,330,904	71,211
6	seed6	81,291	2,211,520	2,926,771	77,166
7	seed7	145,735	2,610,703	4,190,908	136,049
8	seed8	127,280	2,209,567	3,064,289	112,134
9	seed9	99,136	2,264,063	2,890,350	92,122
10	seed10	146,940	2,043,221	2,891,511	123,870
11	seed11	80,783	1,895,836	2,642,290	76,277
12	seed12	166,051	3,028,371	4,756,265	140,071
13	seed13	105,360	2,689,659	3,839,848	97,343
14	seed14	59,690	1,829,478	2,197,479	56,410
15	seed15	116,903	2,393,819	3,213,962	108,031
16	seed16	92,994	2,398,581	3,344,647	85,501
17	seed17	95,963	1,983,074	2,795,366	87,198
18	seed18	235,412	2,788,013	4,386,640	183,936
19	seed19	124,933	2,612,337	3,843,792	113,917
20	seed20	56,050	1,120,192	1,380,774	52,865
21	seed21	176,188	2,417,491	3,345,417	129,515
22	seed22	65,557	1,537,283	2,209,091	62,020
23	seed23	168,863	2,569,855	3,797,214	143,327
24	seed24	144,127	2,309,893	3,146,906	115,953
25	seed25	83,523	2,205,691	3,326,822	78,790
26	seed26	36,385	892,132	911,243	34,733
27	seed27	105,004	1,828,452	2,474,013	86,296
28	seed28	107,462	1,252,161	2,068,040	86,595
29	seed29	139,381	1,647,325	2,150,527	96,793
30	seed30	165,569	2,919,382	4,335,470	141,194
31	seed31	143,387	1,604,688	2,128,243	107,211
32	seed32	105,589	1,561,688	2,150,820	76,701
33	seed33	100,651	2,004,163	2,780,413	90,602
34	seed34	335,736	2,565,399	3,939,575	209,935
35	seed35	433,218	2,365,183	3,305,968	202,750
36	seed36	155,291	2,473,151	3,675,082	120,029
37	seed37	70,728	1,712,268	2,460,751	64,107
38	seed38	107,799	1,852,511	2,439,085	96,494
39	seed39	130,192	2,024,837	2,767,723	96,162
40	seed40	68,671	1,492,056	1,928,451	65,168
41	seed41	185,983	2,584,003	3,640,621	154,044
42	seed42	36,075	708,855	737,495	33,465
43	seed43	144,917	1,214,439	1,493,401	95,660
44	seed44	74,334	1,639,618	2,239,967	70,679
45	seed45	48,607	1,160,047	1,510,021	45,167
46	seed46	68,649	1,540,422	2,090,725	60,031
47	seed47	69,196	915,157	1,197,834	61,500
48	seed48	131,272	2,314,643	3,465,515	103,960
49	seed49	89,441	1,185,518	1,739,086	82,767
Average:		118,598.8	1,935,268.26	2,732,446.08	95,739.16

Table 1: Table of states explored for various A*

Part 3

Implement and compare Repeated Forward A* and Repeated Backward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both versions of Repeated A* should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly.

Answer:

While both search algorithms depend on the maze generation, Repeated Forward A* is frequently more optimal in both runtime and number of expanded cells. The reason for this is that since we are tie-breaking based on higher g values, forward A* starts going straight towards the goal, as long as it is within a region with the same f-values. The f-values for the entire rectangle between the current position and the goal are all the same since moving one square closer will also reduce the heuristic by one. Since there are no more walls past where the agent has explored in forward A*, this allows the search to beeline straight to the goal and end that search most of the time.

However, when running backward A*, it is much more common for the goal to be slightly occluded by a few walls since that is where the agent is exploring. The rectangle of the same f-values still applies, except when there is an obstacle to get around, that requires A* to look a little past the obstacle, which it would not do until it explores all the squares of lower f-value first. This makes the backward A* waste a lot of time exploring the unknown part of the maze.

Part 4

The project argues that "the Manhattan distances are consistent in gridworlds in which the agent can move only in the four main compass directions." Prove that this is indeed the case. Furthermore, it is argued that "The h-values $h_{new}(s)$ ... are not only admissible but also consistent." Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

Answer:

To prove that Manhattan distances are consistent in gridworlds, we must prove that the heuristic will never overestimate the distance from a state to the goal. In other words, it cannot be possible to reach a goal from a state in fewer moves than the Manhattan distance heuristic.

We will prove this by contradiction. Suppose that, from a state s, we have a heuristic value $h(s)$. Since the heuristic is the Manhattan distance, it must be an integer quantity of squares between s and the goal state. If there were ever a path that used fewer moves than the number of squares in the Manhattan distance, this would be a contradiction since each move costs exactly one square. Therefore, it's impossible to have a path to the end shorter than the heuristic says, so it is admissible.

Next, we will prove that the Manhattan distance heuristic is consistent. To do this, the following inequality must hold:

$$h(n) \leq c(n, a, n') + h(n') \quad \forall(n, a, n')$$

where n is a node, n' is a next node, and a is the action that is taken to get from n to n'. In other words, the heuristic must not decrease by more than the cost of the action taken when going from n to n'.

In our case, there are at most 4 actions for any given state: left, right, up, or down. Each of these actions has a cost of 1. After moving to a new state, this will either increase or decrease the horizontal/vertical distance by 1. If we go towards the goal, then $h(n')$ will be 1 less, and if we move away from the goal, then $h(n')$ will be 1 more. In both cases, the inequality holds, because the maximum decrease of 1 in the next node's heuristic value is always offset by the cost of 1 to go from the current state to the next one.

Next, we will prove that Adaptive A* leaves consistent h-values as consistent even if action costs can increase. For Adaptive A*, the h-values update by assigning:

$$h(n) = g(n_{goal}) - g(n)$$

for all states s during the A* search algorithm. Let $h_{new}(n)$ represent the h-values after the updates. To prove that $h_{new}(n)$ is consistent, we need to prove that the following inequality holds:

$$1) \quad h(n) \leq c(n, a, n') + h(n').$$

We know trivially that the following inequality holds:

$$h_{new}(n) \leq h_{new}(n).$$

Next, substitute the right side:

$$h_{new}(n) \leq g(n_{goal}) - g(n).$$

By definition, we know that $g(n') = g(n) + c(n, a, n')$, so we substitute that in as well:

$$h_{new}(n) \leq g(n_{goal}) - (g(n') - c(n, a, n'))$$
$$h_{new}(n) \leq c(n, a, n') + g(n_{goal}) - g(n')$$

Thus, we have proven the inequality:

$$h_{new}(n) \leq c(n, a, n') + h_{new}(n').$$

This proves that the new heuristic is consistent when comparing a node and its successor. However, this is only true when both nodes were explored in the previous iteration of A*, which uses the new heuristic. Two cases are yet to be proven: starting from a node that was explored previously and moving to one that was not, or starting from a node that was not explored and moving to one that was. This means we also have to prove the following two inequalities:

$$2) \quad h_{new}(n) \leq c(n, a, n') + h(n')$$

$$3) \quad h(n) \leq c(n, a, n') + h_{new}(n')$$

To prove the first inequality, we know that we are going to a node that wasn't explored, so that means that its f value was larger than the g value of the goal node. Therefore:

$$g(n_{goal}) \leq f(n')$$
$$g(n_{goal}) \leq g(n') + h(n')$$
$$g(n_{goal}) \leq g(n) + c(n, a, n') + h(n')$$
$$g(n_{goal}) - g(n) \leq c(n, a, n') + h(n')$$
$$h_{new}(n) \leq c(n, a, n') + h(n')$$

To prove the second inequality, we use the inequality $h(n) \leq h_{new}(n)$ (This was provided to us from the assignment document, and it's because we know that h is admissible, so $h \leq g(n_{goal}) - g(n)$). Then we know:

$$h(n) \leq h_{new}(n) \leq c(n, a, n') + h_{new}(n')$$

$$h(n) \leq c(n, a, n') + h_{new}(n')$$

Now that consistency was proven for Adaptive A* with constant action costs, we can show that it is also consistent if there are increasing action costs because only the right side of all of the 3 inequalities would increase (since it is only affecting the cost from n to n'), leaving the inequality valid and the h-values consistent.

Part 5

Implement and compare Repeated Forward A* and Adaptive A* with respect to their runtime. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both search algorithms should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly.

Answer:

Under the same methods of breaking ties among cells, Adaptive A* is faster than Repeated Forward A*. These observations come from the optimality Adaptive A* holds over Repeated Forward A*. Adaptive A* updates the h-values of the expanded cells, thus focusing on the scope of future A* searches. It makes sure when rechecking nodes previously expanded, the algorithm will search those fewer as the heuristic would increase, giving the algorithm fewer nodes to expand. A more in-depth, statistical result is displayed in Table 1.

For instance, the average number of states explored in Repeated Forward A* is 118,598.8. On the other hand, the average number of states explored in Adaptive A* is 95,739.16. Just by comparing these two numbers at their face value, the Adaptive A* searches fewer states on average compared to Repeated Forward A*, lessening its runtime. These observations extend to each seeded maze where the same tread is demonstrated.

Part 6

Performance differences between two search algorithms can be systematic in nature or only due to sampling noise (= bias exhibited by the selected test cases since the number of test cases is always limited). One can use statistical hypothesis tests to determine whether they are systematic in nature. Read up on statistical hypothesis tests (for example, in Cohen, Empirical Methods for Artificial Intelligence, MIT Press, 1995) and then describe for one of the experimental questions above exactly how a statistical hypothesis test could be performed. You do not need to implement anything for this question, you should only precisely describe how such a test could be performed.

Answer:

Under the formatting of statistical hypothesis testing, two initial hypotheses are formed to verify whether the performance of Repeated Forward A* compared to Adaptive A* is systematic. Assuming the original conjecture is the performance is systematic, the null hypothesis in this experiment is as follows: "The performance differences between Repeated Forward A* and Adaptive A* come from sampling noise." Thus, the alternative hypothesis is "The performance differences between Repeated Forward A* and Adaptive A* are systematic in nature."

To properly investigate the claim, the experiment should ensure control variables, such as algorithm implementation, methods of dealing with ties, and the programming environment, are maintained while broadening the scope to include different machines. Under these conditions, the tester will run experiments across an expansive amount of mazes before coming to a conclusion based on analysis of the data. The way the test is run is to specify the null and alternative hypotheses, then decide upon a significance level, which is the degree to which we accept or reject the null, which is usually around 5% meaning 95% of the time you get the same result. Then there is the p which is the actual observed probability assuming the null is true, and if that is less than the significance level defined you should reject the null hypothesis. To show this you would calculate a test statistic based on observed data, and compare it to the critical value, usually p to reject or accept. This test statistic can be calculated by using several tests, in our case our sample is small so using a T-test would be appropriate. Once you get the T-statistic compare it to the critical value.