

Part 1

(a)

Because when the agent does not know initially which cells are blocked, it will calculate the h values and g values to get f values in a star algorithm to find the shortest path by selecting smallest f value. The northern cell has larger manhattan distance than the eastern cell, and they have same g value that is from $A.g_value + 1$, so A moves to eastern cell that has smaller f value.

(b)

In our algorithm, every adjacent square that is not placed in the close_list is traversed before each run, and the agent keeps running among the adjacent cells to find the path, so it ensures that we can reach the end point from the starting point if there's indeed a path. As it's not allowed to re-enter the visited cells in close list, so every time the a star restart when it runs into a block, the number of moves of the agent could not be over the number of grids in every gridworld. Each time we encounter a block we rerun the algorithm at the blocked position, in which case the number of times we rerun should be less than or equal to the number of unblocked cells. Then the number of expanded cells in each traversal run should be less than the number of unblocked cells, so 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.

set n = unblocked cells, x = repeated times of A star, y = grid put into close_list every time

$x \leq n$, $y \leq n$, so $x*y \leq n^2$

Part 2

After implemented two versions of repeated forward A star to solve the maze:

The 5*5 gridworld from the question (Figure 9 in assignment) that has 25 unblocked grids could provide clear illustration. When the repeated forward A star prefer smaller g value, it will go through all the cells in this gridworld. When the algorithm prefer larger g, it will go through the side of this square to reach the goal directly.

When there's block in the grid world, the repeated forward A star algorithm that prefers larger g value seems to still run less number of grids to reach goal.

Figure 1 is the repeated forward A star that prefers larger g value, and Figure 2 is the one prefers smaller g value.

Figure 1

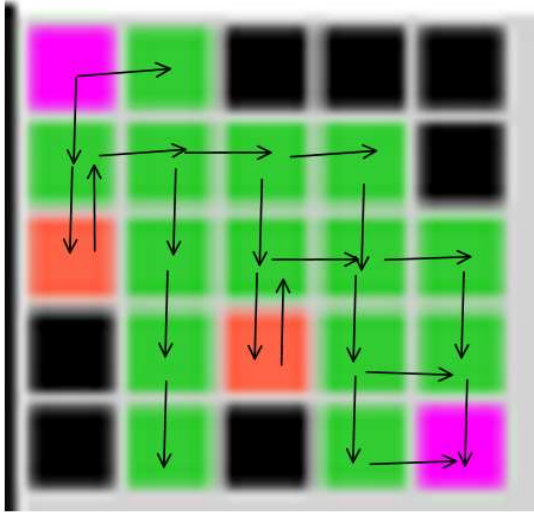
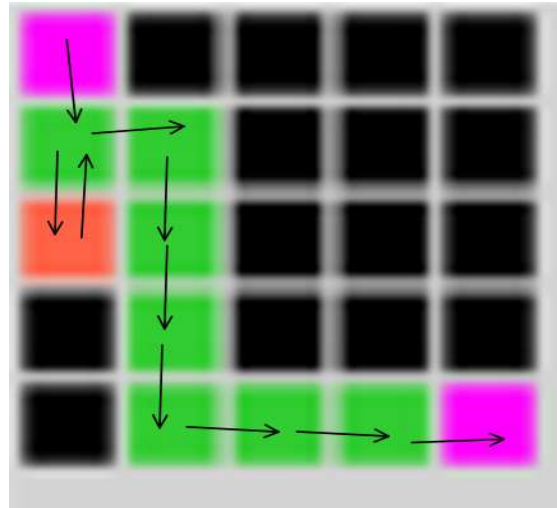


Figure 2



Part 3

By implement repeated forward A star and repeated backward A star on 40*40 example gridworld, the results are shown below:

Figure 3

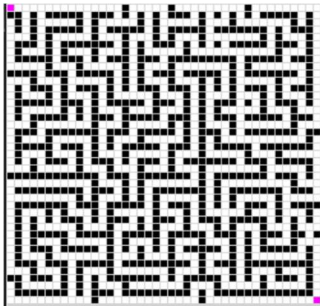


Figure 4

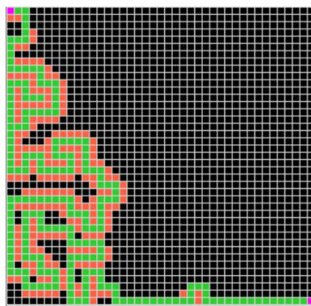


Figure 5

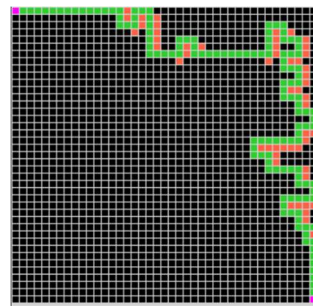


Figure 3 is the maze for presentation purpose. Figure 4 is the repeated forward A star to solve this maze, and Figure 5 is the repeated backward A star to solve this maze. They don't have very obvious differences on running, but have different expanded cells.

Whether backward A* will be faster than forward A* depends on whether the possible path around the end is more limited than the start, such that backward A* will have less cells to expand.

Part 4

(a)

$$H_s = |X_s - X_g| + |Y_s - Y_g|$$

By Chengyu Liang

Xg and Yg are consistent.

Agent moving in four directions in grid world.

$Hs' = Hs + C(s1, a, s2)$, where $C(s1, a, s2)$ is also consistent in each move, because it can only move to four directions and cost is 1.

Hence we can get the Manhattan distance of every other cells, and it's consistent.

(b)

In the original version, the Manhattan distance for every move should be limited to:

$c \geq g(s2) - g(s1)$, where c is the cost of each move.

From the question, we know that $h_{new}(s1) = g(s_{goal}) - g(s1) = (g(s_{goal}) - g(s2)) - (g(s1) - g(s2)) = h_{new}(s2) + (g(s2) - g(s1)) \leq h_{new}(s2) + c$

And the h_{new} value is always in range from $g_{goal} - g_{init_start}$ to $(g_{goal} - g_{goal} = 0)$

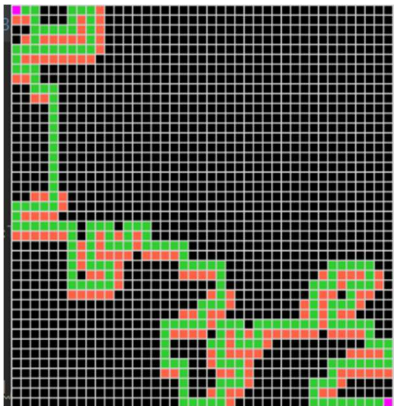
So $h_{new}(s)$ is consistent when cost increase.

Part 5

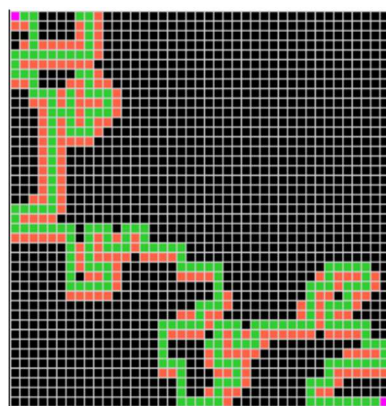


For better understanding, I choose this gridworld to show the difference between adaptive A star and repeated forward A star.

Adaptive A star



Repeated Forward A star



Running Time: 17.307209014892578 seconds Running Time: 24.489699363708496 seconds

By Chengyu Liang

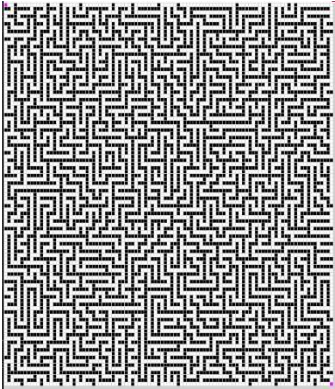

It is quite obvious that Adaptive A star passes through fewer squares, especially in the area between (3, 10) and (10, 25). Compared to Repeated Forward A star, Adaptive A star is able to reach a good position by traversing fewer cells. Comparing with their running time, Adaptive has clear advantage on running time and less cells. As the h value of Adaptive A star is gained from $g(\text{goal}) - g(s)$ and keep updating for expanded cells, it makes the new h value more accurate to be a higher consistent value, so that the agent can go to a cell with lower $g+h$ to avoid some cells every time running A star.

Part 6

To perform this test between Adaptive A star and Repeated Forward A star, the method I would use is to set the independent and dependent variables, and then observe the different performance of the two different methods to solve the same maze, mainly in terms of the nodes expanded and the time spent.

Example:

To solve this maze: Here's a 101*101 maze

	Repeated Forward A*	Adaptive A*
		
Running time:	420.97428464889526 seconds	95.73347067832947 seconds
Expanded cells:	22680	3697

By doing so, we can know that there's an obvious difference on the efficiency for the two algorithm to solve this maze, and Adaptive A* provide great choices to avoid waste of time to reach the goal. Since that there are too many block cells in my generated maps and only one feasible solution made by the maze generator, and it's really time consuming for going through the 100*100 maze with one solution, I believe that the visual system cause additional time consuming for me to reach the goal. But for the construction of my adaptive A star, it serves for the visualization too much and may cause some extra time consumption.