AI Assignment 1

Covid Escape Game
15 March 2021

Introduction -

The main aim of this task was to implement some efficient algorithm to help our *actor* reach the *home* by avoiding the *covid* prone area by either using a *mask* or visiting the *doctor* using the shortest path from the initial point to the home. The map is made up of N×N blocks which consist of two covid zones infecting several cells around them, a doctor, and a mask.

To solve this task we can use either of the two search algorithms namely *Backtracking* and *A**.

Backtracking -

Backtracking is a very naive and inefficient algorithm as DFS will start to look at all the neighbors and try to visit them in order to check every cell and a possible path to reach the destination. Although several improvements can be done in order to prevent such inefficiency.

If a map having 9 × 9 blocks with 2 covid cells, 1 doctor, and 1 mask is taken for illustration, then the length of the shortest path has to be less than 18 blocks. Therefore, if the path to the cell that we reached exceeds this value of 18 blocks, then it is a bad path with absolutely no optimization, so the actor will preferably choose another cell to go to.

Also, when the actor reaches the home for the first time, the length of that path is saved, so if another path exceeds this value then it is also not the optimal way, and the actor will choose another cell to go to.

Since the first found path may not be the shortest, we need to find all possible routes and find the shortest between them. This is a tedious job and consumes a lot of CPU time.

The basis for this implementation is as follows: -

1. At any arbitrary time, if the current cell is outside of the boundaries, or at a COVID infected cell (when not protected yet) then, this route can't be a solution and a new route must be chosen.
2. One optimization that I thought of adding to make the code run faster is to backtrack at any current route if the number of steps remaining is less than the diagonal distance to the home from the current destination. (I did not add this optimization as it was making backtracking run faster than A*)
3. If the coordinates of the current cell are equal to those of the home then it's a valid solution.

It's worth mentioning that this algorithm is *not affected* by the variant change. As DFS chooses one of all the possible neighbors at a time, knowing that some possible move will lead to having the current cell as a neighbor for a COVID infected cell won't change the way this algorithm works.

A* -

A* is a BFS based algorithm and it does not consider all the possible routes but tries to go directly to the home. This algorithm can be either implemented using two lists namely the Open list and Closed list or by having a priority queue.

1. It looks at all the neighbors of the current cell in the route and adds all of them to a priority queue. The queue is then sorted by the shortest diagonal length. Each cell has a parent cell(previously visited cell). When the *home* agent reaches its destination, we can trace the parent cells to find the entire route.

2. The open list contains all the cells that the actor can visit and their parent. Closed list contains all visited cells and their parents. Initially, it is empty. Also for each cell, the algorithm calculates special values: F = G + H, where G is Chebyshev's distance from the initial to the current cell, H is Chebyshev's distance from home to the current cell and, F is the sum of G and H.

3. It starts from the initial point, adds this cell into the Open List - parent for the initial cell is that cell itself, and G, H, and F values are 0. Then the following actions are repeated until the actor reaches home:

    a. Picks the cell with the lowest F.
    b. Adds this cell and its parent into the closed list.
    c. For each adjacent cell that is not in the closed list: calculates G, H, and F •
    d. If this cell is not in the open list just then add this cell, its parent, and values into the open list.
    e. Else: if the $G_{existed} > G_{current}$ then the algorithm will replace the old cell with the new and puts new G, H, F values.
    f. When the actor reaches home, the current path is saved and the algorithm stops.

The base for this implementation is as follows: -

1. At any arbitrary time, if the current cell is outside of the boundaries, or at a COVID infected cell (when not protected yet) then, this route can't be a solution and a new route must be chosen.
2. If the coordinates of the current cell are equal to those of the home then it's a valid solution.

It's worth mentioning that this algorithm is *affected* by the variant change.

PEAS -

First, the *agent* type is an *actor* who performs actions to safely reach the home by choosing the shortest optimal path. Second, the *performance measure* is that the actor should reach the home without coming in contact with the covid prone area and winning the game. Third, the *environment* is an N × N map with covid blocks that cause infected cells around them, a doctor, and a mask to protect the actor from getting infected. We have no restriction on the actor's movement as the actor can freely move in this environment in all directions but only one cell forward at a time. Fourth, *actuators* are legs that are used to go home accompanied by the doctor and masks that help the actor to stay safe. Fifth, the actor has a *sensor* that is a covid predictor which helps him in predicting the presence of a covid cell and avoiding it.

The environment properties can be summarised as follows - First, the environment is *partially observable* because the actor can feel the covid only one cell before and has no idea about the doctor or mask. Second, the environment has *multiple agents* because apart from the actor there are covid cells with infected zones where the actor cannot go without protection, also there are cells with a doctor or mask where an actor can get protection from the covid. Third, the environment is *deterministic* because all agents except the actor are stationary and do not change their properties, so the state of the environment is constant. Fourth, the environment is *sequential* because the next step of the actor depends on the step that was made before. Fifth, the environment is *static* because nothing happens in the environment while the actor is choosing the next step. Sixth, the environment is *discrete* because the actor moves only one cell forward and after each step carefully chooses the next step. Finally, the environment is *known* because the actor knows the size of the map and the infected cells.

Comparison of algorithms -

Using *Null-Hypothesis* we can say that Backtracking does not change in the given 2 variants.
For comparing the other algorithms let us use the *Student-T* test. To do so, we need to run the algorithms several times and track the time for each algorithm for each variant. Then finding the mean value and standard deviation for each variant. Lastly, comparing the numerical results.

Making a *null hypothesis* H_0 that all variants for searching the shortest path are the same.

For this experiment, the position of the covid cells, doctor, and mask will be generated randomly.

Sr.no	Backtracking Time(ms)	A* Time(ms)	A* v.2 Time(ms)
1.	739	51.6	31,3
2.	23.7	10.9	13.7
3.	949	16.4	8.1

4.	312	29.3	17,8
5.	283	38.3	26.2
6.	81.5	33.4	22.9
7.	59.3	12	6.3
8.	193	31.3	30
9.	39.8	11.5	9
10.	26	10	7.2

Table for mean, standard deviation, and variance of time taken by the different algorithms.

Terms	Backtracking	A*	A* v.2
Mean	263.295	24.47	18.15
Standard Deviation	323.518	14.373	9.941
Variance(s^2)	104664.28	206.600	98.836

As we can see, the time of execution differs for the 1st variant and the 2nd variant of A*because the 2nd variant algorithm does extra computations but it does not affect the search of the shortest path.

For Student T-test we need to calculate the t-value for each pair of variants:
Since each algorithm has 10 samples, the degree of freedom is df = 10 + 10 − 2 = 18.

Also, let us make the probability of not rejecting our Null-Hypothesis equals p = 0.05. According to T-table, for p = 0.05 and df = 18 critical value approximately around 12.

1. Comparing Backtracking and A* - therefore t-value = 13.27, as this is bigger than the critical value, we reject the null hypothesis that these algorithms are the same.
2. Comparing Backtracking and A* v.2 - therefore t-value = 13.63, as this is bigger than the critical value, we reject the null hypothesis that these algorithms are the same.
3. Comparing A*and A* v.2 - therefore t-value = 6.509, as this is smaller than the critical value, we do not reject the null hypothesis that these algorithms are the same.

Impossible Maps -

A map is considered to be impossible if the actor cannot safely reach home, i.e. the actor is blocked by infected zones and cannot visit a cell with a doctor or mask, so cannot leave the blocked zone.

1.

					D			
X	X	X						
X	C	X				H		
X	X	X						
		X	X	X				
		X	C	X				
A		X	X	X				M

2.

								H
			D					
X	X	X						
X	C	X						
X	X	X						
		X	X	X				
		X	C	X				
A		X	X	X				M

3.

				X	X	X	D	H
				X	C	X		M
				X	X	X	X	X
						X	C	X
						X	X	X
A								

4.

				X	X	X	D	H
				X	C	X		M
				X	X	X		
						X	X	X
						X	C	X
						X	X	X
A								

5. A map with the following coordinates is unsolvable with backtracking

Covid 1: 8,6
Covid 2: 7,8
Home: 9,9
Mask: 9,1
Doctor: 9,8

Actually, all of these examples are almost the same, i.e. the actor is separated from home, doctor, and mask by infected cells but the time of execution differs.

In the first example program quickly checks all possible ways and finds that there is no way home, however, in the second variant checking all possible ways takes much more time.

Conclusions -

The comparison of 2 algorithms for searching the shortest path was done and by statistical analysis using the Student T-test, we can conclude that A* is better almost in all cases than backtracking.

However, in some cases, A* may give one step longer path because it has several adjacent nodes to choose from and it may choose a cell that is not the closest to the home.

Additionals -

Apart from the normal Prolog code, I also implemented a beautiful GUI for making the evaluation easier. I used thecompose library supported by Kotlin to parse the prolog code and take its output query as an input for my Kotlin program.

During this process, I realized that prolog is faster than C++ and has a flaw with memory caching as it does not clear the previous threads. This prohibited me from running the code again and again as prolog does not allow multi-threading.