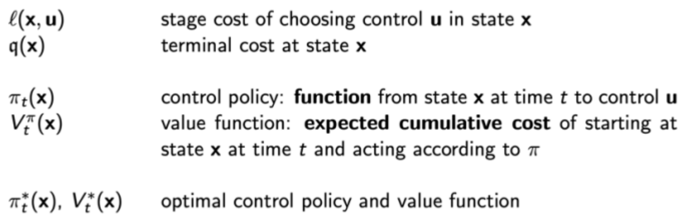
Dynamic Programming
: Autonomous navigation in a 'Door & Key' environment

Orish Jindal
Department of Electrical Computer Engineering
University of California, San Diego
ojindal@ucsd.edu

I. Introduction

Path planning is the first step for navigation of an Autonomous vehicle. This is done by first deciding the states that a robot needs to visit throughout the whole motion and then ossifying the optimal set of control inputs which leads our robot to the respective states. The most popular way to do it is by using one of the many popular dynamic programming algorithms such as Label Correcting, A*, Dijkstra etc. In this paper, we will focus on the need for using dynamic programming, the algorithm used in our case, technicalities of the algorithm along with the results that were obtained on a particular problem of autonomous navigation in a door & key environment.

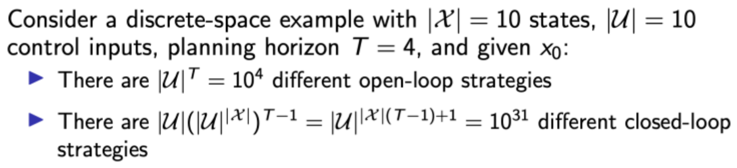
Usually, deterministic path planning on discretized time works in this way: a robot knows the coordinates of its start position and then given the coordinated of its destination, it decides the intermediate states while moving from start to end. Those intermediate states have an associated cost called stage cost that comes to play while transitioning from one state to the other. A value function is then calculated by adding the stage costs at each discretized time stamp in order to compare between different possible combinations of transition states which are leading the robot to the desired destination. The optimal value function is the minimum possible value function that represents a particular set of states visited by the robot at respective timestamps. The control policy associated with the optimal value function is the optimal control policy.



$\ell(\mathbf{x}, \mathbf{u})$	stage cost of choosing control **u** in state **x**
$q(\mathbf{x})$	terminal cost at state **x**
$\pi_t(\mathbf{x})$	control policy: **function** from state **x** at time t to control **u**
$V_t^\pi(\mathbf{x})$	value function: **expected cumulative cost** of starting at state **x** at time t and acting according to π
$\pi_t^*(\mathbf{x})$, $V_t^*(\mathbf{x})$	optimal control policy and value function

II. Problem Formulation

The most basic approach to plan a path would be to calculate the value function for all possible combinations of the states in state space and then compare them for the minimum and decide the optimal combination of states and hence control inputs to fix a control policy. This approach is very easy to understand but is most inefficient. In a problem with large time horizon and state space, obtaining results this algorithm will be almost impossible as the number of calculations will be very high and hence will take a huge amount of time to calculate the optimum policy. This will lead to a huge unwanted delay and the system will be considered a failure. For example:



Consider a discrete-space example with $|\mathcal{X}| = 10$ states, $|\mathcal{U}| = 10$ control inputs, planning horizon $T = 4$, and given x_0:

- There are $|\mathcal{U}|^T = 10^4$ different open-loop strategies
- There are $|\mathcal{U}|(|\mathcal{U}|^{|\mathcal{X}|})^{T-1} = |\mathcal{U}|^{|\mathcal{X}|(T-1)+1} = 10^{31}$ different closed-loop strategies

Now in order to use a to better algorithm for the obtaining the optimal control policy, we need formulate such kind of problems with Markov Decision process. In our example of a n*n grid with random walls, doors, key position and a goal, the states x_t and control inputs $u_t$will become nodes in the Markov Chain.

In the problem discussed in this paper,

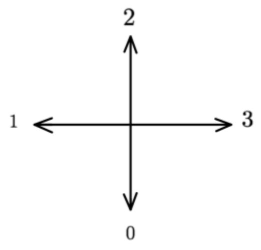
X (discrete state space), $x_t \in$ X will be a 3D vector with first entry representing the x-coordinate, second represents the y-coordinate and the third represents the direction in which the agent (robot) is facing. A custom convention has been used for representing four possible directions in this problem which is as follows (also shown in figure below):

Facing up (-y by given convention): 2

Facing down (+y by given convention): 0

Facing left (-x by given convention): 1

Facing right (+x by given convention): 3



U (control space), $u_t \in$ U will have 5 actions:

MF (move forward), TR (turn right), TL (turn left), PK (pick key), UD (unlock door).

f (Motion model), x_{t+1} = f (x_t, u_t)

To roughly visualize the motion model

{

[u, v, (dir+1) % 4] = f ([u, v, dir], TR)

[u, v, (4+dir-1) % 4] = f ([u, v, dir], TL)

[u, v+1, 0] = f ([u, v, 0], MF), if x_{t+1} != door

[u-1, v, 1] = f ([u, v, 1], MF), if x_{t+1} != door

[u, v-1, 2] = f ([u, v, 2], MF), if x_{t+1} != door

[u+1, v, 3] = f ([u, v, 3], MF), if x_{t+1} != door

Pick Key = f (x_t, PK), if facing a key

Unlock Door = f (x_t, UD), if facing a door and carrying a key

}

Initial state, $\boldsymbol{x_0}$ = given initial coordinates and direction

Planning horizon, T = maximum possible number of steps or actions,

T = (height of grid * width of grid * number of directions) + pick key (this action is performed max one time in my algorithm) + unlock door (this action is performed max one time in my algorithm).

Hence,

T = (height*width*4) + 2

Stage cost: l $(\boldsymbol{x}, \boldsymbol{u})$, Terminal cost: q $(\boldsymbol{x})$: for all t (τ = destination)

$$\ell(x,u) := \begin{cases} 0 & \text{if } x = \tau \\ \perp & \text{otherwise} \end{cases} \qquad q(x) := \begin{cases} 0 & \text{if } x = \tau \\ \infty & \text{otherwise} \end{cases}$$

Next step would be to use an appropriate algorithm exploiting the assumptions of Markov Decision process which is faster than the brute force.

III. Technical Approach: Label Correcting

Dynamic programming is one of the most popular techniques used to solve problems like the one stated above. It works on the principal of avoiding unnecessary calculations sometimes by narrowing down the state space by eluding the states that are clearly out of scope of the current protocol for the robot or by storing the calculated values for subproblems time to time in order to reuse them to avoid calculating the same value again if the model requires so in future. It offers a variety of algorithms that are functionally almost same but have different ease of feasibilities with different category of problems.

One such algorithm that is used in the problem discussed in this paper called the Label Correcting Algorithm. It is mostly used in path planning problems, and it works on the principal of avoiding the seemingly impossible paths and thus reduces the required number of calculations. This accounts for the efficiency of this algorithm. This algorithm algorithms prioritizes the visited nodes 'i' using the cost-to-arrive values V_t^F(i).

Key ideas:

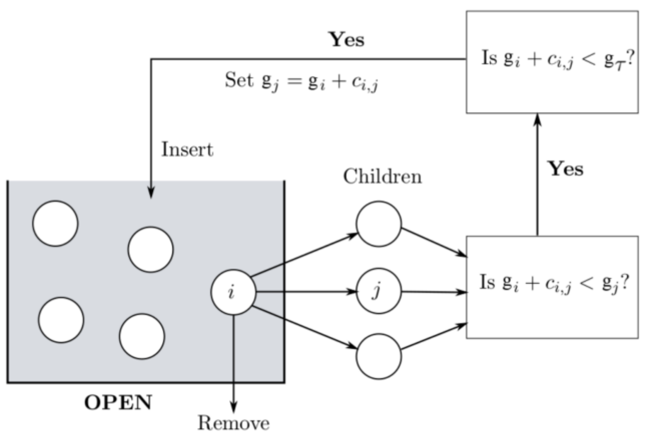
- Label g_j = g_i + c_{ij} estimate of the optimal cost from s to each visited node i ∈ V
- Each time g_i is reduced, the labels g_j of the children of i are corrected: g_j = g_i + c_{ij}
- OPEN: set of nodes that can potentially be part of the shortest path to τ

The algorithm is as follows:

Algorithm: Label Correcting Algorithm

```
1: OPEN ← {s}, $g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
2: while OPEN is not empty do
3:     Remove i from OPEN
4:     for $j \in$ Children(i) do
5:         if $(g_i + c_{ij}) < g_j$
6:             $g_j = g_i + c_{ij}$
7:             Parent(j) = i
8:             if $j \neq \tau$ then
9:                 OPEN = OPEN ∪{j}
```

If there exists at least one finite cost path from s to τ, then the Label Correcting (LC) algorithm terminates with g_τ = dist (s, τ), the shortest path length from s to τ . Otherwise, the LC algorithm terminates with $g_\tau = \infty$. Visually:


Yes
Is $g_i + c_{i,j} < g_\tau$?
Set $g_j = g_i + c_{i,j}$
Insert
Children
Yes
i
j
Is $g_i + c_{i,j} < g_j$?
OPEN
Remove

In the project discussed in this paper, the above shown algorithm is implemented in python. All the elements of the state space were represented as nodes for compatibility to the LC algorithm.

First, a function is written which gives the children of the argument node. Although there are 5 possible actions at each state but two of them (pick key: PK, unlock door: UD) are used at max. one time only and that too at the known states respectively.

So, the effective children of a node are as follows:

1. Turn right: TR
2. Turn left: TL

If the forward position is not blocked by a wall or closed door:

3. Move forward: MF

Value function (V) is initiated as a 3D matrix of size V.shape = (height, width, 4) and g_{ijk} will be the entries. Initially, g_{ijk} = infinity except the starting position which is 0 (where i, j, k represent the x-coord, y-coord, and direction of the start position.)

The algorithm is then implemented as it is which gradually fills and rewrites (if needed) the dictionary 'Parent' from which we can backtrack the best parent of each node starting from the end node and hence, the sequence of control inputs that represents the optimal control policy.

This problem can be broken down into two sections:

Section 1: The robot searches for an optimal path from start position to end position considering the door as a wall hence, removing the element of picking key and unlocking door.

Section 2: The robot searches for optimal control sequence from start position to all the nodes where it can pick key(a function is defined specifically for calculating the states from which the key can be picked or the door can be opened) and the key is picked [PK], then same thing from the position where key is picked to the positions where the door can be opened and then the door is opened [UD]. Further, door to goal position. A combined optimal control sequence is found adding all five actions.

Now, the costs of both paths are compared and the sequence with lower cost will give us the best possible control sequence (optimal).

For problem in **part B** of the project, there are two doors instead of one. There are three possible cases with combinations as follows:

- Both doors closed
- Both open
- One open one closed

The code used for part A is modular and is made to handle the randomness of part B with small modifications (to incorporate two doors) in the helper functions as:

case 1: In part A, we were using the approaching nodes (from which the door can be opened) of only one door in our calculation. This time, we just need to take union of the approaching nodes of both the doors and the rest will follow as the code is modular.

case 2: Just considered that there is no door on the given environment.

case 3: Same as part A, just considered the open door as an empty cell (unblocked node).

IV. RESULTS

For part A:

The optimal control sequences that were obtained for all the given environments of part A, were used to plot the gif showing movement of the robot based on the given environment. All these gifs were saved in a folder and submitted in the zip file containing code on gradescope.

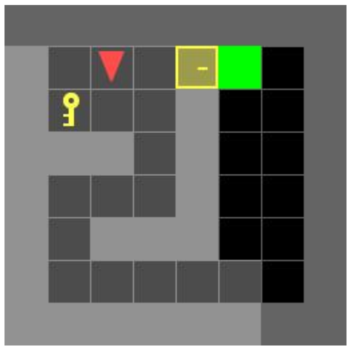
For part B:

2 to 3 examples of random environments for each of the three cases (both door closed, both open, one open one closed) were saved in a folder and submitted in the zip file containing code on gradescope.

Discussion on a few examples of the results:

Part A:

For the example "doorkey-8x8-shortcut":

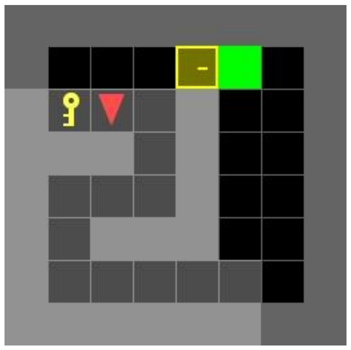


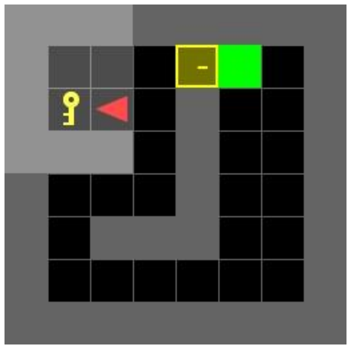
This is the given problem and the robot is facing downward. Now, there are two possible paths:

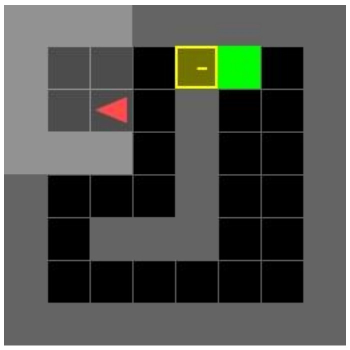
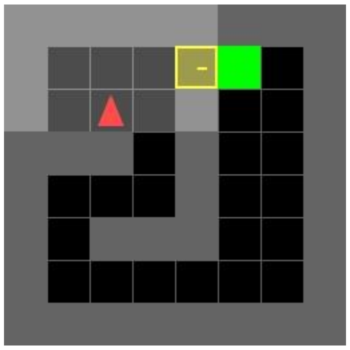
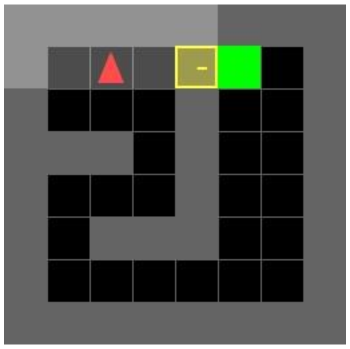
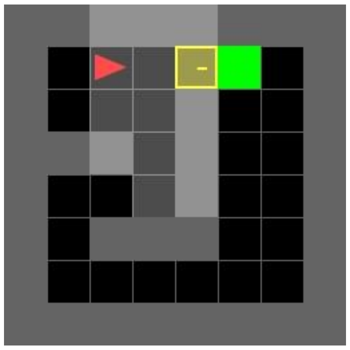
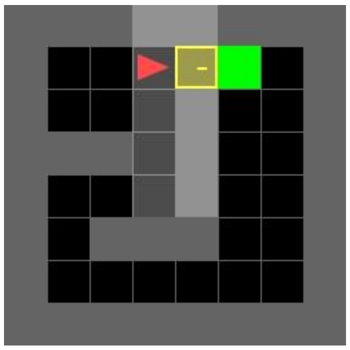
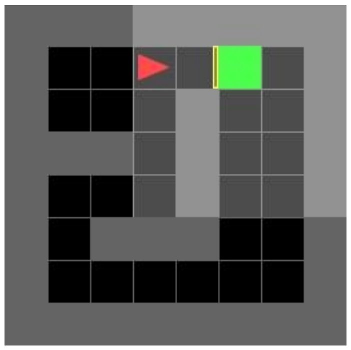
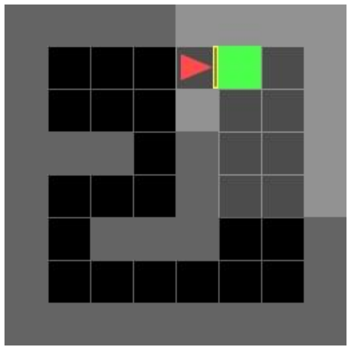
Path 1 – By completely evading the door and reaching the goal by going through the bottom boundary of the grid.

Path 2- By taking key, opening door and then reaching goal.

Here is what our model suggested (pictures in sequence):

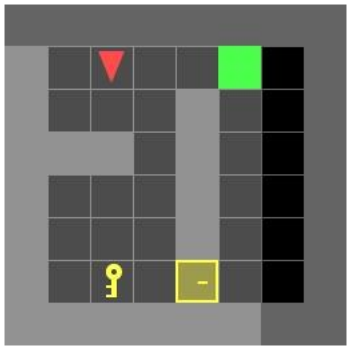
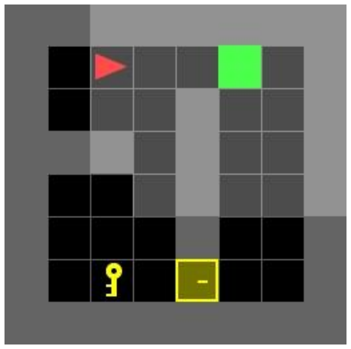


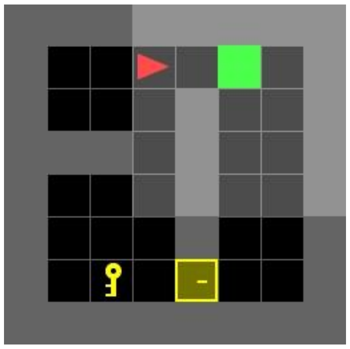


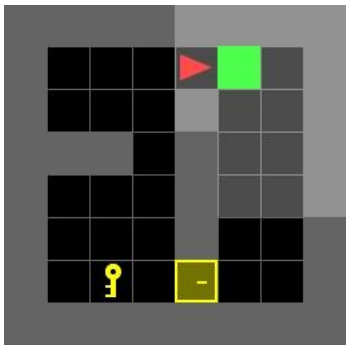








Our robot took the Path 2 (described above) as it is the shortest among those.

Also, here is the other example “doorkey-8x8-direct”:



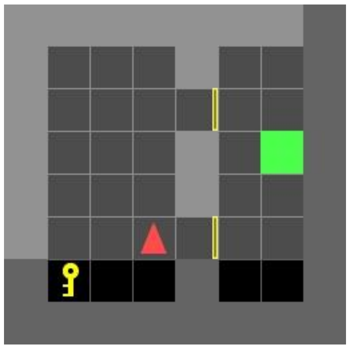


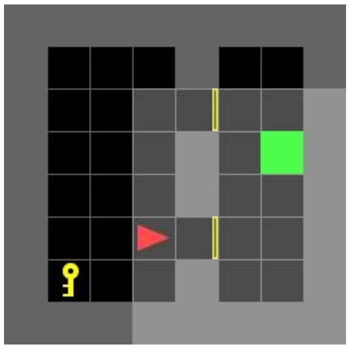
This time, the robot took a direct path because it was shortest.

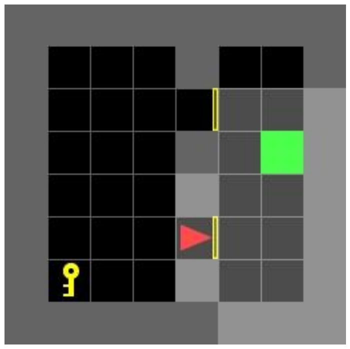
Part B:

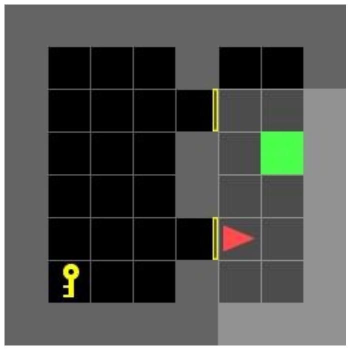
One example each of the three variations in part B.

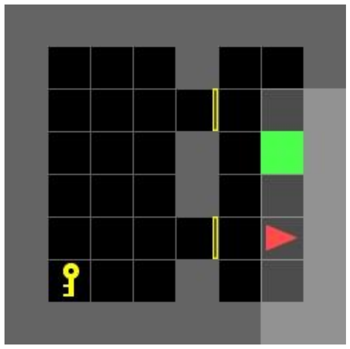
1) *Two closed doors:*

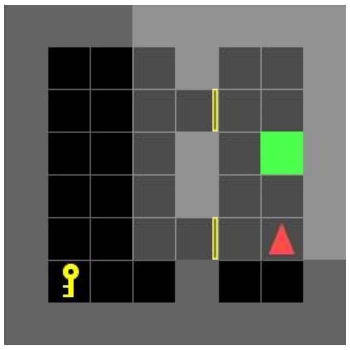


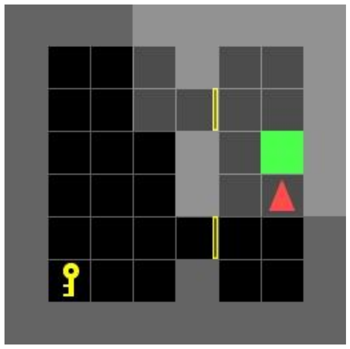




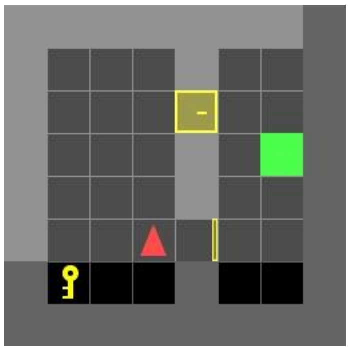


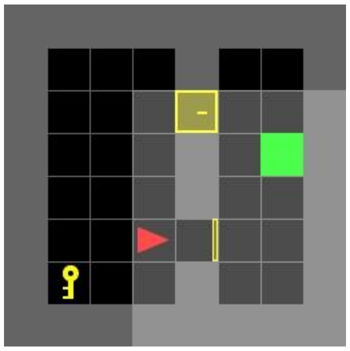


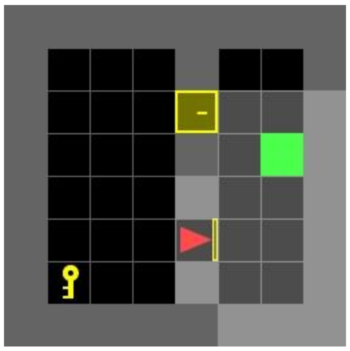


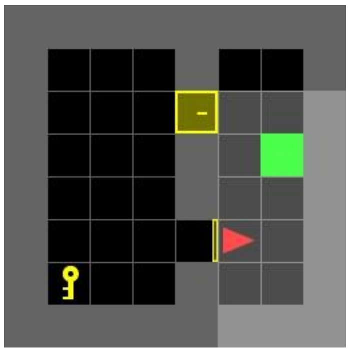


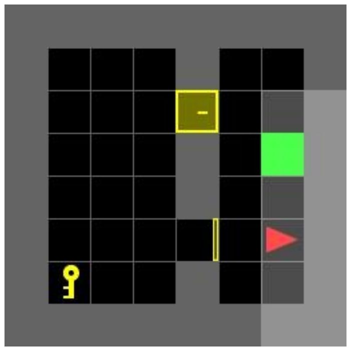
2) *One closed doors:*

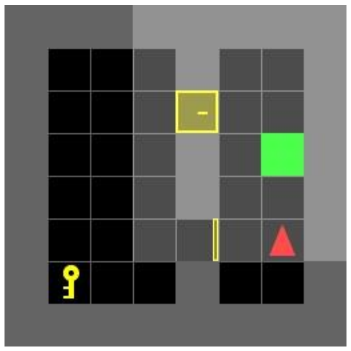


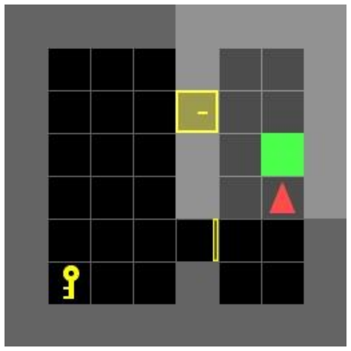




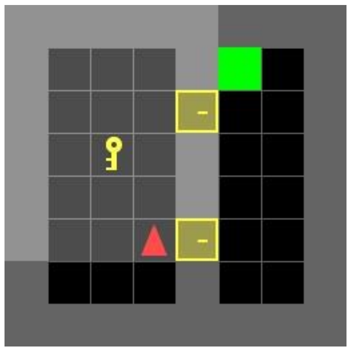


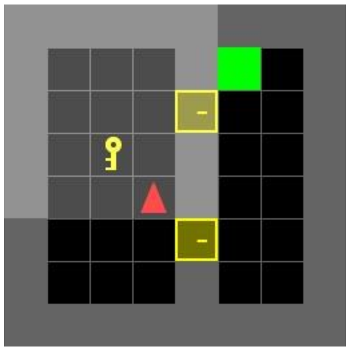
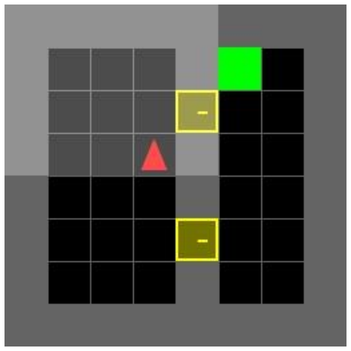
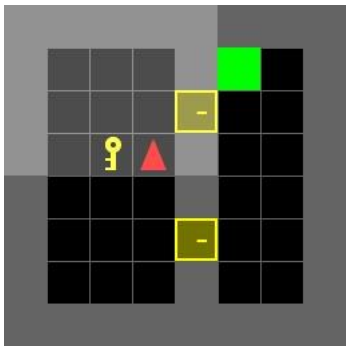
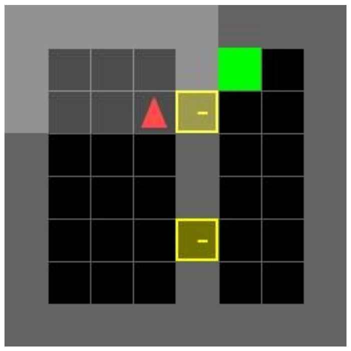
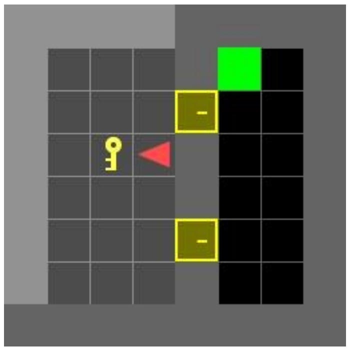
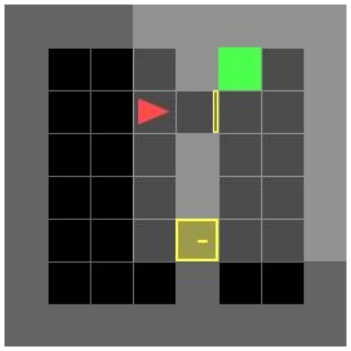
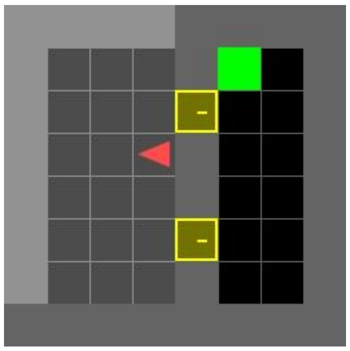
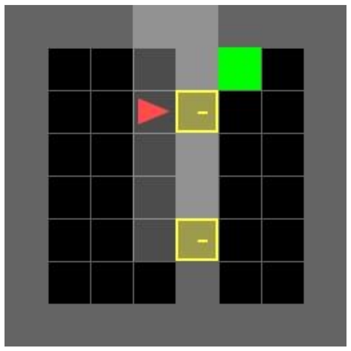


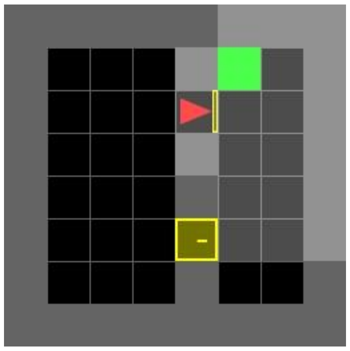
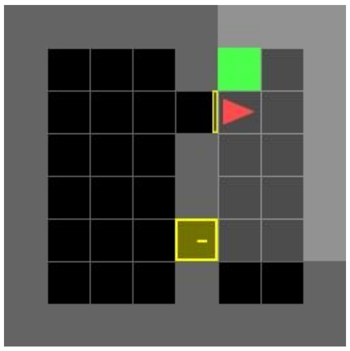
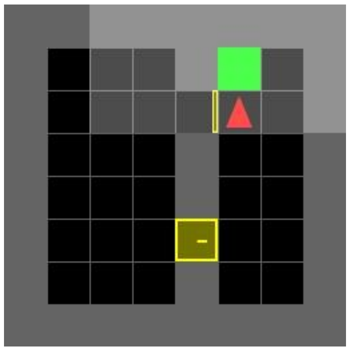




3) *Both doors closed:*



All these results show that the program written for this project is successful in predicting the shortest path in every variation possible.

V. ACKOWLEDGEMENTS

I would like to thank my professor of the course ECE 276B: Prof. Nikolay Atanasov and teaching assistant: Hanwen Cao for providing the knowledge that is used to successfully complete this project. I would like to thank my classmate Ashish Farande as well, with whom I had a discussion about the concepts of Label Correcting Algorithm and its usefulness in this project.

VI. REFERANCES

- Lecture notes of ECE 276B provided by the professor: Nikolay Atanasov for the year 2022 at UC San Diego.