Autonomous Navigation in a Door & Key Environment using Dynamic Programming

Sanchit Gupta
Department of Electrical and
Computer Engineering
University of California San Diego
sag006@ucsd.edu

***Abstract*—This report discusses about the methodology to design and implement autonomous navigation of a robot in a door and key environment using one of the Dynamic Programming algorithms. The Label Correcting algorithm in forward path has been implemented in this project. The results of the shortest path taken by the robot for all the known and unknown map environments are presented in this report.**

***Keywords*—Autonomous Navigation, Dynamic Programming, Label Correcting Algorithm, Shortest Path**

I. Introduction

Path planning is an integral function for the autonomous navigation of the robot. It is essential to enable the robot to navigate autonomously from a given location to the target location in the shortest path possible in any environment, while avoiding the obstacles as present in the path. Given a map of an environment, the path planning algorithms comprise of determining the paths to the target from all possible locations using different control inputs and policies. It is further responsible for choosing the shortest path from all the paths possible on the basis of cumulative costs determined for each of the paths. While path planning is essential in all kinds of environments, it becomes further necessary where the maps are denser, such as in indoor and warehouse navigation.

Dynamic Programming algorithms are popularly used to solve the path planning problems. These algorithms run backward in time, from the target location to the start position, and determine the paths available from each node to the target location along-with costs associated with each of them. This framework allows the robot to decide the shortest path with the least associated cost. Label Correcting algorithm is one such algorithm in Dynamic Programming which runs forward in time by examining the child nodes of every node, up to the target location.

This project requires us to implement a Dynamic Programming algorithm to determine the agent's shortest path from start location to the goal location in an 8x8 minigrid – door and key environment. The tasks include picking up the key and unlocking the door while traversing to the goal. The robot is provided with a set of controls comprising of Move Forward (MF), Turn Right (TR), Turn Left (TL), Pickup Key (PK) and Unlock Door (UD). In order to design the path planning function, the problem is formulated as a Markov Decision Process (MDP) with implementation of Label Correcting algorithm in forward time to determine the optimal control policy and value function, in-order to reach the goal in shortest path possible. The algorithm computes the shortest path between a direct path to the goal and path via crossing the door, and accordingly provides the output of optimal control policy and value function.

II. Problem Formulation

This section will discuss about the formulation and implementation of Markov Decision Process and Dynammic Programming in terms of mathematical formulas. It is given that the map comprises of a door and key environment with the red triangle representing the agent and the green square representing the goal location. The objective is to get the agent to the goal location following the shortest possible path while traversing through doors and walls. If the shortest path goes via door and the door is locked, the agent needs to pickup the key and unlock the door.

The agent can perform actions such as Move Forward (MF), Turn Right (TR), Turn Left (TL), Pickup Key (PK) and Unlock Door (UD), thus, the control inputs become a set as shown in equation 1. The state space for this problem is shown in equation 2. Executing any of the given actions has a cost associated with it.

$$U = \{MF, TR, TL, PK, UD\} \tag{1}$$

$$X = \begin{bmatrix} Agent\ Position \\ Agent\ Orientation \\ Door\ Position \\ Key\ Position \\ Goal\ Position \end{bmatrix} \tag{2}$$

A Markov Decision Process (MDP) is a stochastic process with controlled transitions defined by the tuple $(X, U, p_0, p_f, T, l, q, \gamma)$. However, the given problem is completely deterministic in nature as there is no randomness defined in the movement of the agent. It is guaranteed that for a given control input, the agent will move to the desired location. The references for the terms in the tuple are given below:

- X is the state space. It is discreet in our problem, as given in equation 2.

- U is the set of controls. It is also discreet in our problem as given in equation 1.
- p_0 is the prior PDF defined on X. This will vary with the cases as defined in Part A and Part B of our problem.
- p_f is the conditional PDF defined on X. It will not be applicable to our problem as our problem is deterministic in nature and the motion model defined in Part A and Part B of our problem defines the movement of the agent.
- T is the finite time horizon in our problem. It refers to the finite number of steps it takes to reach from start to goal location.
- $l(x, u)$ is the stage cost of choosing control $u \in U$ in state $x \in X$. $q(x)$ is the terminal cost. The costs for the problems are defined further in this section.
- γ is the discount factor. It is equal to 1 in our problem as the problem is of finite time horizon.

Cost Definition: As each action has a cost associated with it, the step costs are defined as follows:

- Cost of each action is defined on the basis of proximity of the node with key, door and goal location. If the shortest path requires picking up the key and unlocking the door, the cost of going to each node in the path is updated according to its proximity with key first, then with door and ultimately with goal. This is in general the cost associated with the action MF.
- An additional cost of 0.5 is added for the turns (TL, TR)as turns are an additional step to take. If it requires multiple turns to reach a node, 0.5 * number of turns is added to the cost.
- No cost is given to pick-up key (PK) and unlock door (UD) to simplify the problem. Further, these are the rewards for these intermediate milestones.
- The stage cost $l(x, u)$ is computed using the three definitions stated above.
- The terminal cost $q(x)$ is infinity at the starting and once the agent reaches the goal location, it is updated to 0, in order to emphasize the highest reward for reaching the goal.

The Markov Decision Process problem is now formulated for both Part A and Part B of our problem and it is elaborated below.

Part A: In part A, it is given that the map is known and the positions of key, door and goal remains fixed for one environment across all instances. Further, the starting position and location of the agent also remains fixed for one environment across all instances. However, all these parameters vary from one environment to another. The size of the environments also differs from each other. Hence, the state space (X) and control inputs (U) for this part of the problem are as defined in equation 3.

$$X_T = \begin{bmatrix} Agent\ Position \\ Agent\ Orientation \\ Door\ Position \\ Key\ Position \\ Goal\ Position \end{bmatrix} \tag{3}$$

$$U_T = \{MF, TR, TL, PK, UD\}$$

It is required to compute a control policy for each of the 7 environments, which will enable the agent reach the goal through the shortest possible path. Basis the cost definition, the problem in Part A is to minimize the value function at every step and accordingly, calculate the optimal control policy ($\pi_t^*(x_t)$) and optimal value function ($V_t^*(x)$), as given in equation 4.

$$V_t^*(x) = \min V_t^\pi(x) = q(x_T) + \sum_t^{T-1} l(x_T, \pi_T(x_T))$$
$$\pi_t^*(x_t) = argmin(V_t^*(x)) \tag{4}$$

Part B: Part B consists of random environments and hence, it is different from part A. The size of the environment is an 8x8 grid, with the fixed starting location of the agent at (3,5). Its starting orientation is also fixed to be facing up. There are two doors in the environments instead of one, at (4,2) and (4,5). The doors can either be open or closed. The key is randomly located in one of the positions from {(1,1), (2,3), (1,6)} and the goal is randomly located in one of the positions from {(5,1), (6,3), (5,6)}. Thus, there can be 2 x 2 x 3 x 3 different environments and states. However, once the environment is loaded, there would be one key position, one goal position and two door positions with their status given. The state space for this part is thus modified while the control inputs and cost definitions remain the same. They are shown in equation 5.

$$X_T = \begin{bmatrix} Agent\ Position \\ Agent\ Orientation \\ Door\ 1\ Position \\ Door\ 2\ Position \\ Key\ Position \\ Goal\ Position \end{bmatrix} \tag{5}$$

$$U_T = \{MF, TR, TL, PK, UD\}$$

There would be a single optimal policy which will enable the agent to reach the goal via the shortest path. The policy for one configuration of environment will however be different from another configuration. Thus, this part requires us to compute the optimal control policies for all 36 configurations and accordingly, provide the output for the configuration which is loaded in the environment. The problem to compute the optimal control policy ($\pi_t^*(x_t)$) and optimal value function ($V_t^*(x)$) remains the same as given in equation 5. Instead of computing these once, these are now to be computed recursively for all 36 configurations of the environment.

Dynamic Programming Formulation: The Dynamic Programming algorithm is required to be implemented to compute optimal policies for Part A and Part B. As the given problem is of Deterministic Shortest Path, the Label Correcting (LC) Algorithm in forward time would yield fast and better results for this problem. The LC algorithm will not visit every node in the graph, but it will give the shortest path from start location to all nodes. This will enable us to compute paths to key and door locations if the shortest path to the goal requires unlocking the door.

To formulate the problem for LC algorithm, the environment is treated as the graph. The four locations where the agent can move to from current location would be treated as child nodes. The conditions would be checked as shown in the Figure 1 and the loop will run recursively till the goal position is reached. The algorithm would choose the child node with the least cost to move forward.

LC algorithm would also determine if a direct path is available to the goal. Incase the direct path is not available or if the cumulative value function for it is higher than that of going through the door, the algorithm will give the output of optimal control policy accordingly.

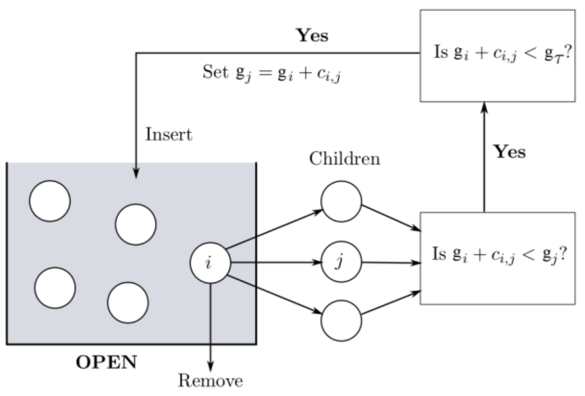

Yes
Set $g_j = g_i + c_{i,j}$
Is $g_i + c_{i,j} < g_T$?
Insert
Children
Yes
i
j
Is $g_i + c_{i,j} < g_j$?
OPEN
Remove

Figure 1: Flowchart for Label Correction Algorithm

Now, we have the state space and control inputs for both the parts of the problem. We have defined the stage and terminal costs and we have idea about the environment. Further, we have the framework of the LC algorithm to solve both the problems. Once the algorithm has found the goal position, it would give the shortest path with optimal control policy and optimal value function. The implementation of these is discussed in next section of the report.

III. Technical Approach

The technical approach to solve the shortest path problem for both the parts using Label Correcting algorithm, as formulated in Section II, is discussed in this section and is elaborated in different sub-sections below.

A. Initialization of Label Correcting Algorithm

In the Label Correcting algorithm, the labels of all the nodes are first initialized as infinity. The label for the starting position of the agent is initialized with the value zero. These are initialized in a matrix of the same dimensions as that of the environment and it is called as ‘State Cost matrix’. This indicates that the values of labels for key position, door positions, walls and goal position are infinity initially. An Open list is created which will store the co-ordinates of all the nodes being visited.

A function ‘get_child_nodes’ is written in utils.py to obtain the child nodes for any parent node. This function takes in the input of co-ordinates of parent node and provides a list of co-ordinates of all four child nodes in the order of North, West, East and South. The type of each of these nodes is also identified from the environment functions.

B. Information obtained from the Environment

The environment is initialized as the dictionary. The information such as starting position of the agent, orientation of the agent at the start, co-ordinates of key position, co-ordinates of positions of doors and co-ordinates of goal position are extracted from the dictionary and stored in the variables. For Part B of the problem statement, the status of both the doors is also obtained from the environment and stored.

C. Step Cost function in utils.py

All the costs as defined in Section II are implemented in this function. The function is modified from the originally provided one. The function is divided into three parts – the costs till the key position, costs from key position to the door position and lastly, costs from door position to the goal position. These are executed only if the direct path to the goal is not available. If a direct path is possible, it calculates the costs directly with respect to the goal location. The agent location, its orientation, child nodes and their types, key position, door position, goal position and State Cost matrix are given as input to this function. Basis the scenario, it updates the State Cost matrix and returns it back. This is where the labels for all the nodes are updated following the Label Correcting algorithm.

D. Determination of Next Step

Once the State Cost matrix is updated, the costs to transiting to each of the child nodes is analyzed following the two conditions given in Figure 1. The child node which satisfies both the conditions and which has the least stage cost becomes the next step for the agent. This node is then added to the Open list.

E. Motion Model and Policy function

This function is responsible for giving the sequence of actions to transit from current location to the next step. This is also written in utils.py. It takes the input of agent location, agent orientation, key position, door position and goal position. It analyzes for different configurations between the next step and agent's current location using its orientation and accordingly, gives the output of sequence of control inputs required to navigate the agent to next location. It further checks if the next step is the key position or the door position and correspondingly, adds PK or UD control input to the sequence.

F. Check for Direct Path

The algorithm deployed for both Part A and Part B first checks if a direct path to the goal is available. The direct path can be the shortest path available or it can be via the door. In either case, the cumulative value function of the direct path is computed once the goal location is found and it is denoted as 'value_Function_Direct' in the code. The optimal policy is also stored in this case.

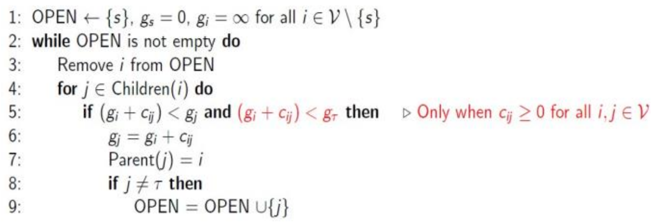
G. Check for path via key pickup and unlocking the door

The algorithm further computes the optimal policy and optimal value function for the path which includes picking up the key and unlocking the door. This algorithm also runs till the goal location is found. The cumulative value function of all the steps combined is stored and it is denoted as 'value_Function_Indirect' in the code. In Part A, it computes the optimal policy and value function only for single door. However, in Part B, it computes the parameters for both the doors. The one with the low optimal value function between the sequences for both the doors is taken as the shortest path sequence.

H. Choice between Direct Path and Path via key pickup

Now that the cumulative value functions for both the paths are computed, they are compared in order to obtain the optimal policy. The one with the lower value function is taken as the optimal policy and action sequence. However, if the direct path goes via door, it is neglected. This optimal policy is passed to execute the sequence of actions.

On a high level, the algorithm as shown in Figure 2 is implemented. The while loop is run until the goal position is found. The Step cost calculation function and the motion model and policy function are called in between in-order to update the state cost matrix and to determine the next step with corresponding action sequences.



```
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 $(g_i + c_{ij}) < g_j$ and $(g_i + c_{ij}) < g_\tau$ then    ▷ Only when $c_{ij} \geq 0$ for all $i, j \in \mathcal{V}$
6:             $g_j = g_i + c_{ij}$
7:             Parent($j$) = $i$
8:             if $j \neq \tau$ then
9:                 OPEN = OPEN ∪$\{j\}$
```

Figure 2: Algorithm implemented for Label Correcting Algorithm

The value function for different steps taken by the algorithm over time is plotted for a couple of environments from Part A. The plot obtained justifies that the algorithm has been implemented properly and it is functioning efficiently. The value function plots for 8x8-normal.env and 8x8-shortcut.env environments from Part A are shown in Figures 3 and 4 respectively.

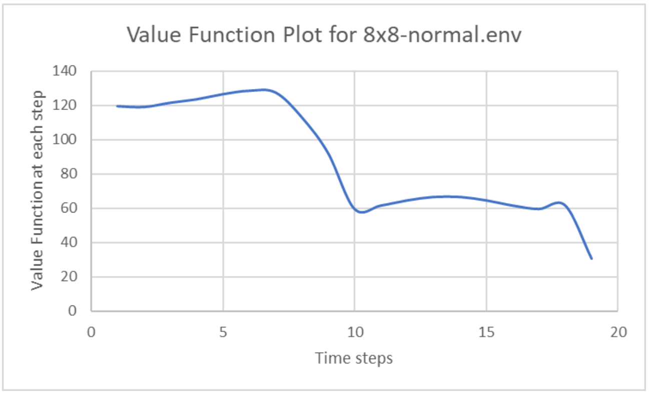


Figure 3: Value function plot for 8x8-normal.env over all time steps

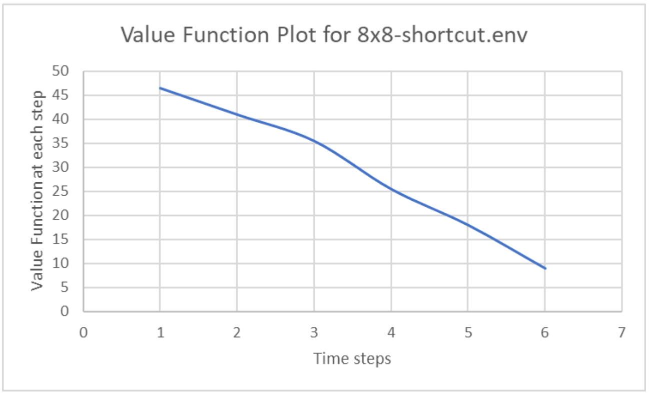


Figure 4: Value function plot for 8x8-shortcut.env over all time steps

IV. RESULTS

This section discusses about the results as obtained for all the 7 environments in Part A and for 4 random environments in Part B. For all these environments, the sequence of actions and the series of images are reported. The GIF files are also submitted along with this report for all these environments.

A. Results for Part A

1) Environment: doorkey-5x5-normal.env:

Control Policy output: $TL \to PK \to TR \to UD \to MF \to MF \to TR \to MF$. The sequence of images is shown in Figure 5.

2) Environment: doorkey-6x6-normal.env:

Control Policy output: $MF \to TR \to PK \to MF \to MF \to TR \to MF \to TL - MF \to TR \to UD \to MF \to MF \to TR \to MF \to MF \to MF$. The sequence of images is shown in Figure 6.

3) Environment: doorkey-8x8-normal.env:

Control Policy output: $TR \to MF \to TR \to TR \to MF \to MF \to TR \to MF \to MF \to MF \to TR \to PK \to TR \to MF \to MF \to MF - MF - TR \to UD \to MF \to MF \to TR \to MF \to MF \to MF \to MF \to MF \to TL \to MF$. The sequence of images is shown in Figure 7.

4) Environment: doorkey-6x6-direct.env:

Control Policy output: $TR \to TR \to MF \to MF$. The sequence of images is shown in Figure 8.

5) Environment: doorkey-8x8-direct.env:

Control Policy output: $TL \to MF \to MF \to MF$. The sequence of images is shown in Figure 9.

6) Environment: doorkey-6x6-shortcut.env:

Control Policy output: $PK \to TR \to TR \to UD \to MF \to MF$. The sequence of images is shown in Figure 10.

7) Environment: doorkey-8x8-shortcut.env:

Control Policy output: $MF \to TR \to PK \to TR \to TR \to MF \to TL \to MF - TR \to UD \to MF \to MF$. The sequence of images is shown in Figure 11.

B) Results for Part B

1) Environment: Key position at (1,1) and Goal position at (5,1) with first door open and other door closed

Control Policy output: $MF \to MF \to MF \to TR \to MF \to MF \to TL \to MF$. The sequence of images is shown in Figure 12.

2) Environment: Key position at (1,6) and Goal position at (5,1) with both the doors open

Control Policy output: $MF \to MF \to MF \to TR \to MF \to MF \to TL \to MF$. The sequence of images is shown in Figure 13.

3) Environment: Key position at (1,1) and Goal position at (5,6) with first doors closed and second door open

Control Policy output: $TR \to MF \to MF \to TR \to MF$. The sequence of images is shown in Figure 14.

4) Environment: Key position at (1,6) and Goal position at (5,6) with both the doors closed

Control Policy output: $TL \to MF \to MF \to TL - PK - TL \to MF \to MF \to UD \to MF \to MF \to TR \to MF$. The sequence of images is shown in Figure 15.

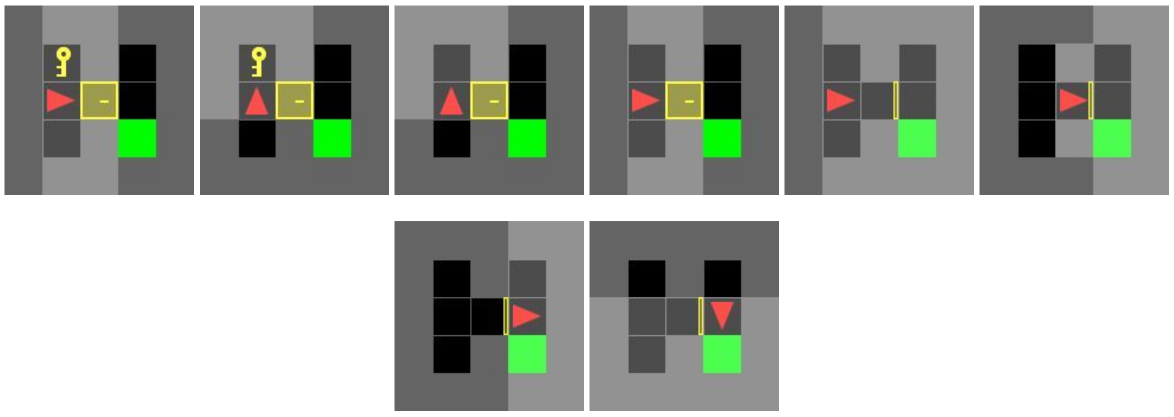


Figure 5: Sequence of images as per optimal control policy for Environment in Part A: doorkey-5x5-normal.env

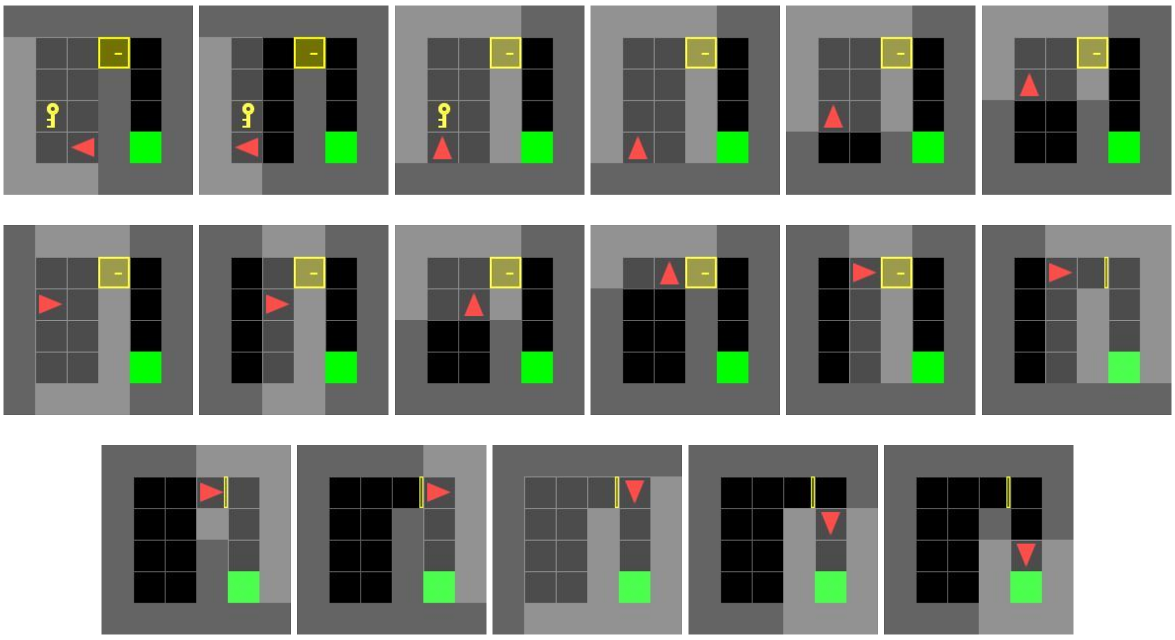


Figure 6: Sequence of images as per optimal control policy for Environment in Part A: doorkey-6x6-normal.env

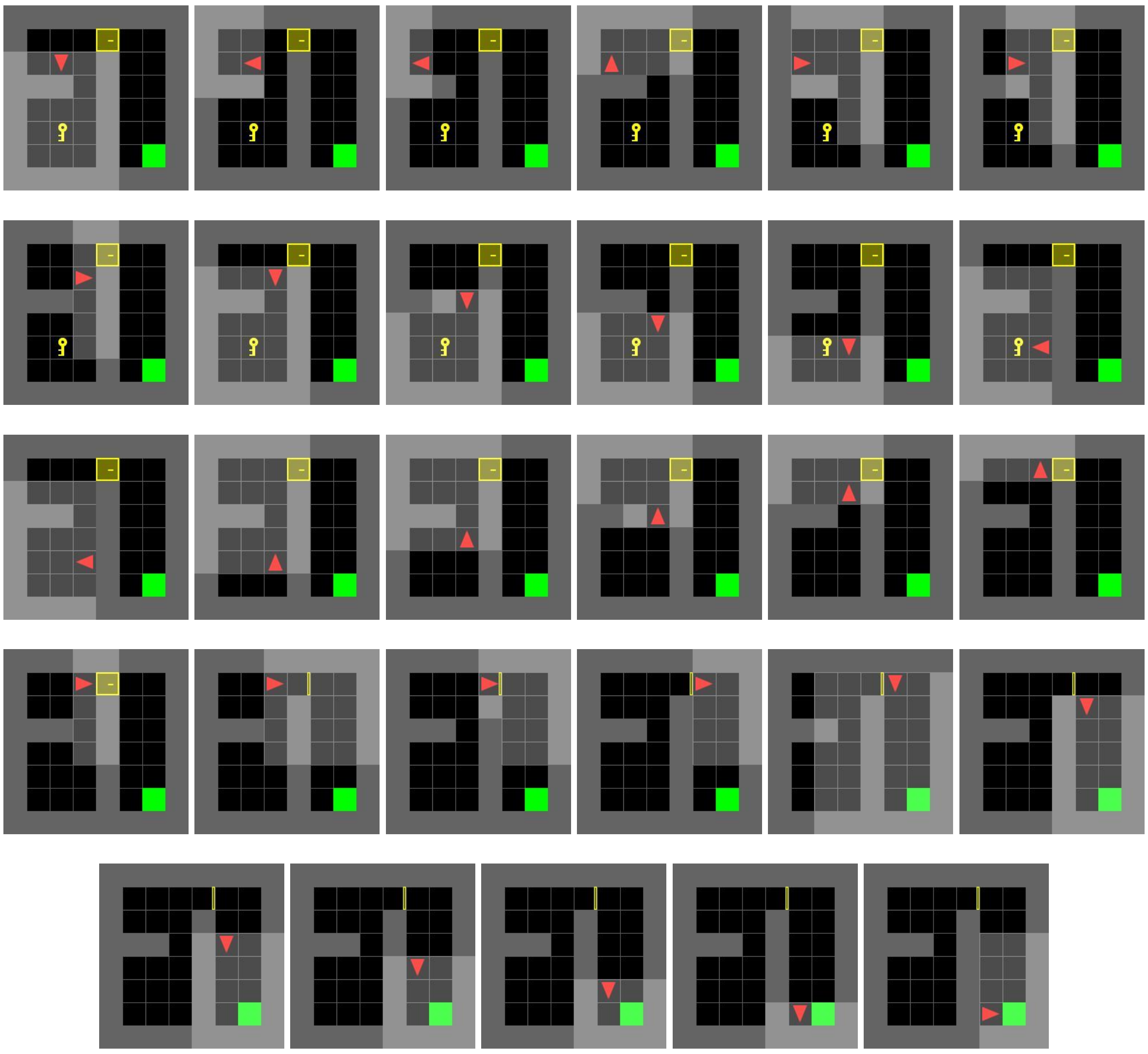


Figure 7: Sequence of images as per optimal control policy for Environment in Part A: doorkey-8x8-normal.env

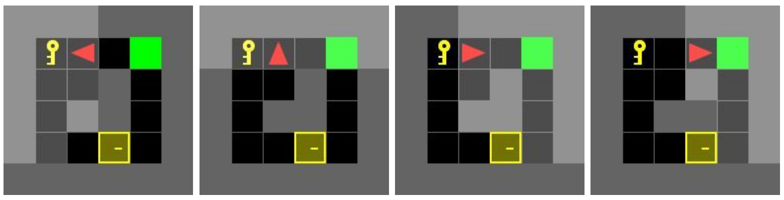


Figure 8: Sequence of images as per optimal control policy for Environment in Part A: doorkey-6x6-direct.env

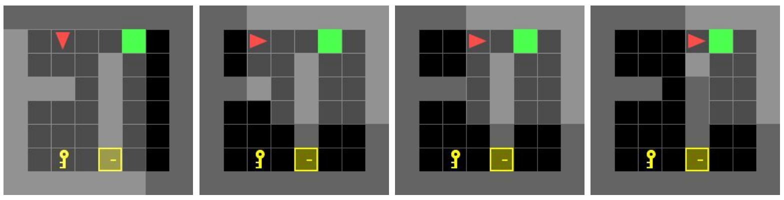


Figure 9: Sequence of images as per optimal control policy for Environment in Part A: doorkey-8x8-direct.env

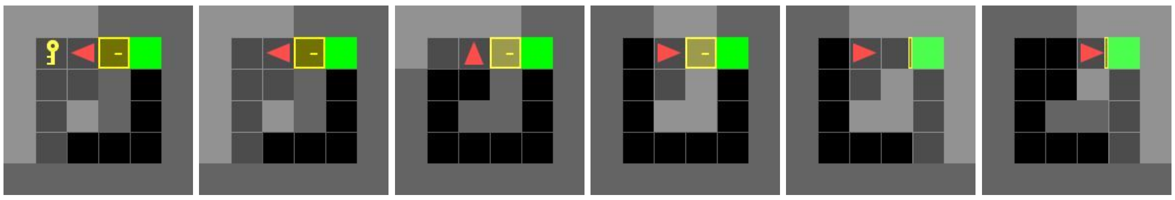


Figure 10: Sequence of images as per optimal control policy for Environment in Part A: doorkey-6x6-shortcut.env

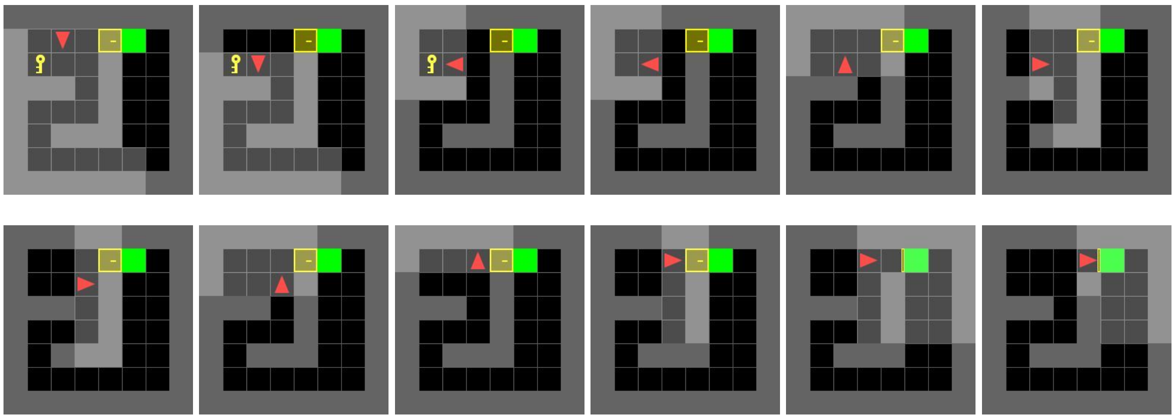


Figure 11: Sequence of images as per optimal control policy for Environment in Part A: doorkey-8x8-shortcut.env

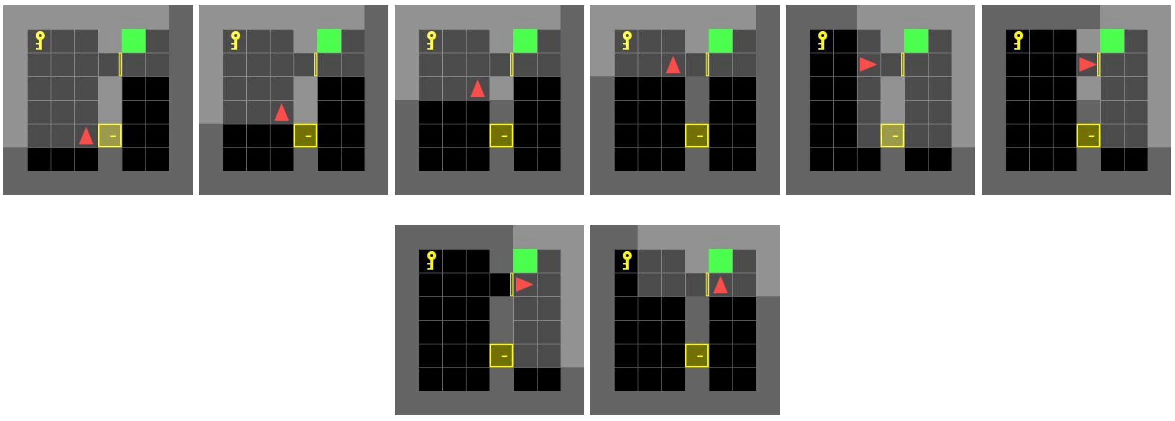


Figure 12: Sequence of images as per optimal control policy for Environment in Part B: Random Environment 1

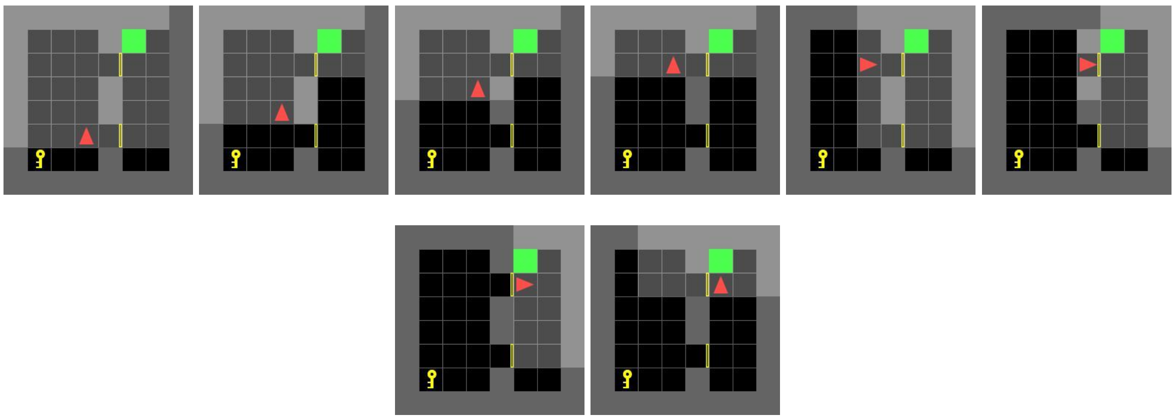


Figure 13: Sequence of images as per optimal control policy for Environment in Part B: Random Environment 2

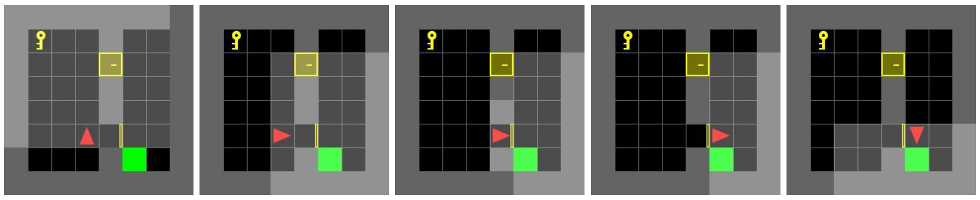


Figure 14: Sequence of images as per optimal control policy for Environment in Part B: Random Environment 3

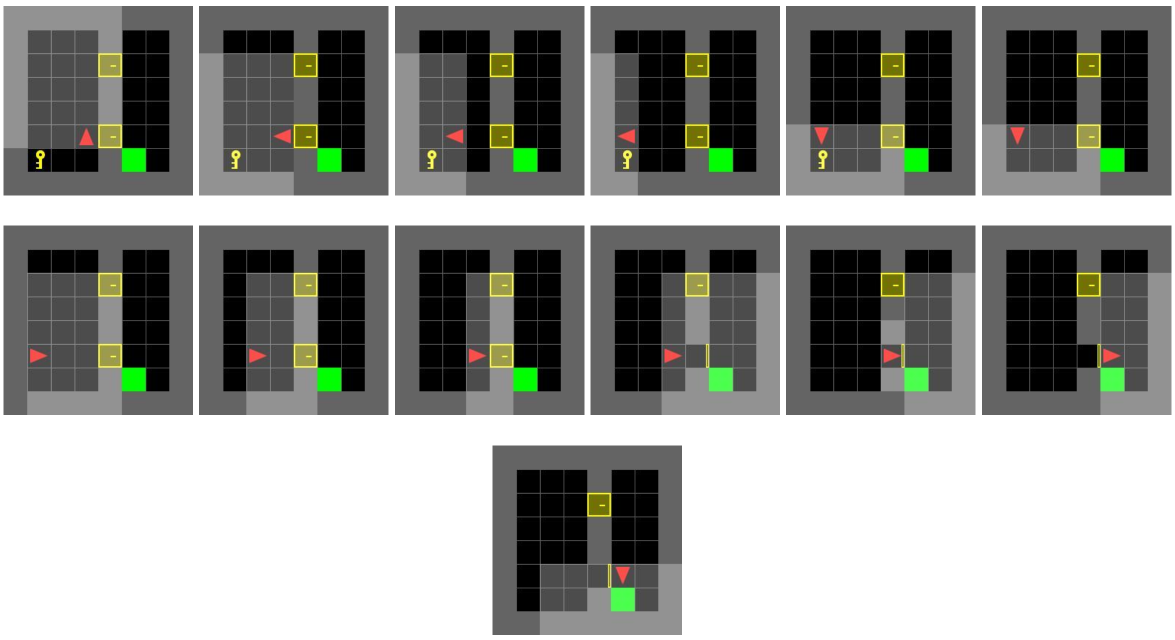


Figure 15: Sequence of images as per optimal control policy for Environment in Part B: Random Environment 4