Dynamic Programming

Manoj Kumar Reddy Manchala
Department of Mechanical and Aerospace Engineering
University of California San Diego
La Jolla, U.S.A.
mmanchala@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
University of California San Diego
La Jolla, U.S.A.
natanasov@eng.ucsd.edu

***Abstract*—This paper focuses on developing a robust dynamic programming approach tailored for a simulated 'Door & Key' environment, where an agent must navigate through a grid with randomly placed obstacles, keys, and doors, formulating the problem as a Markov Decision Process and implemented both known and random map scenarios. This potentially forms a base for the applications of autonomous navigation and robotics.**

***Index Terms*—Orientation Tracking, Constrained Gradient Descent, Panorama, styling, insert**

I. Introduction

The problem this paper aims to solve is autonomous navigation in a Door & Key environment with an objective of getting our agent to the goal location by collecting the keys and navigating through doors. This is an important step to proceed further with autonomous navigation in complex and uncertain environments characterized by movable obstacles and essential interactable objects like doors and keys. The solution contains two parts a) known-map, where the entire environment is known prior to the computation of the control policy and part b) random-map where the map essentially remains the same although the key position, goal position and the door status is not fixed. The state space and the motion model for both the cases is defined differently to handle the flexibility of working for a generalized map for the first part and to handle the changing positions of keys and goal position in the second part. Based on this motion model, the dynamic programming algorithm is applied to compute the optimal control policy.

II. Problem Formulation

This section details the formulation of the autonomous navigation problem in a door & key environment within the framework of a Markov Decision Process (MDP). Our MDP is defined as follows:

A. State Space

The state space X encompasses all possible configurations of the agent within the environment.

1) ***Known Map****:* In the first case of known map, the key state of the key and the status of the door are included to be part of the state. Each state $x \in X$ is defined as a tuple (x, y, o, k, d), where:

- x and y denote the agent's grid position,
- o represents the orientation of the agent (North, South, East, West),
- k is a binary variable indicating whether the agent is carrying the key (0 = no, 1 = yes),
- d indicates the status of the door (0 = locked, 1 = unlocked).

2) ***Random Map****:* The state space X encompasses all possible configurations of the agent within the environment. In the second case of random map, the status of the two doors, goal position and key position indexes are included to be part of the state. Each state $x \in X$ is defined as a tuple $(x, y, o, k, d0, d1, ki, di)$, where:

- x and y denote the agent's grid position,
- o represents the orientation of the agent (North, South, East, West),
- k is a binary variable indicating whether the agent is carrying the key (0 = no, 1 = yes),
- $d0$ indicates the status of the first door (0 = locked, 1 = unlocked),
- $d1$ indicates the status of the second door (0 = locked, 1 = unlocked),
- ki indicates the index of position of the key,
- gi indicates the index of position of the goal.

B. Action Space

The action space U consists of all the actions the agent can take at any state. The set of actions includes:

- Move Forward (MF),
- Turn Left (TL),
- Turn Right (TR),
- Pick Up Key (PK),
- Unlock Door (UD).

C. Motion Model

The motion model f determines the state transitions given the current state and action. It is defined functionally as:

$$x_{t+1} = f(x, u)$$

where x_{t+1} is the next state resulting from action u applied at state x. The motion model takes into account the environmen-

tal constraints such as walls, doors, and different key positions and goal positions.

Let $\Delta x, \Delta y$ be the change in coordinates based on o :

$$\Delta x, \Delta y = \begin{cases} (0,-1) & \text{if } o = \text{N} \\ (1,0) & \text{if } o = \text{E} \\ (0,1) & \text{if } o = \text{S} \\ (-1,0) & \text{if } o = \text{W} \end{cases}$$

where o represents orientation.

*1) **Known Map**:*

- Move Forward (MF): If the next cell is within bounds (or) cell contains a door and the agent has the key, then

$$\text{next state} = (x + \Delta x, y + \Delta y, o, k, d)$$

- Turn Left (TL) or Turn Right (TR): Direction dictionaries are created to handle the new orientations as follows:

$$\text{left orientation} = N : W, W : S, S : E, E : N$$
$$\text{right orientation} = N : E, E : S, S : W, W : N$$
$$\text{next state} = (x, y, o', k, d)$$

  where o' is new-orientation[o] corresponding to left or right turn.
- Pick Key (PK): If the next cell contains a key and the agent is not carrying a key then

$$\text{next state} = (x, y, o, 1, d)$$

- Unlock Door (UD): If the next cell contains a door and the door is locked while the agent is carrying a key then

$$\text{next state} = (x, y, o, k, 1)$$

*2) **Random Map**:* Given the action and the current state, the new state $(x', y', o', k', d_0', d_1', k_i, g_i)$ is defined as:

- Move Forward (MF): If the next cell is within bounds and empty (or) cell contains a door and the agent has the key, then

$$\text{next state} = (x + \Delta x, y + \Delta y, o, k, d_0, d_1, k_i, g_i)$$

- Turn Left (TL) or Turn Right (TR): Direction dictionaries are created to handle the new orientations as follows:

$$\text{left orientation} = N : W, W : S, S : E, E : N$$
$$\text{right orientation} = N : E, E : S, S : W, W : N$$
$$\text{next state} = (x, y, o', k, d_0, d_1, k_i, g_i)$$

  where o' is new-orientation[o] corresponding to left or right turn.
- Pick Key (PK): If the next cell contains a key and the agent is not carrying a key then

$$\text{next state} = (x, y, o, 1, d_0, d_1, k_i, g_i)$$

- Unlock Door (UD): If the next cell contains a door and the door is locked while the agent is carrying a key then

$$\text{next state} = (x, y, o, k, d_0', d_1', k_i, g_i)$$

D. Initial State

The initial state is dependent on the environment and is initialized as follows :

*1) **known map case**:*

- Agent's initial position as info[initial agent position],
- Agent's initial orientation as info[initial orientation],
- status of the key as env.carrying,
- status of the door as door.isopen.

*2) **random map case**:*

- Agent's initial position as (3,5),
- Agent's initial orientation as 'N',
- status of the key as 0,
- status of the door1,
- status of the door2,
- key index from (1, 1), (2, 3), (1, 6),
- goal index from (5, 1), (6, 3), (5, 6).

E. Planning Horizon

The planning horizon T is defined based on the task requirements. For this problem, we consider a finite horizon(10-100), where T is the maximum number of steps the agent takes to reach the goal or a predefined timeout.

F. Costs

The stage costs $\ell(x, u)$ and terminal costs $q(x)$ are defined as follows:

- $\ell(x, u)$ is set to 1 for any action u taken in state x which are not terminal or goal states, to encourage the shortest path. It is set to be 0 at the goal states.
- $q(x)$ is 0 if the state x is a goal state, and a high penalty otherwise to discourage terminal states that are not goal states.

These elements collectively define the MDP for our autonomous navigation problem, providing a structured approach to achieving optimal decision-making under uncertainty.

III. Technical Approach

A. Dynamic Programming Algorithm

This section describes the dynamic programming algorithm used to compute the optimal policy for an agent navigating through an environment based on a Markov Decision Process. The algorithm iteratively updates the value function and determines the optimal policy using a backward induction method.

The dynamic programming algorithm operates in a finite horizon framework where the state space X and the control space U are predefined. The terminal states have a stage cost of zero, emphasizing the goal of reaching these states. The algorithm proceeds as follows:

1) Initialize the policy for each state in X to None and the value function VT for each state to a large number (representing a high cost), except for terminal states where the cost is zero.
2) Iterate from $t = T$ down to $t = 0$:

a) For each state in X, compute the cost Q for each action in U using the equation:

$$Q(\text{action}) = \text{stage\_cost} + VT[f(\text{state, action})]$$

b) Determine the action that minimizes Q for each state and update the value function VT with the minimum cost found. Update the policy for that state with the action that resulted in the minimum cost.

The policy that minimizes the expected total cost from each state is stored, which guides the agent to the goal with the least cost.

B. Python Code

Below is the Python implementation of the above dynamic programming algorithm:

Algorithm 1 Optimal Policy Calculation using Dynamic Programming

Initialize $policy(state) \leftarrow$ None for all $state \in X$
Initialize $VT(state) \leftarrow \infty$ for all $state \in X$, except $VT(terminal\ states) \leftarrow 0$
for $t \leftarrow T$ to 0 step -1 **do**
  for each $state \in X$ **do**
    Initialize $Q(action) \leftarrow \emptyset$ for all $action \in U$
    for each $action \in U$ **do**
      $Q(action) \leftarrow$ stage_cost $+$ VT[motion_model_B$(state, action)$]
    end for
    $min\_action \leftarrow \text{argmin}(Q)$
    $VT(state) \leftarrow \min(Q)$
    $policy(state) \leftarrow min\_action$
  end for
end for=0

C. Strategy for Known Map

In the case of known maps, even though we know the environment completely, but all the 7 environments are different in size, different in their structure(wall positions) which are not included in the state. Hence we compute the dynamic programming policy seven times for the seven different environments and extract the control policy as follows:

1) **Initialization:** The agent's initial position and orientation are retrieved from the environment information:
   - *Position* (`initial_position`) is set to `info['init_agent_pos']`.
   - *Orientation* (`init_orientation`) is set to `info['init_agent_dir']`.
2) **Environment Interaction:**
   - The door's state for the specific environment is retrieved (`info["door_pos"]`)
   - The orientation is mapped from a vector to a directional label (N, E, S, W) using a predefined dictionary.
   - The status of key is checked and set to k.
   - The door's state (open or closed) is determined to set the state d.
3) **Policy Execution:** Starting from the initial state, defined as the tuple $(x, y, \text{orientation}, k, d)$:
   a) The agent follows the policy until the goal position (`info['goal_pos']`) is reached.
   b) In each step, the policy determines the next action based on the current state.
   c) The action is executed using the `motion_model`, which updates the agent's state.
   d) The new state is recorded and the action is stored in the sequence.
4) **Termination:** The process continues until the agent's position matches the goal position. The sequence of actions taken is returned.

*1) **Algorithmic Representation**:* The Python code for this procedure is represented by the following pseudocode:

Algorithm 2 Python Code for Control Policy Extraction

$initial\_position \leftarrow info['init\_agent\_pos']$
$init\_orientation \leftarrow info['init\_agent\_dir']$
$door \leftarrow env.grid.get(info["door\_pos"][0], info["door\_pos"][1])$
$orient \leftarrow \{(0,-1) : 'N', (1,0) : 'E', (0,1) : 'S', (-1,0) : 'W'\}$
$k \leftarrow 0$ **if** $env.carrying$ **is** $None$ **else** 1
$d \leftarrow 1$ **if** $door.is\_open$ **else** 0
$current\_state \leftarrow initial - state$
$actions \leftarrow []$
while $(current\_state[0], current\_state[1]) \neq info['goal\_pos']$ **do**
  $action \leftarrow policy[current\_state]$
  $actions.append(action)$
  $current\_state \leftarrow motion\_model(env, current\_state, action)$
end while
return $actions$ =0

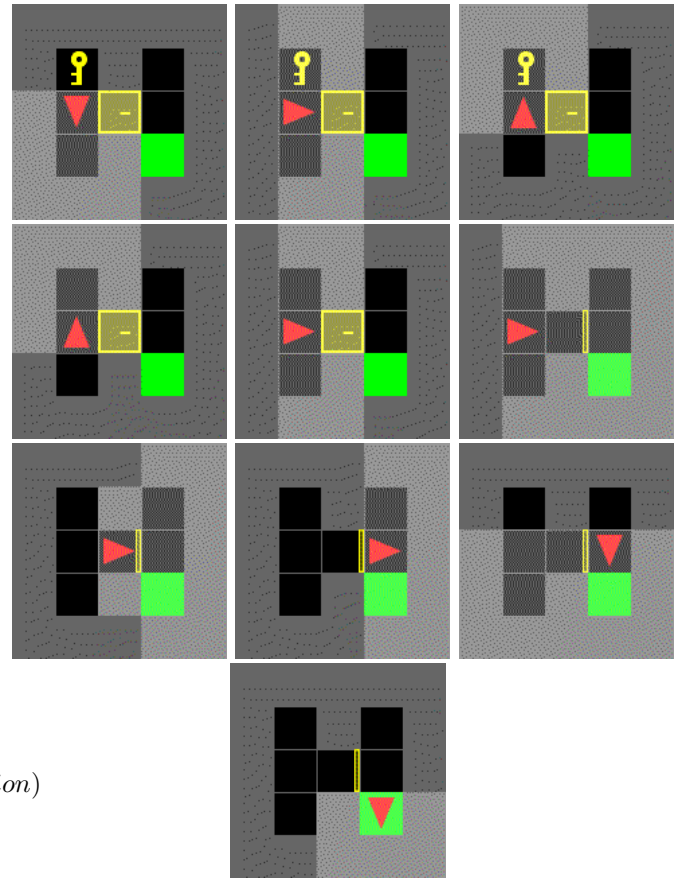
D. Strategy for Random Map

In the case of random map, we are allowed to compute the control policy only once. Here all the 36 environments are same but only the positions of key and goal are altered along with the status of the door. In order to deal with this we have included the position of key and goal and the status of both the doors inside the state vector to keep track of those elements. So we compute the policy initially and extract the control sequence from the policy.

*1) **Algorithm Description**:* The algorithm initializes with the agent's position and orientation and proceeds to check the status of doors and the positions of keys to configure the initial state. It then iterates through the environment using a predefined policy until the goal is reached.

Algorithm 3 Policy Extraction for Random Map

$initial\_position \leftarrow info['init\_agent\_pos']$
$init\_orientation \leftarrow info['init\_agent\_dir']$
$orient \leftarrow \{(0,-1) :' N', (1,0) :' E', (0,1) :' S', (-1,0) :' W'\}$
$d0 \leftarrow 1$ **if** $info['door\_open'][0]$ **else** 0
$d1 \leftarrow 1$ **if** $info['door\_open'][1]$ **else** 0
for $i \leftarrow 0$ **to** 2 **do**
  if np.all$(info['key\_pos'] == keys[i])$ **then**
    $ki \leftarrow i$
  end if
end for
for $j \leftarrow 0$ **to** 2 **do**
  if np.all$(info['goal\_pos'] == goals[j])$ **then**
    $gi \leftarrow j$
  end if
end for
$current\_state \leftarrow initial - state$
$path \leftarrow [current\_state]$
$actions \leftarrow []$
while np.any$((current\_state[0], current\_state[1]) \neq info['goal\_pos'])$ **do**
  $action \leftarrow policy[current\_state]$
  $actions.append(action)$
  $current\_state \leftarrow$ **motion_model_B**$(current\_state, action)$
  $path.append(current\_state)$
end while
return $actions$ =0



IV. Results

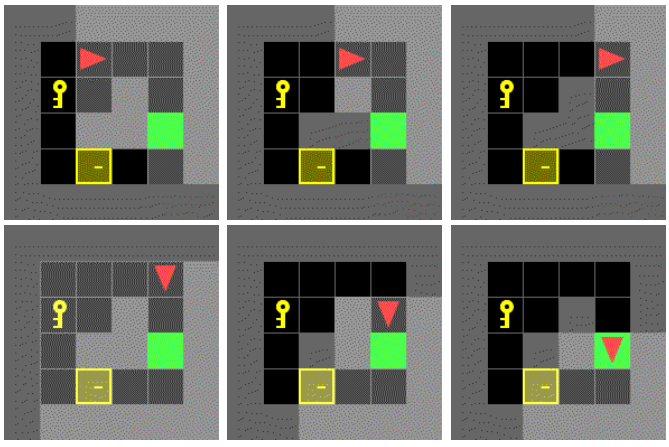
A. *Known Map*

*1) **Action Sequences**:* The following are the control sequences for all the 7 different known maps.

Env	Actions
5x5-N	$TL \rightarrow TL \rightarrow PK \rightarrow TR \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
6x6-D	$MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF$
6x6-N	$TL \rightarrow MF \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow TR \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
6x6-S	$PK \rightarrow TL \rightarrow TL \rightarrow UD \rightarrow MF \rightarrow MF$
8x8-D	$MF \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
8x8-N	$TR \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow PK \rightarrow TL \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF$
8x8-S	$TR \rightarrow MF \rightarrow TR \rightarrow PK \rightarrow TL \rightarrow UD \rightarrow MF \rightarrow MF$

*2) **Visualization : 5x5-normal**:* The following represent the sequence of images showing the motion of the agent through the environment 5x5-normal.

*3) **Visualization : 6x6-direct**:* We can see that in this environment, even though the agent does not have the key, there is still a way to navigate to the goal and the agent successfully reaches the goal.



*4) **Visualization : 8x8-normal**:* In this environment, the agent succesfully picks up the key, unlocks the door and reaches the goal.

B. *Random Map*

*1) **Action Sequences**:* The following are the control sequences for all the 36 variations of the random map.

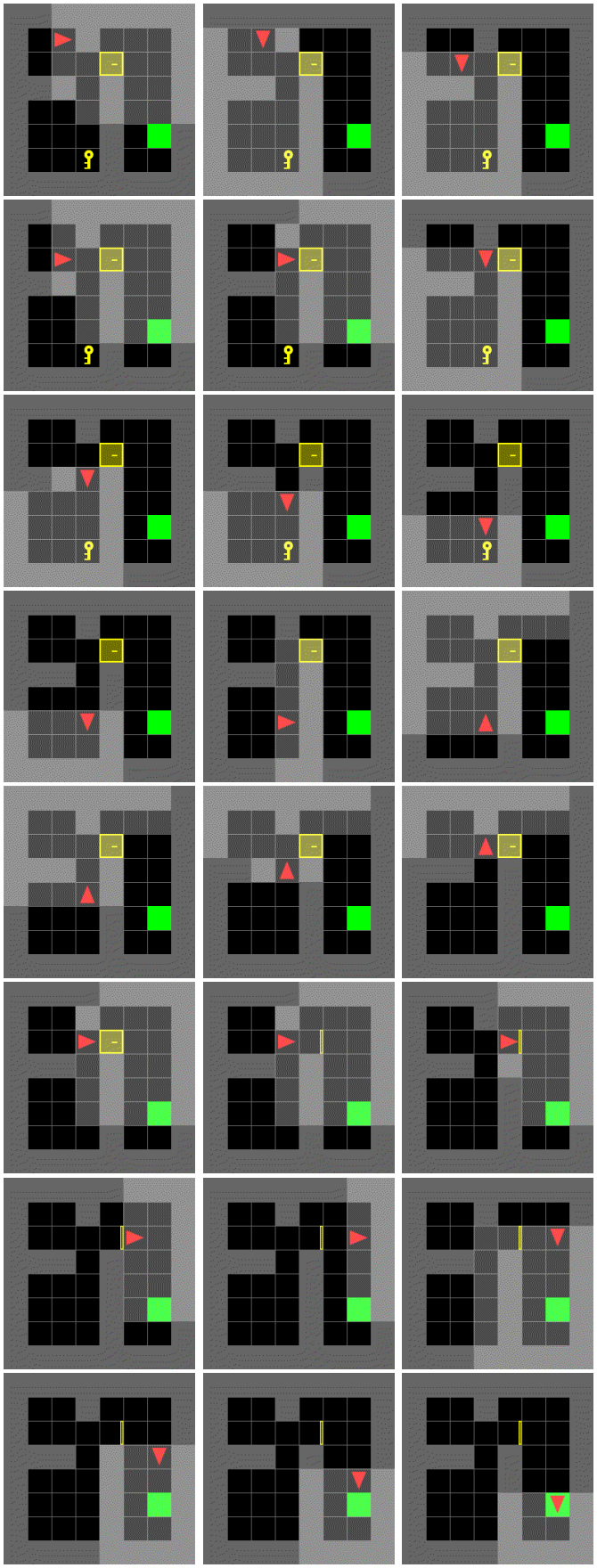
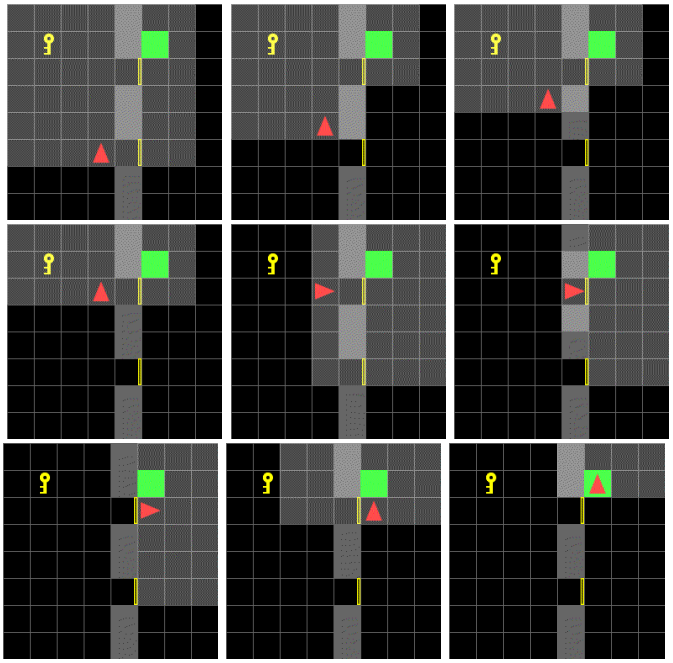


Fig. 1. Agent's motion through the 8x8-normal environment - From left to right and continue.

Env	Actions
1	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
2	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
3	$TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
4	$MF \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
5	$TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
6	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
7	$TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
8	$MF \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
9	$TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
10	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
11	$TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
12	$MF \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
13	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
14	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
15	$TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
16	$MF \rightarrow MF \rightarrow TL \rightarrow PK \rightarrow TR \rightarrow MF \rightarrow TR \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
17	$TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
18	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
19	$TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
20	$MF \rightarrow MF \rightarrow TL \rightarrow PK \rightarrow TR \rightarrow MF \rightarrow TR \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
21	$TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
22	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
23	$TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
24	$MF \rightarrow MF \rightarrow TL \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
25	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$

Env	Actions
26	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF$
27	$TR \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
28	$TL \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
29	$TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
30	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
31	$TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
32	$TL \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow MF \rightarrow MF$
33	$TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
34	$MF \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF \rightarrow MF \rightarrow MF \rightarrow MF$
35	$TR \rightarrow MF \rightarrow MF \rightarrow TR \rightarrow MF$
36	$TL \rightarrow MF \rightarrow MF \rightarrow TL \rightarrow PK \rightarrow TL \rightarrow MF \rightarrow MF \rightarrow UD \rightarrow MF \rightarrow MF \rightarrow$

*2) **Visualization : 8x8-1**:* The following represent the sequence of images showing the motion of the agent through the environment random 8x8-1.



*3) **Visualization : 8x8-4**:* The following represent the sequence of images showing the motion of the agent through the environment random 8x8-4.

*4) **Visualization : 8x8-36**:* The following represent the sequence of images showing the motion of the agent through the environment random 8x8-36.

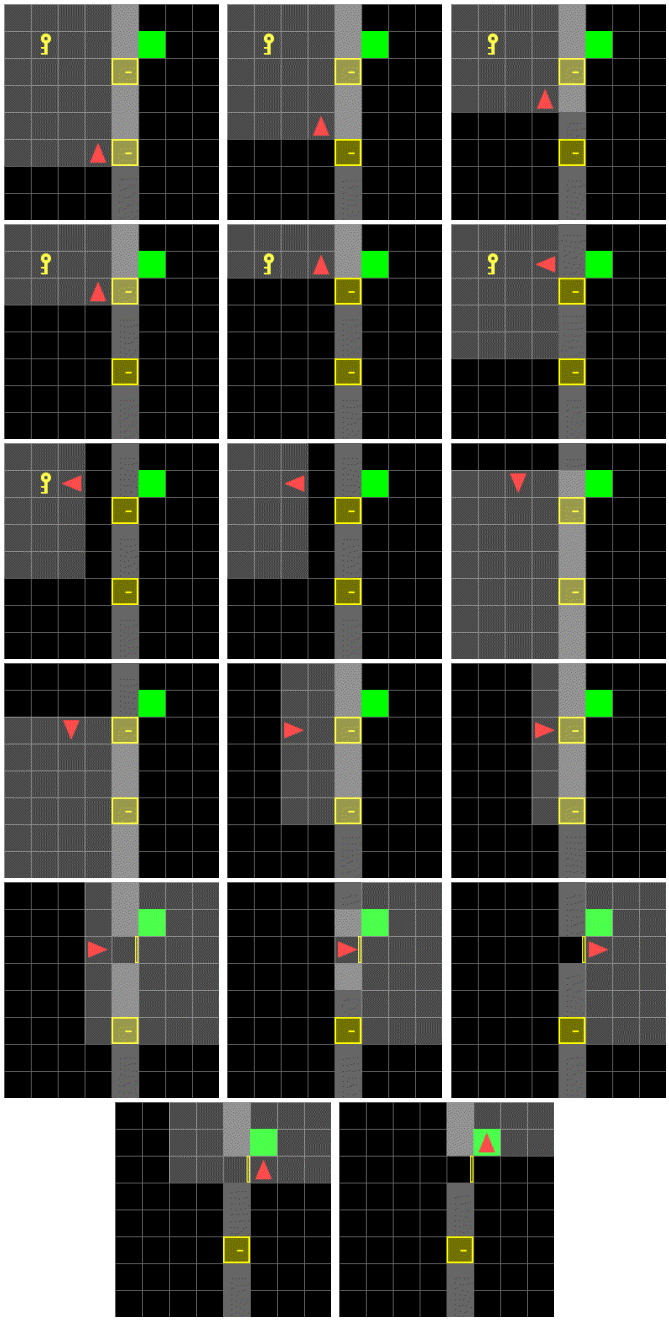


Fig. 2. Agent's motion through the 8x8-4 environment - From left to right and continue.

*5) **Visualization : 8x8-35**:* The following represent the sequence of images showing the motion of the agent through the environment random 8x8-36. We can see that since the door is already open, the agent does not try to pickup the key.

*6) **Visualization : 8x8-28**:* The following represent the sequence of images showing the motion of the agent through the environment random 8x8-28. We can see that, even though the key position is same as 8x8-36, the goal position is different and the agent reaches the goal.

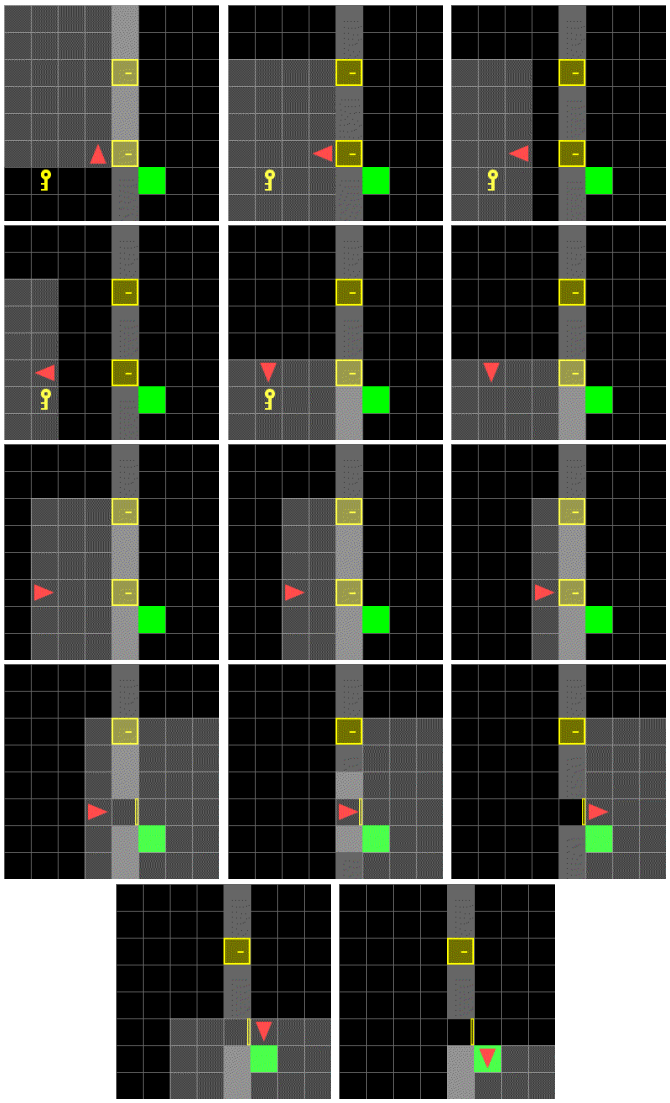


Fig. 3. Agent's motion through the 8x8-36 environment - From left to right and continue.

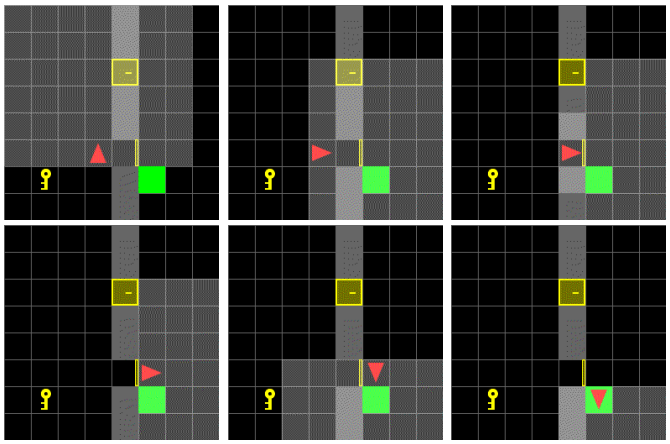


Fig. 4. Agent's motion through the 8x8-35 environment - From left to right and continue.

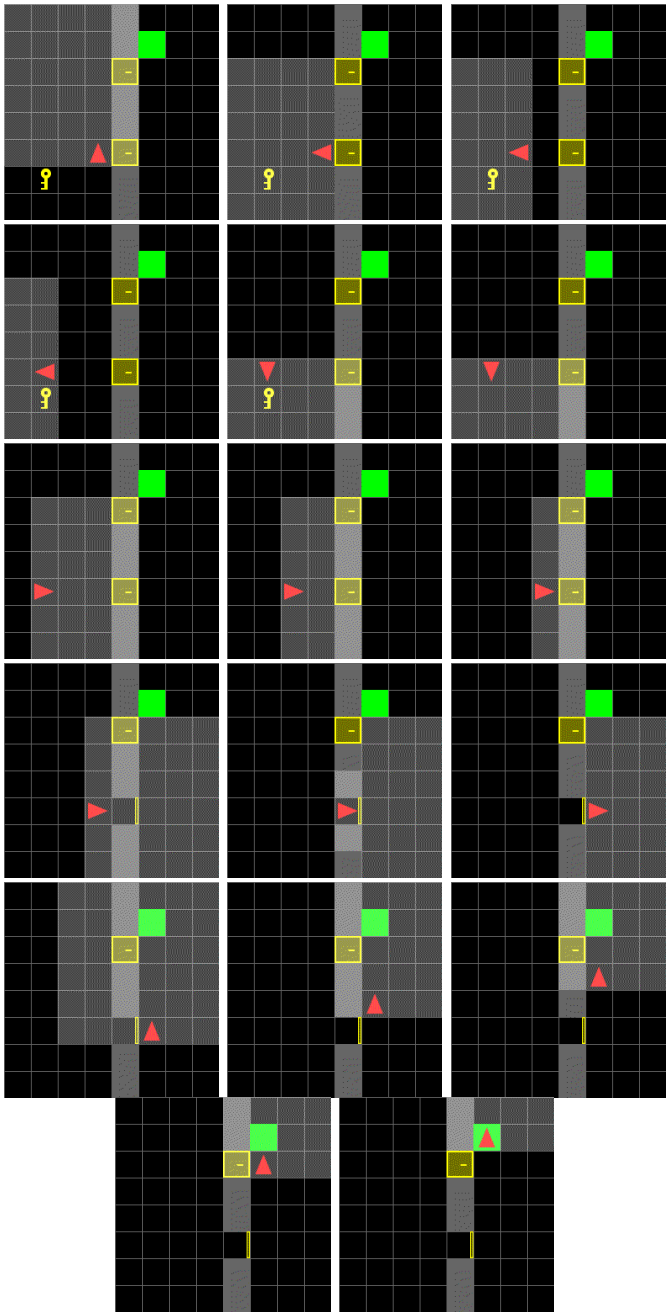


Fig. 5. Agent's motion through the 8x8-28 environment - From left to right and continue.

7) ***Visualization : 8x8-24****:* In this environment, the goal position is same as 8x8-36, but the key position is different, and the agent is able to successfully pickup the key and reach the goal.

8) ***Visualization : 8x8-22****:* In this environment, the key and goal position is same as 8x8-36, but one of the doors is open, and the agent is able to successfully go through the open door and reach the goal.

9) ***Visualization : 8x8-11****:* In this environment, the goal position is just beside the open door, and agent succesfully navigates through the open door and reaches the goal.

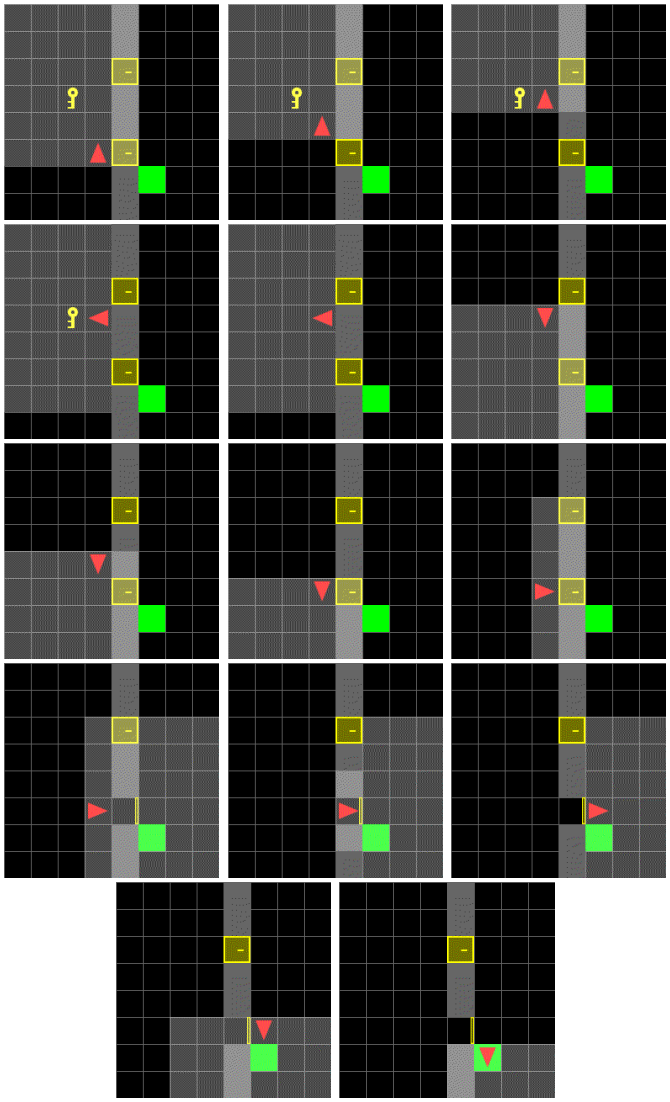


Fig. 6. Agent's motion through the 8x8-24 environment - From left to right and continue.

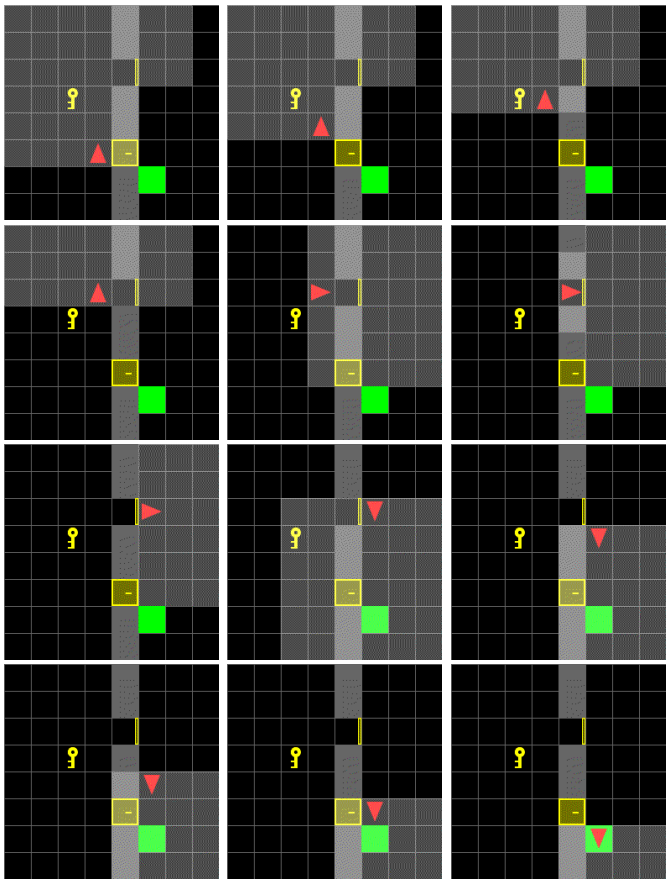


Fig. 7. Agent's motion through the 8x8-22 environment - From left to right and continue.

C. *Discussion*

- From the comparision of the motion sequences we can see that the agent is able to succesfully navigate and reach the goal both in the known map and random map cases.
- The agent is robust to different key and goal positions.
- The succesfull navigation of the agent highly depends on the way the costs are defined, for example, if the terminal states are strictly restricted to be the states where the door must be open, then even though there might be a direct path to the goal, the agent will open the door and then reach the goal instead of directly reaching the goal. The same is depicted in the sequence below.

*1) **Visualization : 8x8-direct-forced**:* In this environment, even though there is a direct route to the goal, the agent is forced to make sure that the door is open. We can see that the agent picks up the key, opens the door and only then reaches the goal.

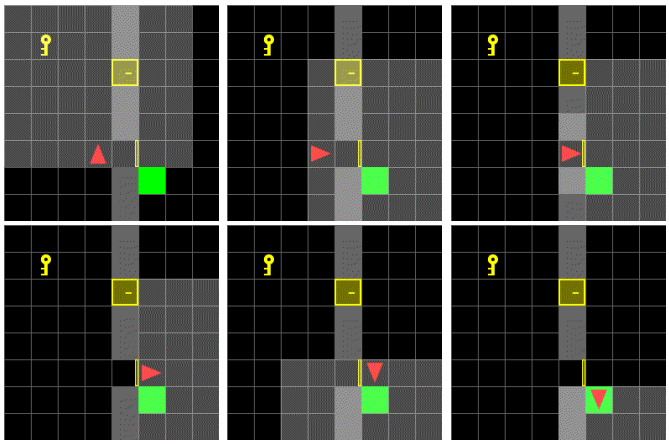


Fig. 8. Agent's motion through the 8x8-11 environment - From left to right and continue.

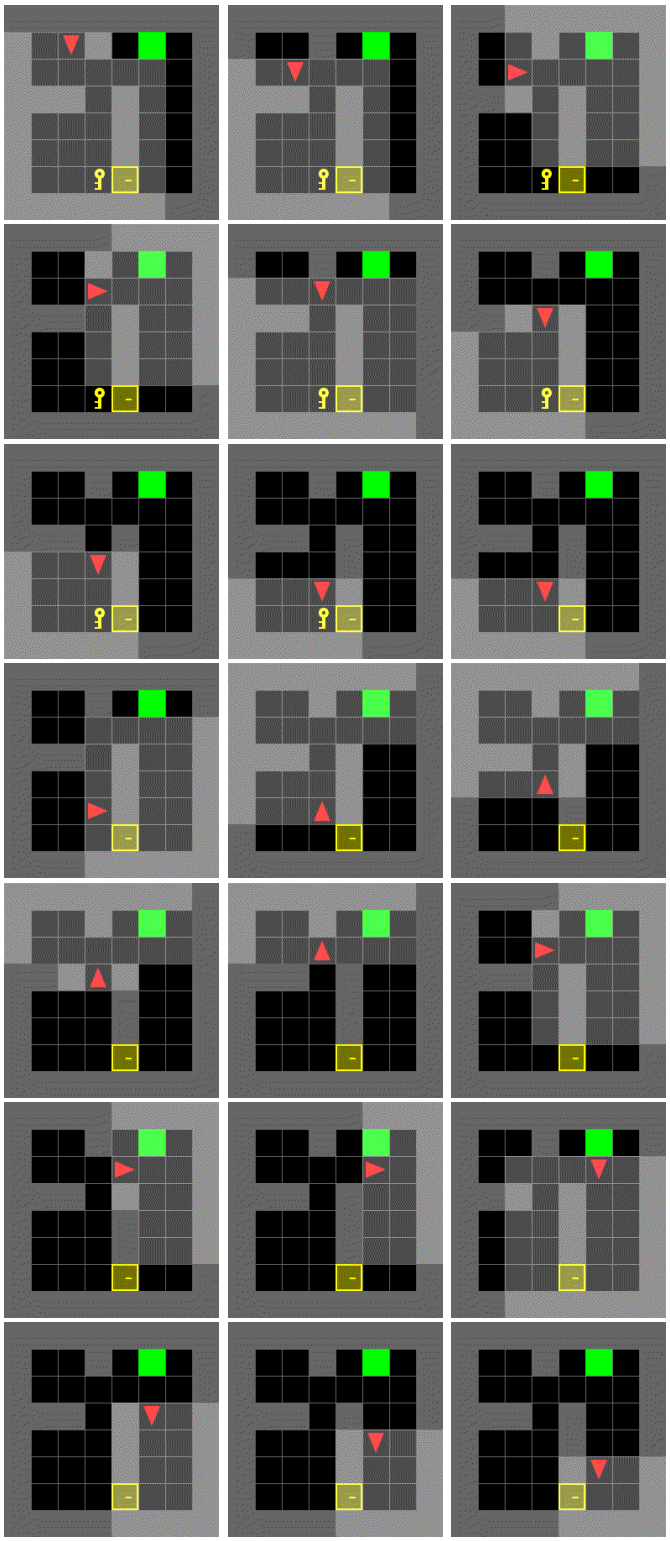


Fig. 9. Agent's motion through the 8x8-direct environment - From left to right and continue.

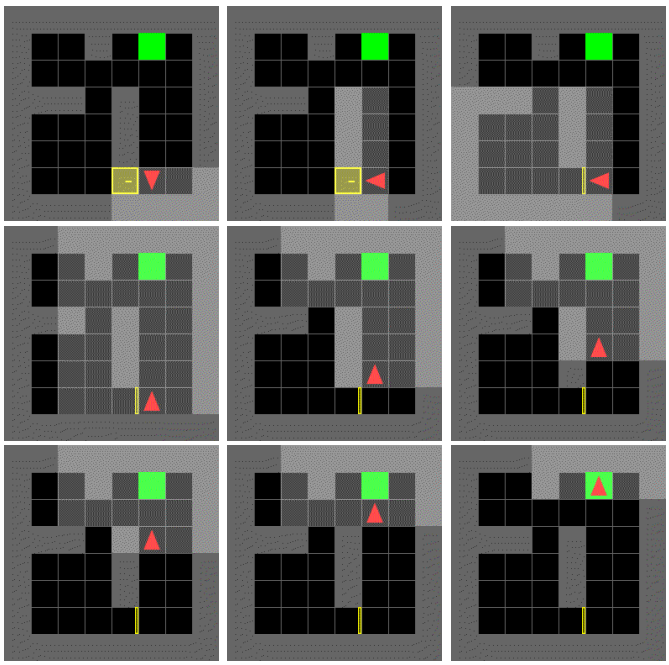


Fig. 10. Agent's motion through the 8x8-direct environment - From left to right and continue.