# Solving Door-Key Problem Using Dynamic Programming

Baoqian Wang
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA, 92093
Email:bawang@ucsd.edu

***Abstract*—This project uses dynamic programming to solve a specific motion planning problem, the Door-Key problem. In particular, given a 2-D grid map, in which there is an agent, door, key and target position, dynamic programming is used to navigate the agent to reach the target position with the minimum cost, i.e., number of actions. The promising performance of the dynamic programming is demonstrated in 7 different test scenarios.**

## I. INTRODUCTION

Motion planning is a popular research topic in the area of robotics, which has many real applications both in civilian and military areas such as precision agriculture , search and rescue, etc. It deals with planning the motion of a robotics to complete a task while satisfying some constraints and achieves optimal performance, which is the focus of this project.

The Door-Key problem we considered in this project is a typical motion planning problem. In particular, the agent in this problem is navigated to reach the target using minimum actions under the environment where there are key, door, walls, etc. Despite the fact that the Door-Key problem is a specific problem, it can be extended and adapted to other similar motion planning problems, especially path planning of robot in grid environments

Various of algorithms have been developed to solve the motion planning problem. The first type of algorithm is the optimal control, in particular, the motion planning is formulated as optimal control problem. Given the knowledge of the environment and the motion model, the problem can be solved using optimal control algorithms such as dynamic programming. The second type of algorithm is to use the reinforcement learning to solve the motion planning. Particularly, the reinforcement learning does not to know the environment and motion model in advance, it can directly approximate the value function of the motion planning or the motion model, environment model. In this project, we use the dynamic programming algorithm to solve the Door-Key problem, which is efficient and simple to implement.

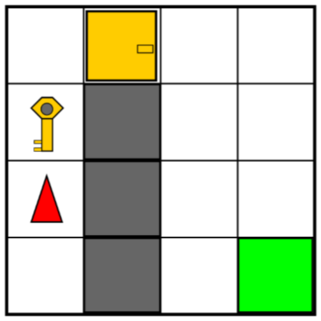
The rest of this paper is organized as follows. Section II formulates the Door-Key problem. Section III presents the technical approaches to solve the problem. The performance of the approaches are evaluated comprehensively in Section **??**. Section **??** concludes the paper with a brief summary.

## II. PROBLEM FORMULATION

In this section, the Door-Key problem is formulated by first describing the operating environment and then modeling the problem using Markov Decision Process (MDP).

### A. Environment

The environment of the Door-Key problem is a 2-D grid map (see Figure 1 as an illustration). In the environment, there is an agent (red triangle), key, door, target position (green area), and walls (grey areas). The agent can choose different actions including turn left, turn right, pick up the key, open the door, and move forward. The goal of the agent is to reach the target position with the minimum cost, while satisfying some constraints including that the door can only be opened with a key and the agent can not be in wall areas.



(a)

Figure 1. Illustration of the Doo-Key environment

### B. Markov Decision Process

Based on the description of the environment, the Door-Key problem can be formulated using the Markov Decision Process (MDP). Typically, a MDP problem is specified by a tuple, MDP $(\mathcal{S}, \mathcal{A}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma)$

- $S$: state space of the agent. In particular, $S = \{s | s = [x, y, k, d, o]\}$, where $\forall x \in \{0, 1, ..., w\}$ and $\forall y \in \{0, 1, ..., h\}$ represent the location of the agent in the map ($w$ and $h$ are the width and height of the map, respectively, $x = 0, y = 0$ indicates the top-left corner of the map), $k$ takes values 0 or 1 indicating whether the agent carries the key (0 for no, 1 for yes ), $d$ takes values 0 or 1 indicating whether the door is open or not (0 for no, 1 for yes ), $o$ takes values from $[1,0],[0,1],[-1,0],[0,-1]$ representing $east, south, west, north$ direction, respectively.

- $A$: action space of the agent. $A = \{a|\forall a \in \{MF, TL, TR, PK, UD\}$, where $MF, TL, TR, PK, UD$ represent actions "move forward", "turn left", "turn right", "pick up the key", "unlock the door".
- $p_0$: distribution of initial state of the agent. $P$: $S \rightarrow [0,\ 1]$. This project considers the deterministic Door-Key problem, which means $p_0(s_0) = 1, p_0(s) = 0, \forall s \neq s_0$.
- $p_f$: motion model of the agent, $s_{t+1} \sim p_f(s_t, a_t)$, which is described as follows, given $s_t = [x_t, y_t, k_t, d_t, o_t]$
  - if $a_t$=$MF$, $s_{t+1}$=$[x_{t+1}$,$y_{t+1}$,$k_t$,$d_t$,$o_t]$, where $[x_{t+1}, y_{t+1}] = [x_t, y_t] + o_t$
  - if $a_t$=$TL$, $s_{t+1}$=$[x_t$,$y_t$,$k_t$,$d_t$,$o_{t+1}]$, where $o_{t+1} = R(\frac{\pi}{2})o_t$
  - if $a_t$=$TR$, $s_{t+1}$=$[x_t$,$y_t$,$k_t$,$d_t$,$o_{t+1}]$, where $o_{t+1} = R(-\frac{\pi}{2})o_t$
  - if $a_t$=$PK$, $s_{t+1}$=$[x_t$,$y_t$,$k_{t+1}$,$d_t$,$o_t]$, where $k_{t+1} = 1$
  - if $a_t$=$UD$, $s_{t+1}$=$[x_t$,$y_t$,$k_t$,$d_{t+1}$,$o_t]$, where $d_{t+1} = 1$
- $T$: planning horizon. As the worst case of the optimal plan is to go through all the states, $T$ is set to the total number of possible states of the agent.
- $q$: terminal state cost, which is set to 0.
- $l(s_t, a_t)$: stage cost, which indicates the cost of taking action $a_t$ at state $s_t$. In particular, given the state $s_t = [x_t, y_t, k_t, d_t, o_t]$,
  - if $[x_t, y_t] \in \mathcal{C}_{obs}$, where $\mathcal{C}_{obs}$ is the wall area, $l(s_t, a_t) = \infty, \forall a_t \in \mathcal{A}$.
  - if $[x_t, y_t] \in \mathcal{C}_{free}$, where $\mathcal{C}_{free}$ is the free area,
    - if $a_t$=$MF$,
      $l(s_t, a_t) = \infty, if[x_{t+1}, y_[t+1] \in \mathcal{C}_{obs}$,
      $l(s_t, a_t) = 1, if[x_{t+1}, y_[t+1] \in \mathcal{C}_{free}$.
    - if $a_t$=$TL$,
      $l(s_t, a_t) = 1$
    - if $a_t$=$TR$,
      $l(s_t, a_t) = 1$
    - if $a_t$=$PK$,
      $l(s_t, a_t) = 1, if[x_t, y_t] = [x_k, y_k], k_t = 0, d_t = 0,\ l(s_t, a_t) = \infty, otherwise.$
    - if $a_t$=$UD$, $l(s_t, a_t) = 1, if[x_t, y_t] = [x_d, y_d], k_t = 1, d_t = 0,\ l(s_t, a_t) = \infty, otherwise$
- $\gamma$: discount factor, which is set to 1.

Given the formulated MDP, we aim to solve the following problem,

$$a^*_{0:T-1} = \underset{a_{0:T-1}}{\arg\min} V_0(\mathbf{s}_0) \tag{1}$$

where $V_0(\mathbf{s}_0) = \mathfrak{q} + \sum_{t=0}^{T-1} \ell(\mathbf{s}_t, a_t)$

## III. Technical Approaches

This section presents the dynamic programming approach to solve the Problem 1.

To apply the dynamic programming, we convert the original problem to a deterministic shortest path planning problem. In particular, each possible state of the agent is regarded as a node. Then a cost matrix $c_{i,j}$ is used to store the cost from node $i$ to node $j$. In particular, if there is an action in action space that can enable state $i$ to go to state $j$, the cost from $i$ to $j$ is then 1. Otherwise, the cost is infinity. The details of the algorithm to generate the cost matrix is summarized as follows,

```
Algorithm 1: Get Cost Matrix
  Input: MDP $(\mathcal{S}, \mathcal{A}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma)$, $env$
  Output: Optimal Action Sequence $a^*_{0:T-1}$ and
          Value Function $V_0$
  // Step 1: Initialize cost matrix
1 $c_{i,j} = \infty, \forall i, j \in S$
2 for $s_i \in \mathcal{S} \backslash \{S_o\}$ do
3     for $s_j \in \mathcal{S} \backslash \{S_o\}$ do
4         if $i$==$j$ then
5             $c_{i,j}$=0
6         for $a_k \in \mathcal{A}$ do
7             if $s_j == p_f(s_i, a_k)$ then
8                 $c_{i,j}$=1
9     Return Cost matrix $c_{i,j}$
```

After obtaining the cost matrix, the dynamic programming approach can be used to find the shortest path. In particular, the input of the algorithm are the cost matrix $c_{i,j}$, MDP $(\mathcal{S}, \mathcal{A}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma)$, and environment information including the position of key $[x_k, y_k]$, door $[x_d, y_d]$, target $[x_g, y_g]$. The first step is to initialize the terminal state cost $V_T(s) = \mathfrak{q}$, $\forall s \in S_T$, where $S_T = \{[x_T, y_T, k_T, d_T, o_T] | [x_T, y_T] = [x_g, y_g], \forall [k_T, d_T] \in \{[1,1],[1,0],[0,0]\}, o_T \in \{[1,0],[0,1],[-1,0],[0,-1]\}\}$. It indicates that the terminal state is that the position of the agent equals to the target position while the door, key, orientation can take different values. But one rule is that the door can not be open if the agent does not have the key. The second step is to iterate backwards in time, in particular, in each time step, calculate $Q_t(i,j)$ for each pair $(i,j), \forall s_i, s_j \in \mathcal{S}$. Then the value function $V_t(i)$ can be obtained by taking the minimal of the $Q_t(i,j)$ over $j$ and the associated $s_j$ will be the optimal next state at time step $t$. If the value functions of two successive time steps are equal, then the iteration is terminated. The details of dynamic programming is described in the following

Note that the optimal policy obtained from the above dynamic programming approach is the sequence of the states, additional procedures are needed to obtain the optimal action sequence using the following algorithm.

```
Algorithm 2: Dynamic Programming
    Input: Cost Matrix $c_{i,j}$, MDP
          $(\mathcal{S}, \mathcal{A}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma)$, $env$
    Output: Optimal State Sequence $s^*_{0:T-1}$ and Value
            Function $V_0$
    // Step 1: Initialize terminal
       state cost
1   $V_T(s) = \mathfrak{q}, \quad \forall s \in S_T$ // $S_T$ is the
       terminal state set
    // Step 2: Iterate backwards
2   for $t = (T-1) : 0$ do
3       $Q_t(s_i, s_j) = c(i,j) + V_{t+1}(s_j), \quad \forall s_i, s_j \in \mathcal{S}$
4       $V_t(s_i) = \min_{s_j \in \mathcal{S}} Q_t(s_i, s_j), \quad \forall s_i, s_j \in \mathcal{S}$
5       $\pi^*_t(s_i) = \arg\min_{s_j \in \mathcal{A}} Q_t(s_i, s_j), \quad \forall s_i, s_j \in \mathcal{S}$
6       if $V_t(s) = V_{t+1}(s), \ \forall s \in \mathcal{S}$ then
7           break
8   Obtain $s^*_{0:T-1}$ by following $\pi^*_t(s)$ with $s_0$
9   Return Optimal State Sequence $s^*_{0:T-1}$ and Value
      Function $V_0(s_0)$
```

```
Algorithm 3: Convert Optimal State Sequence to
Optimal Action Sequence
    Input: Optimal State Sequence $s^*_{0:T-1}$, MDP
          $(\mathcal{S}, \mathcal{A}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma)$, $env$
    Output: Optimal Action Sequence $a^*_{0:T-1}$
1   a=[]
    // Create a list
2   for successive $s_i, s_j \in s_{0:T-1}$ do
3       for $a_k \in \mathcal{A}$ do
4           if $s_j == p_f(s_i, a_k)$ then
5               a.append($a_k$)
6               break
7   Return Optimal Action Sequence $a^*_{0:T-1}$
```

## IV. Results

In this section, we test the proposed dynamic programming approach on seven different environments. For each environment, we show the optimal action sequence, policy function and value function. Note that the optimal action sequence is represented by numbers with the following mapping between numbers and actions, $0 \rightarrow MF$, $1 \rightarrow TL$, $2 \rightarrow TR$, $3 \rightarrow PK$, $4 \rightarrow UD$.

### A. *Environment: doorkey-5x5-normal*

The first environment is shown in Figure 2. The optimal action sequence is {1,3,2,4,0,0,2,0}. The value function and policy are shown in Figure 3. As the agent orientation, key, door are involved in the state space. It is hard to visualize the value and policy function in a 2-D grid map. Therefore, table are used to show value function and policy function of first 16 states of severl time steps. As we can see from the value function, at each time step, each state has a specific value which indicates the cost to reach the target from that state. The value will keep updating and finally converges at specific time step. For the policy function, at each time step, each state is mapped to the state of the next time step. If the states of two successive time steps are same, it means no action is taken. As the time goes, an optimal state sequence can be obtained. Note that in the value function table, the value 999999 indicates $\infty$.

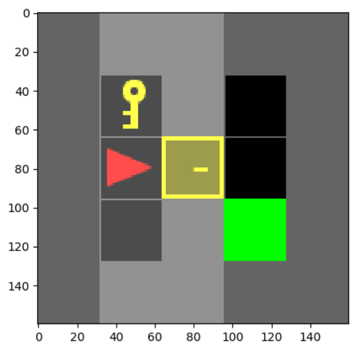
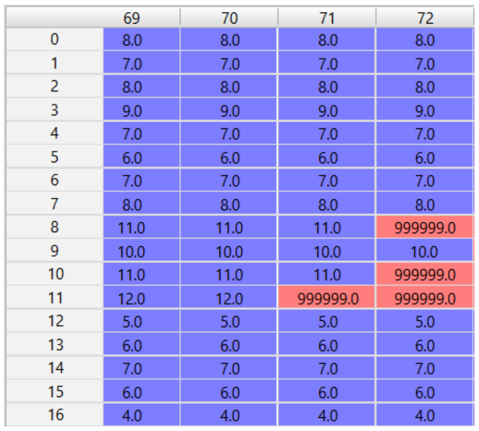
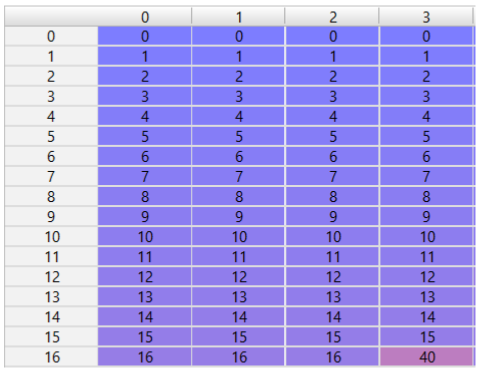


Figure 2. Doorkey-5x5-normal.



| | 69 | 70 | 71 | 72 |
|---|---|---|---|---|
| 0 | 8.0 | 8.0 | 8.0 | 8.0 |
| 1 | 7.0 | 7.0 | 7.0 | 7.0 |
| 2 | 8.0 | 8.0 | 8.0 | 8.0 |
| 3 | 9.0 | 9.0 | 9.0 | 9.0 |
| 4 | 7.0 | 7.0 | 7.0 | 7.0 |
| 5 | 6.0 | 6.0 | 6.0 | 6.0 |
| 6 | 7.0 | 7.0 | 7.0 | 7.0 |
| 7 | 8.0 | 8.0 | 8.0 | 8.0 |
| 8 | 11.0 | 11.0 | 11.0 | 999999.0 |
| 9 | 10.0 | 10.0 | 10.0 | 10.0 |
| 10 | 11.0 | 11.0 | 11.0 | 999999.0 |
| 11 | 12.0 | 12.0 | 999999.0 | 999999.0 |
| 12 | 5.0 | 5.0 | 5.0 | 5.0 |
| 13 | 6.0 | 6.0 | 6.0 | 6.0 |
| 14 | 7.0 | 7.0 | 7.0 | 7.0 |
| 15 | 6.0 | 6.0 | 6.0 | 6.0 |
| 16 | 4.0 | 4.0 | 4.0 | 4.0 |

(a)



| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 11 |
| 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 40 |

(b)

Figure 3. a)Value function b) Policy function (Doorkey-5x5-normal)

## B. *Environment: doorkey-6x6-normal*

The environment is shown in Figure 4. The optimal action sequence is {0,2,3,0,0,0,2,0,4,0,0,2,0,0,0}. The value function and policy function are shown in Figure 5.

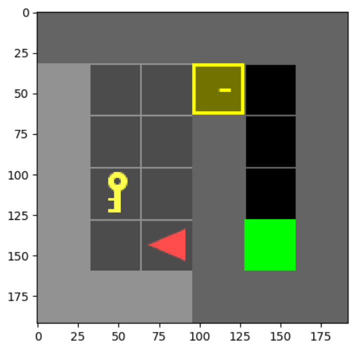
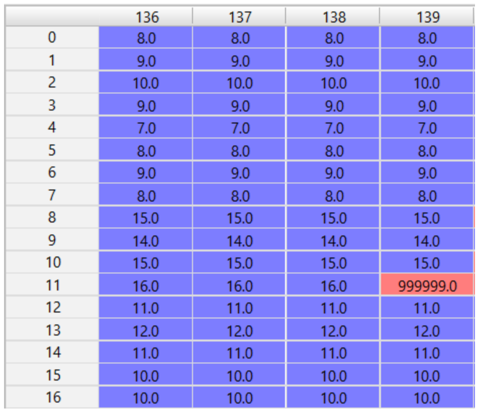
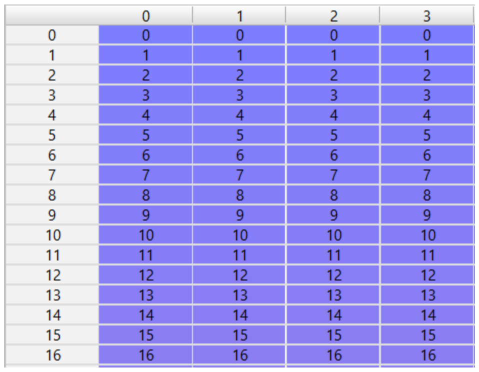


Figure 4. Doorkey-6x6-normal.



| | 136 | 137 | 138 | 139 |
|---|---|---|---|---|
| 0 | 8.0 | 8.0 | 8.0 | 8.0 |
| 1 | 9.0 | 9.0 | 9.0 | 9.0 |
| 2 | 10.0 | 10.0 | 10.0 | 10.0 |
| 3 | 9.0 | 9.0 | 9.0 | 9.0 |
| 4 | 7.0 | 7.0 | 7.0 | 7.0 |
| 5 | 8.0 | 8.0 | 8.0 | 8.0 |
| 6 | 9.0 | 9.0 | 9.0 | 9.0 |
| 7 | 8.0 | 8.0 | 8.0 | 8.0 |
| 8 | 15.0 | 15.0 | 15.0 | 15.0 |
| 9 | 14.0 | 14.0 | 14.0 | 14.0 |
| 10 | 15.0 | 15.0 | 15.0 | 15.0 |
| 11 | 16.0 | 16.0 | 16.0 | 999999.0 |
| 12 | 11.0 | 11.0 | 11.0 | 11.0 |
| 13 | 12.0 | 12.0 | 12.0 | 12.0 |
| 14 | 11.0 | 11.0 | 11.0 | 11.0 |
| 15 | 10.0 | 10.0 | 10.0 | 10.0 |
| 16 | 10.0 | 10.0 | 10.0 | 10.0 |

(a)



| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 11 |
| 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 16 |

(b)

Figure 5. a)Value function b) Policy function (Doorkey-6x6-normal)

## C. *Environment: doorkey-6x6-direct*

The envionment is shown in Figure 6.The optimal action sequence is {1,1,0,0}. The value function and policy function are shown in Figure 7.

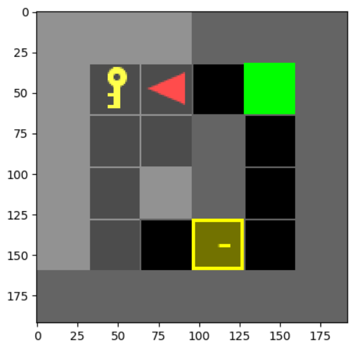
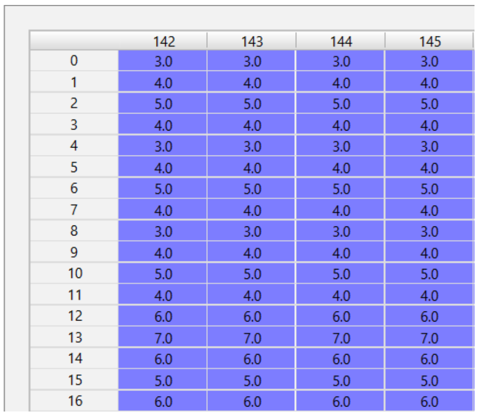
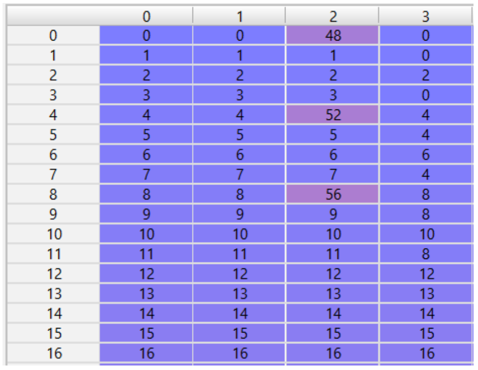


Figure 6. Doorkey-6x6-direct.



| | 142 | 143 | 144 | 145 |
|---|---|---|---|---|
| 0 | 3.0 | 3.0 | 3.0 | 3.0 |
| 1 | 4.0 | 4.0 | 4.0 | 4.0 |
| 2 | 5.0 | 5.0 | 5.0 | 5.0 |
| 3 | 4.0 | 4.0 | 4.0 | 4.0 |
| 4 | 3.0 | 3.0 | 3.0 | 3.0 |
| 5 | 4.0 | 4.0 | 4.0 | 4.0 |
| 6 | 5.0 | 5.0 | 5.0 | 5.0 |
| 7 | 4.0 | 4.0 | 4.0 | 4.0 |
| 8 | 3.0 | 3.0 | 3.0 | 3.0 |
| 9 | 4.0 | 4.0 | 4.0 | 4.0 |
| 10 | 5.0 | 5.0 | 5.0 | 5.0 |
| 11 | 4.0 | 4.0 | 4.0 | 4.0 |
| 12 | 6.0 | 6.0 | 6.0 | 6.0 |
| 13 | 7.0 | 7.0 | 7.0 | 7.0 |
| 14 | 6.0 | 6.0 | 6.0 | 6.0 |
| 15 | 5.0 | 5.0 | 5.0 | 5.0 |
| 16 | 6.0 | 6.0 | 6.0 | 6.0 |

(a)



| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 48 | 0 |
| 1 | 1 | 1 | 1 | 0 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 0 |
| 4 | 4 | 4 | 52 | 4 |
| 5 | 5 | 5 | 5 | 4 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 4 |
| 8 | 8 | 8 | 56 | 8 |
| 9 | 9 | 9 | 9 | 8 |
| 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 8 |
| 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 16 |

(b)

Figure 7. a)Value function b) Policy function (Doorkey-6x6-direct).

## D. *Environment: doorkey-6x6-shortcut*

The envionment is shown in Figure 8. The optimal action sequence is {3, 1, 1, 4, 0, 0}. The value function and policy function are shown in Figure 9.

## E. *Environment: doorkey-8x8-direct*

The envionment is shown in Figure 10. The optimal action sequence is {1,0,0,0}. The value function and policy function are shown in Figure 11.

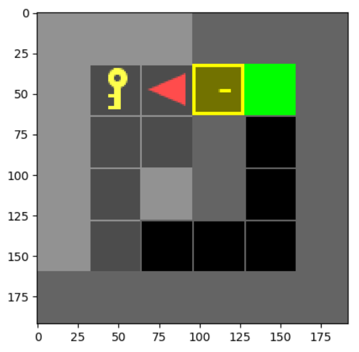
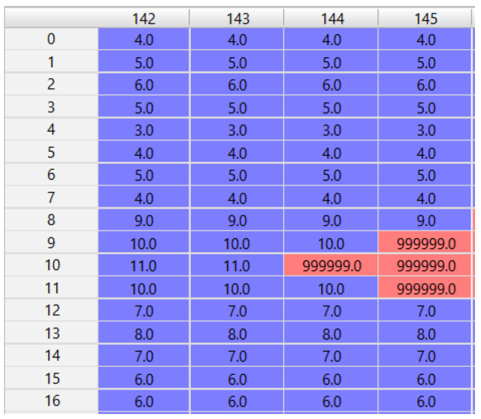
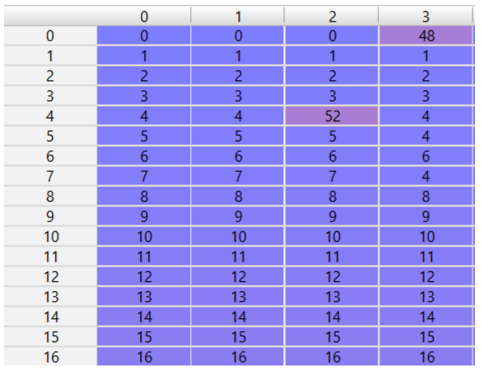


Figure 8.  Doorkey-6x6-shortcut.



|  | 142 | 143 | 144 | 145 |
|---|---|---|---|---|
| 0 | 4.0 | 4.0 | 4.0 | 4.0 |
| 1 | 5.0 | 5.0 | 5.0 | 5.0 |
| 2 | 6.0 | 6.0 | 6.0 | 6.0 |
| 3 | 5.0 | 5.0 | 5.0 | 5.0 |
| 4 | 3.0 | 3.0 | 3.0 | 3.0 |
| 5 | 4.0 | 4.0 | 4.0 | 4.0 |
| 6 | 5.0 | 5.0 | 5.0 | 5.0 |
| 7 | 4.0 | 4.0 | 4.0 | 4.0 |
| 8 | 9.0 | 9.0 | 9.0 | 9.0 |
| 9 | 10.0 | 10.0 | 10.0 | 999999.0 |
| 10 | 11.0 | 11.0 | 999999.0 | 999999.0 |
| 11 | 10.0 | 10.0 | 10.0 | 999999.0 |
| 12 | 7.0 | 7.0 | 7.0 | 7.0 |
| 13 | 8.0 | 8.0 | 8.0 | 8.0 |
| 14 | 7.0 | 7.0 | 7.0 | 7.0 |
| 15 | 6.0 | 6.0 | 6.0 | 6.0 |
| 16 | 6.0 | 6.0 | 6.0 | 6.0 |

(a)



|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 48 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 52 | 4 |
| 5 | 5 | 5 | 5 | 4 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 4 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 11 |
| 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 16 |

(b)

Figure 9.  a)Value function b) Policy function (Doorkey-6x6-shortcut)

### F. *Environment: doorkey-8x8-normal*

The environment is shown in Figure 12. The optimal action sequence is {1, 0, 2, 0, 0, 0, 2, 3, 2, 0, 0, 0, 0, 2, 4, 0, 0, 0, 2, 0, 0, 0, 0, 0}. The value function and policy function are shown in Figure 13.

### G. *Environment: doorkey-8x8-shortcut*

The environment is shown in Figure 14. The optimal action sequence is {2, 0, 1, 3, 1, 0, 0, 4, 0, 0}. The value function and policy function are shown in Figure 15.

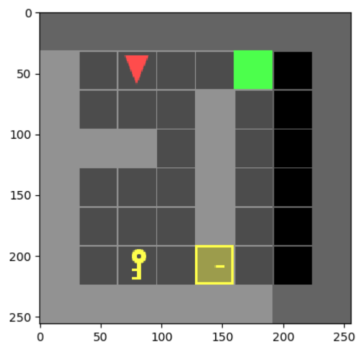
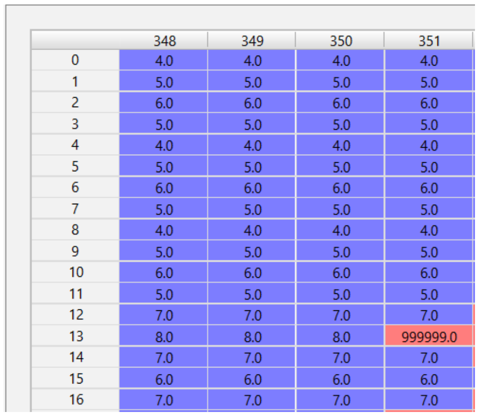
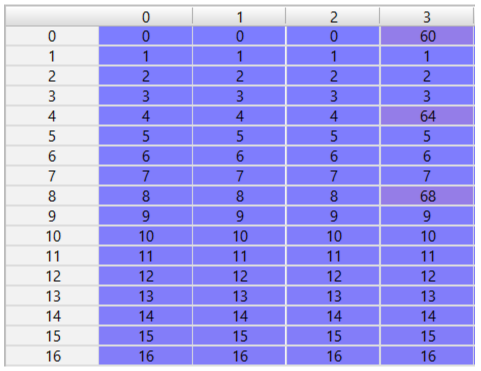


Figure 10.  Doorkey-8x8-direct.



|  | 348 | 349 | 350 | 351 |
|---|---|---|---|---|
| 0 | 4.0 | 4.0 | 4.0 | 4.0 |
| 1 | 5.0 | 5.0 | 5.0 | 5.0 |
| 2 | 6.0 | 6.0 | 6.0 | 6.0 |
| 3 | 5.0 | 5.0 | 5.0 | 5.0 |
| 4 | 4.0 | 4.0 | 4.0 | 4.0 |
| 5 | 5.0 | 5.0 | 5.0 | 5.0 |
| 6 | 6.0 | 6.0 | 6.0 | 6.0 |
| 7 | 5.0 | 5.0 | 5.0 | 5.0 |
| 8 | 4.0 | 4.0 | 4.0 | 4.0 |
| 9 | 5.0 | 5.0 | 5.0 | 5.0 |
| 10 | 6.0 | 6.0 | 6.0 | 6.0 |
| 11 | 5.0 | 5.0 | 5.0 | 5.0 |
| 12 | 7.0 | 7.0 | 7.0 | 7.0 |
| 13 | 8.0 | 8.0 | 8.0 | 999999.0 |
| 14 | 7.0 | 7.0 | 7.0 | 7.0 |
| 15 | 6.0 | 6.0 | 6.0 | 6.0 |
| 16 | 7.0 | 7.0 | 7.0 | 7.0 |

(a)



|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 60 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 64 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 |
| 8 | 8 | 8 | 8 | 68 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 11 |
| 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 16 |

(b)

Figure 11.  a)Value function b) Policy function (Doorkey-8x8-shortcut).

## V. Discussion

The test results show that the proposed dynamic programming has promising performance and can find the optimal action sequence in Door-Key problem. The most important feature of the proposed approach is that it converts the original problem to a deterministic shortest path planning problem, which simplifies the implementation of the algorithm. However, the efficiency of the algorithm

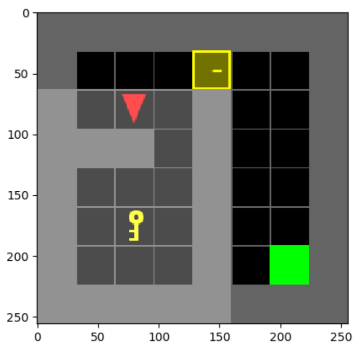
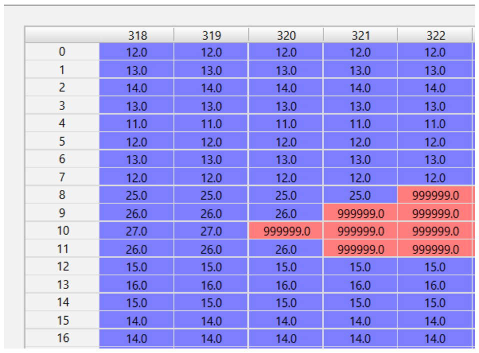
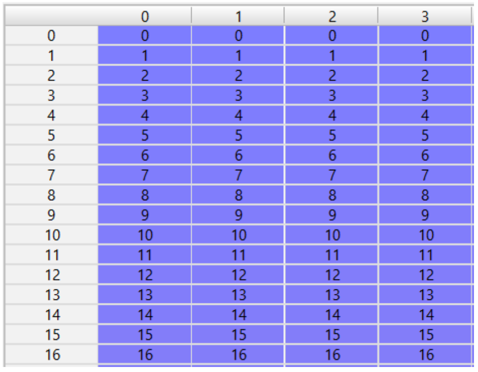


Figure 12.    Doorkey-8x8-normal.



|  | 318 | 319 | 320 | 321 | 322 |
|---|---|---|---|---|---|
| 0 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 |
| 1 | 13.0 | 13.0 | 13.0 | 13.0 | 13.0 |
| 2 | 14.0 | 14.0 | 14.0 | 14.0 | 14.0 |
| 3 | 13.0 | 13.0 | 13.0 | 13.0 | 13.0 |
| 4 | 11.0 | 11.0 | 11.0 | 11.0 | 11.0 |
| 5 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 |
| 6 | 13.0 | 13.0 | 13.0 | 13.0 | 13.0 |
| 7 | 12.0 | 12.0 | 12.0 | 12.0 | 12.0 |
| 8 | 25.0 | 25.0 | 25.0 | 25.0 | 999999.0 |
| 9 | 26.0 | 26.0 | 26.0 | 999999.0 | 999999.0 |
| 10 | 27.0 | 27.0 | 999999.0 | 999999.0 | 999999.0 |
| 11 | 26.0 | 26.0 | 26.0 | 999999.0 | 999999.0 |
| 12 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 |
| 13 | 16.0 | 16.0 | 16.0 | 16.0 | 16.0 |
| 14 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 |
| 15 | 14.0 | 14.0 | 14.0 | 14.0 | 14.0 |
| 16 | 14.0 | 14.0 | 14.0 | 14.0 | 14.0 |

(a)



|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 11 |
| 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 16 |

(b)

Figure 13.    a)Value function b) Policy function (Doorkey-8x8-normal)

may be affected by additional cost matrix generation and optimal state sequence conversion, which can be improved by directly applying the dynamic programming instead of converting to deterministic shortest path planning problem.

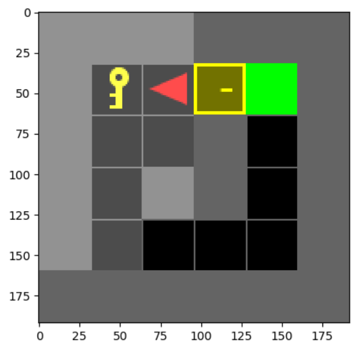
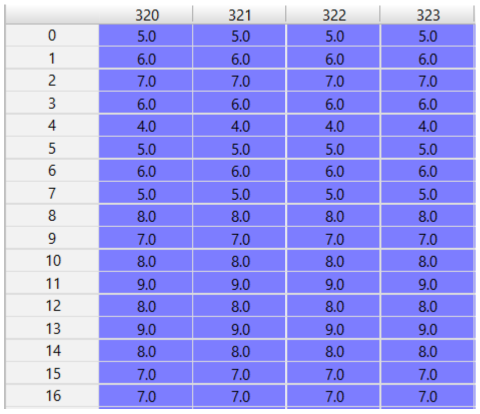
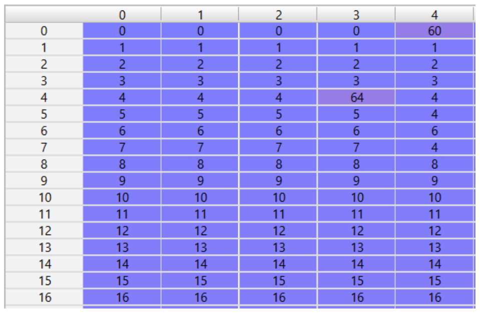


Figure 14.    Doorkey-8x8-shortcut.



|  | 320 | 321 | 322 | 323 |
|---|---|---|---|---|
| 0 | 5.0 | 5.0 | 5.0 | 5.0 |
| 1 | 6.0 | 6.0 | 6.0 | 6.0 |
| 2 | 7.0 | 7.0 | 7.0 | 7.0 |
| 3 | 6.0 | 6.0 | 6.0 | 6.0 |
| 4 | 4.0 | 4.0 | 4.0 | 4.0 |
| 5 | 5.0 | 5.0 | 5.0 | 5.0 |
| 6 | 6.0 | 6.0 | 6.0 | 6.0 |
| 7 | 5.0 | 5.0 | 5.0 | 5.0 |
| 8 | 8.0 | 8.0 | 8.0 | 8.0 |
| 9 | 7.0 | 7.0 | 7.0 | 7.0 |
| 10 | 8.0 | 8.0 | 8.0 | 8.0 |
| 11 | 9.0 | 9.0 | 9.0 | 9.0 |
| 12 | 8.0 | 8.0 | 8.0 | 8.0 |
| 13 | 9.0 | 9.0 | 9.0 | 9.0 |
| 14 | 8.0 | 8.0 | 8.0 | 8.0 |
| 15 | 7.0 | 7.0 | 7.0 | 7.0 |
| 16 | 7.0 | 7.0 | 7.0 | 7.0 |

(a)



|  | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 60 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 64 | 4 |
| 5 | 5 | 5 | 5 | 5 | 4 |
| 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 | 4 |
| 8 | 8 | 8 | 8 | 8 | 8 |
| 9 | 9 | 9 | 9 | 9 | 9 |
| 10 | 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 11 | 11 | 11 | 11 |
| 12 | 12 | 12 | 12 | 12 | 12 |
| 13 | 13 | 13 | 13 | 13 | 13 |
| 14 | 14 | 14 | 14 | 14 | 14 |
| 15 | 15 | 15 | 15 | 15 | 15 |
| 16 | 16 | 16 | 16 | 16 | 16 |

(b)

Figure 15.    a)Value function b) Policy function (Doorkey-8x8-shortcut)