# Project Report

1. Introduction

In this project we were required to come up with an algorithm that can make an agent find the goal autonomously given a 2D gird environment. In this environment, there are different kind of blocks: wall (where agent is not allowed to be on), door (where the agent can open with key), key and goal. This project can be seen as the simplest version of robot motion planning (with certain algorithm) where we are given a environment and trying to maneuver the robot from the starting position to the goal which is the most common seen problem in robotics. It can not only give us a brief understanding of how dynamic programming works on robot motion planning but also a basic structure of how to do the motion planning. In the future, after we learn other more sophisticated algorithms, we can use them to solve more complicated problem.

2. Problem Statement

This project can be seen as a two different Markov Decision Processes, will define them separately in this part.

In part A:

State space X is a n*n*4*2*2 matrix. N equals to the size of the map (since the map is always a square with height equals to width). The 4 represent 4 different directions (with 0 is right, 1 is down, 2 is left and 3 is up). The rest of two twos represent the state of key and door, whether we have picked up the key and whether we have unlocked the door.

The control space U is defined as U a t*n*n*4*2*2 matrix. The t at the front represents the time horizon and the rest is the same as state space. In side the control space stores the optimal control at every time steps for every state.

The motion model f(x) is defined as follow:

$$f(x) = \begin{cases} 0 & \text{move forward} \\ 1 & \text{turn left} \\ 2 & \text{turn right} \\ 3 & \text{pick up the key} \\ 4 & \text{unlock the door} \end{cases}$$

The stage cost l(x, u) is defined as follow:

$$l(x,u) = \begin{cases} 0 & \text{next step is at goal position} \\ \infty & \text{if } f(x) = 1 \text{ and (next cell is wall or} \\ & \text{next cell is unlocked door)} \\ \infty & \text{if } f(x) = 3 \text{ and ( next cell is not key or} \\ & \text{we already have key} \\ \infty & \text{if } f(x) = 4 \text{ and ( next cell is not door or} \\ & \text{next cell is unlocked door} \\ & \text{next cell is locked door but we are not carrying} \\ 1 & \text{otherwise} \end{cases}$$

Terminal cost q(x) is defined as follow:

$$q(x) = \begin{cases} 0 & \text{at goal position} \\ \infty & \text{otherwise} \end{cases}$$

The time horizon is defined as T = height * width * 4 * 2 * 2

For part B:

State space X is a 3*3*10*10*4*2*2*2 matrix. The first two threes represent the three possible goal positions and key positions respectively. The last two twos represent the locking situation of the two doors. And the rest is the same as Part A, except this time all the map is the same size (8*8), so we set n = 10.

The control space U is defined as U a t*3*3*10*10*4*2*2*2 matrix. The t at the front represents the time horizon and the rest is the same as state space. Inside the control space stores the optimal control at every time steps for every state.

The motion model, stage cost and terminal are defined as the same in part A.

The time horizon is defined as T = 3*3*10*10*4*2*2*2. However, it might make the control space too big with a lot of redundant information, we decided to set T = 50. Since after we times of computation we can find that it terminates much early before 50 times.

3. Technical Approach

Generally, we are solving both part with dynamic programming so all the set up and problem solving will below will build up around dynamic programming. The pseudocode of how dynamic programming works is as follow:

```
1: Input: MDP ($\mathcal{X}, \mathcal{U}, f(x), T, \ell, q$)
2: $V_T(x) = q(x)$,  $\forall x \in \mathcal{X}$
3: For $t = (T-1) \dots 0$ do
4:         $Q_t(x,u) = \ell(x,u) + E[V_{t+1}(f(x))]$,  $\forall x \in \mathcal{X}, u \in \mathcal{U}$
5:         $V_t(x) = \min_{u \in \mathcal{U}(x)} Q_t(x,u)$,  $\forall x \in \mathcal{X}$
6:         $\pi_t(x) = \arg\min_{u \in \mathcal{U}(x)} Q_t(x,u)$,  $\forall x \in \mathcal{X}$
7: return policy $\pi_{0:T-1}$ and value function $V_0$
```

And each term is defined as follow:

| | |
|---|---|
| $\mathcal{X}$ — | state space |
| $\mathcal{U}$ — | constrol space |
| $f(x)$ — | motion model |
| $T$ — | time horizon |
| $\ell$ — | stage cost |
| $q$ — | terminal cost |
| $\pi_t$ — | control for time t for every state |
| $V_t$ — | v function for time t for every state |

In both part we can break our algorithm into 3 parts: initialization, problem solving and find the control sequence.

Part A

1) Initialization

   In this part, we were just initializing the state space, control space and the time horizon. Also, we need to initialize the matrix that store the result of v function, as we see in the pseudocode, for both current step and the last step.

   At the same time, we will also set up the motion model as we had discussed in last section.

2) Problem solving

   In this part we are basically using for loop to run through every time step, every state in state space and their every possible control in control space to find the optimal control for every time step in every state. However, we can not simply just let it run through the time horizon since that would be a lot of redundant computation. We set a criterion if the v function of last time is equal to this time step for every state, then the computation terminates.

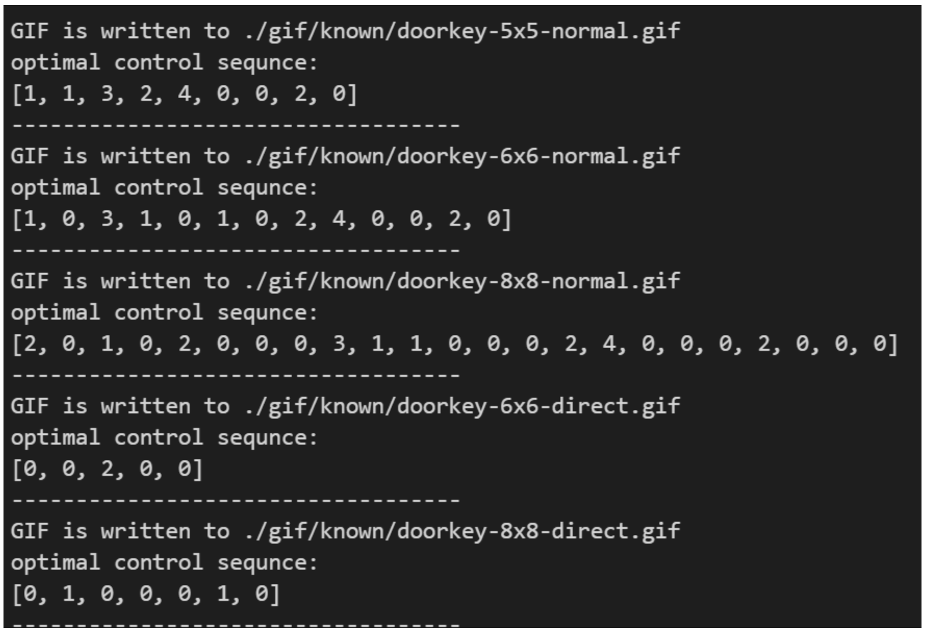
3) Find the control sequence

After the we find the control, we need to trace from starting point to find the best control sequence. This is the place where part A and part B has some difference. For part A we just need to start with agent's initial position and direction (with door locked, not carrying keys). But for part B, is a little bit different. We need to first extract the goal position, key position, and initial door condition, since it differs from different cases. Then we do the same things as part A. After that we just using for loop to trace from starting position, step by step, until we reach the goal position eventually.

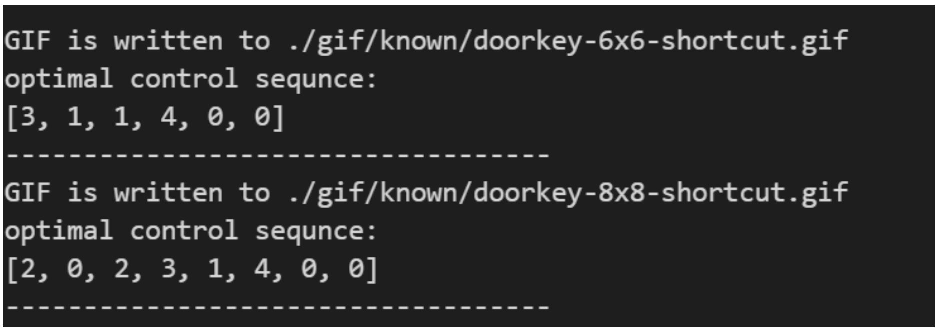
4. Results

Part A:

This is a picture of all the control sequence for every situation in part A

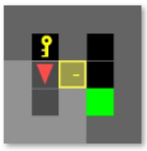
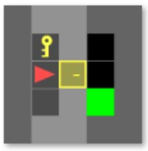
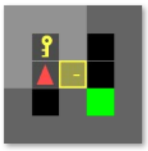
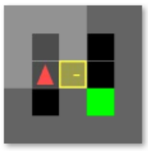
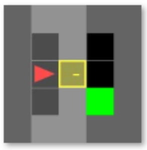
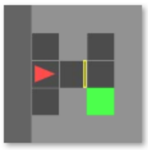
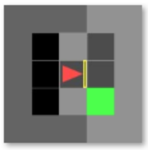
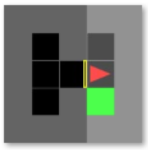
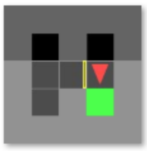
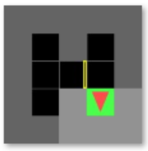


```
GIF is written to ./gif/known/doorkey-5x5-normal.gif
optimal control sequnce:
[1, 1, 3, 2, 4, 0, 0, 2, 0]
----------------------------------
GIF is written to ./gif/known/doorkey-6x6-normal.gif
optimal control sequnce:
[1, 0, 3, 1, 0, 1, 0, 2, 4, 0, 0, 2, 0]
----------------------------------
GIF is written to ./gif/known/doorkey-8x8-normal.gif
optimal control sequnce:
[2, 0, 1, 0, 2, 0, 0, 0, 3, 1, 1, 0, 0, 0, 2, 4, 0, 0, 0, 2, 0, 0, 0]
----------------------------------
GIF is written to ./gif/known/doorkey-6x6-direct.gif
optimal control sequnce:
[0, 0, 2, 0, 0]
----------------------------------
GIF is written to ./gif/known/doorkey-8x8-direct.gif
optimal control sequnce:
[0, 1, 0, 0, 0, 1, 0]
----------------------------------
```

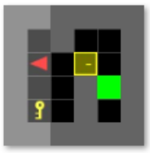
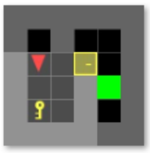
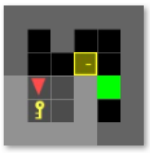
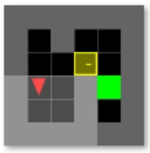
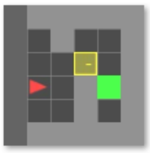
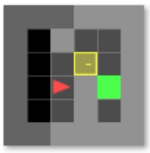
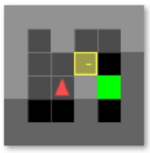
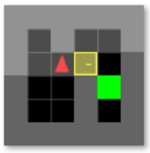
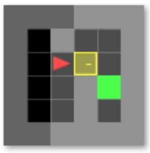
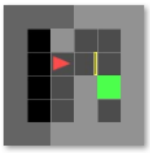
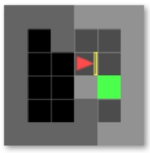
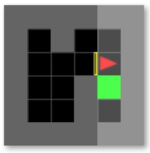
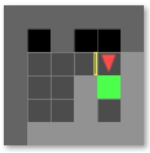
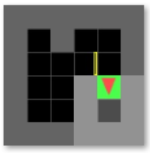


```
GIF is written to ./gif/known/doorkey-6x6-shortcut.gif
optimal control sequnce:
[3, 1, 1, 4, 0, 0]
----------------------------------
GIF is written to ./gif/known/doorkey-8x8-shortcut.gif
optimal control sequnce:
[2, 0, 2, 3, 1, 4, 0, 0]
----------------------------------
```

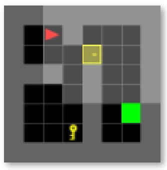
Case 1 (5x5-normal):

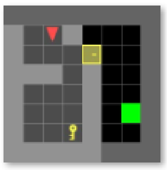

frame_0.png

frame_1.png

frame_2.png

frame_3.png

frame_4.png

frame_5.png

frame_6.png

frame_7.png

frame_8.png

frame_9.png

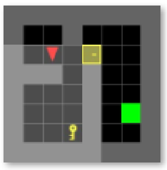
Case 2 (6x6-normal):

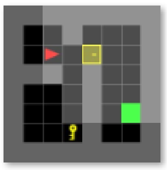

frame_0.png

frame_1.png

frame_2.png

frame_3.png

frame_4.png

frame_5.png

frame_6.png

frame_7.png

frame_8.png

frame_9.png

frame_10.png

frame_11.png

frame_12.png

frame_13.png

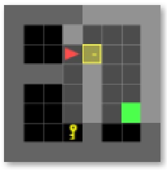
Case 3 (8x8-normal):

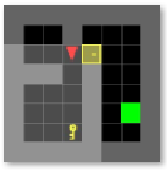

frame_0.png

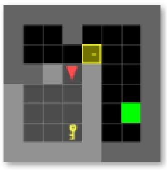

frame_1.png

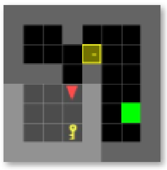

frame_2.png

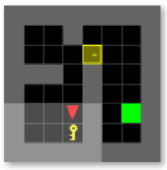

frame_3.png

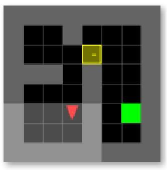

frame_4.png

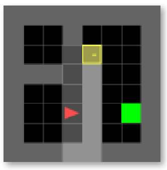

frame_5.png

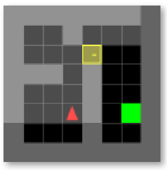

frame_6.png

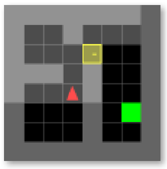

frame_7.png

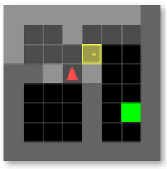

frame_8.png

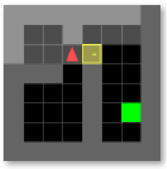

frame_9.png

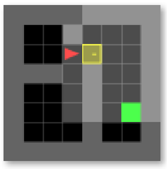

frame_10.png

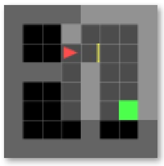

frame_11.png

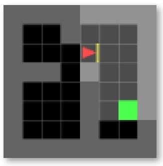

frame_12.png

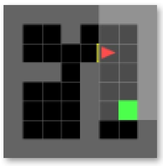

frame_13.png

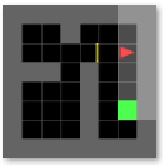

frame_14.png

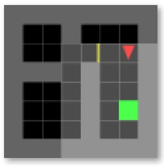

frame_15.png

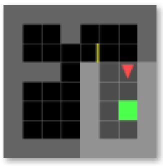

frame_16.png

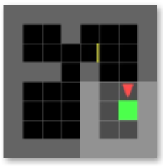

frame_17.png

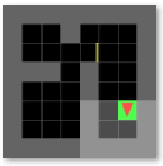

frame_18.png


frame_19.png

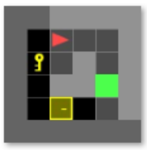

frame_20.png

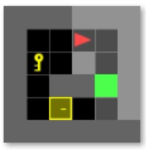

frame_21.png

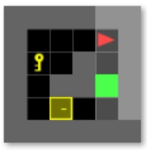

frame_22.png

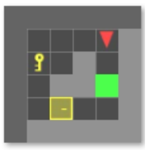

frame_23.png

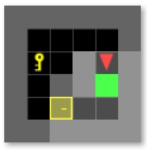
Case 4 (6x6-direct):

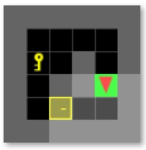

frame_0.png


frame_1.png

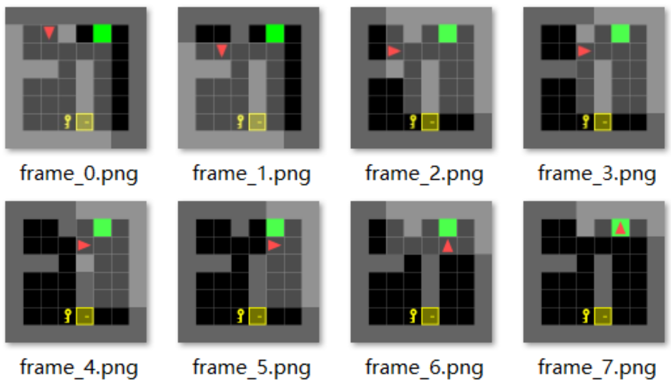

frame_2.png


frame_3.png

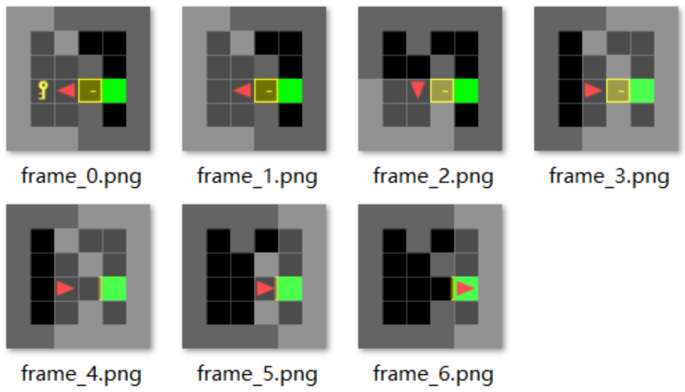

frame_4.png


frame_5.png

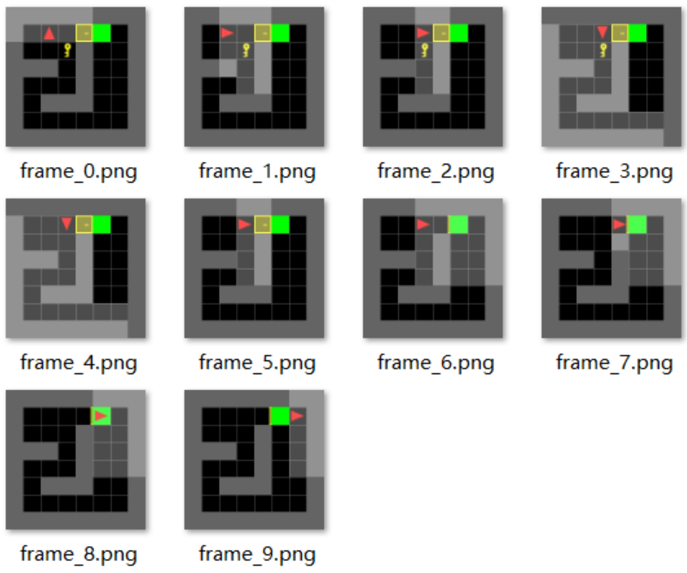
Case 5 (8x8-direct):



Case 6 (6x6-shortcut):



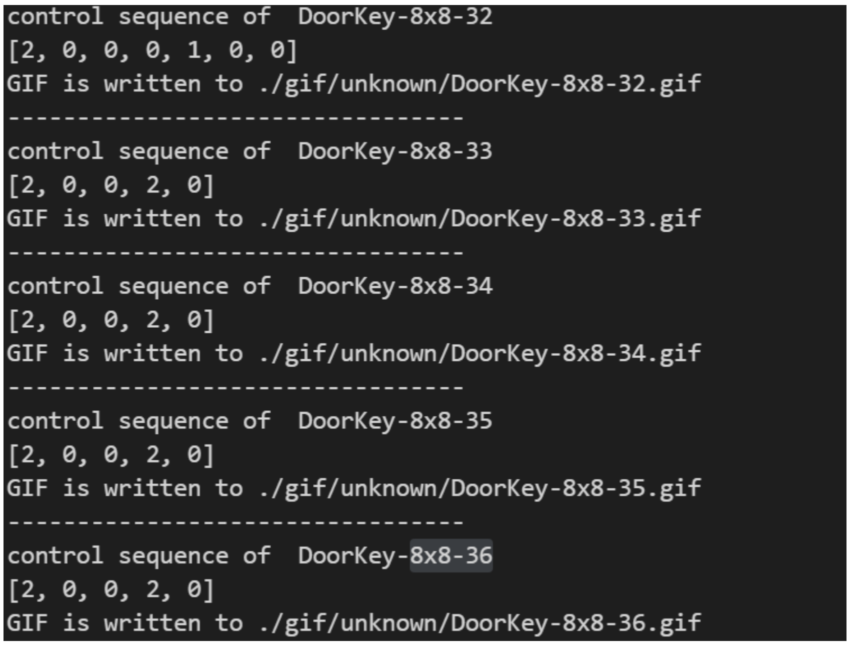
Case 7 (8x8-shortcut):



As we can see here, all of the cases can find the shortest trajectory from the starting point to the end point perfectly

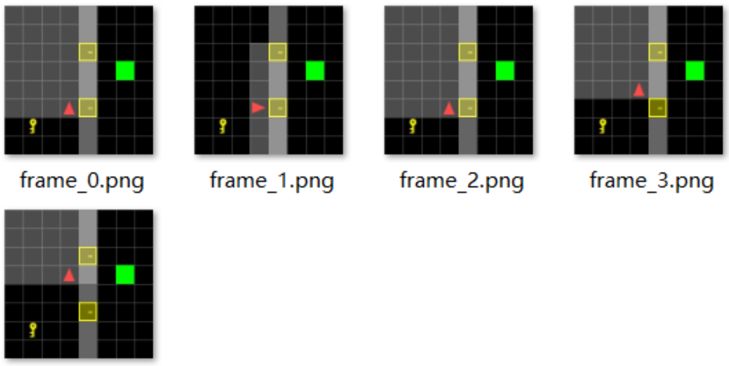
Part B:

Since it is going to take too much time to run every cases in part B, we decided only run part to verify our algorithm.

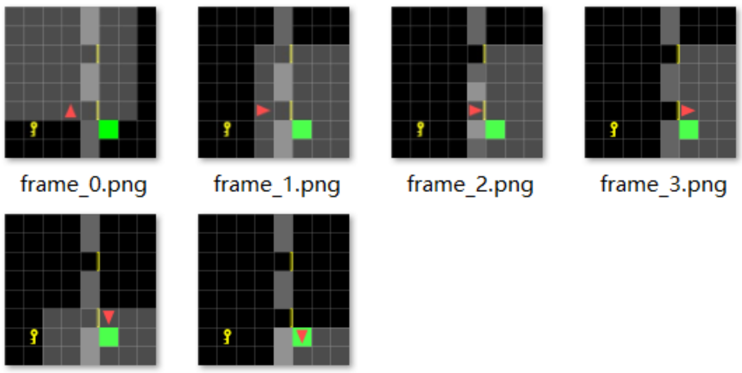


```
control sequence of  DoorKey-8x8-32
[2, 0, 0, 0, 1, 0, 0]
GIF is written to ./gif/unknown/DoorKey-8x8-32.gif
--------------------------------
control sequence of  DoorKey-8x8-33
[2, 0, 0, 2, 0]
GIF is written to ./gif/unknown/DoorKey-8x8-33.gif
--------------------------------
control sequence of  DoorKey-8x8-34
[2, 0, 0, 2, 0]
GIF is written to ./gif/unknown/DoorKey-8x8-34.gif
--------------------------------
control sequence of  DoorKey-8x8-35
[2, 0, 0, 2, 0]
GIF is written to ./gif/unknown/DoorKey-8x8-35.gif
--------------------------------
control sequence of  DoorKey-8x8-36
[2, 0, 0, 2, 0]
GIF is written to ./gif/unknown/DoorKey-8x8-36.gif
```

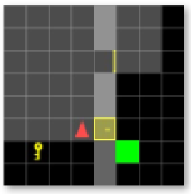
Case 1 (8x8-32):

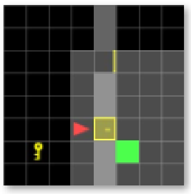


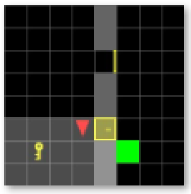
Case 2 (8x8-33):

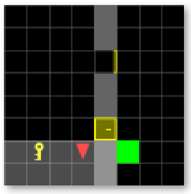


Case 3 (8x8-34):

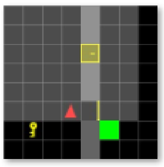

frame_0.png

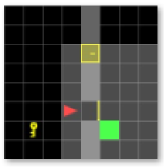

frame_1.png

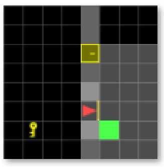

frame_2.png

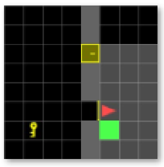

frame_3.png

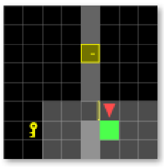
Case 4 (8x8-35):

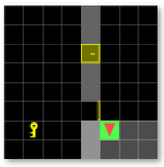

frame_0.png


frame_1.png

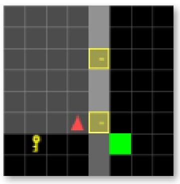

frame_2.png

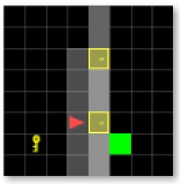

frame_3.png

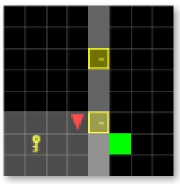

frame_4.png

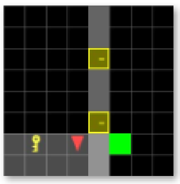

frame_5.png

Case 5 (8x8-36):


frame_0.png


frame_1.png


frame_2.png


frame_3.png

The results above show that in some of the cases we can find the shortest trajectory from the starting position to end position (like 33 and 35), but in some other cases we can't.

After observing the all the pictures and the control sequence, we can find that, if we follow the control sequence that we are able to reach the goal from the starting point and it is also the shortest trajectory. However, there is just one problem, that we lack of the motion of picking up the key and unlock the door. Therefore, we can extrapolate that our initialization and find control sequence part is correct, but the dynamic programming part has some minor problem. Specifically, I believe that the problem lies in the motion model and how we compute the v function for every motion.

Also, during the time when we were writing code and testing the code, we found that there is something I could improve about my code:

1) Data structure: For the state space and control space, we were using ndarray to store information. However, it is not that efficient under this circumstance. Therefore, we can try different kinds of data structure (like dictionary or nametuple) to achieve higher efficiency.

2) Class: We felt like doing everything with class might be a better choice since it might give us a cleaner look and higher efficiency also.