Dynamic Programming - Doorkey Problem

Rishabh Bhattacharya
Department of Mechanical & Aerospace Engineering
University of California, San Diego
ribhattacharya@ucsd.edu

***Index Terms*—dynamic programming, door-key problem, value function, optimal policy, Markov chains, Markov decision process, Deterministic shortest path**

I. Introduction

Path planning is a major area of emphasis for numerous robots, due to its importance in the mobility of any autonomous system. A robot must be able to plan its path in the wake of obstacles in its way, either by offline pre-computation or by online learning. As is obvious, online learning is computationally intensive, and it also requires the computations to be done efficiently to minimize delay between perception and control action.

If we have previous knowledge about the environment, a pre-computed policy seems desirable since almost no computation power is required for navigation. All we are required to do is lookup the corresponding state in the optimal policy and determine the control input. This is thus a perfect approach for path planning of robotic systems/environments which are deterministic in nature, and have discrete states and control inputs.

For this project, we use the gym-minigrid [1] package from Python to solve a doorkey problem. Our environments have an agent, walls, a key, a goal, and one (or two) doors. The objective is to have the agent reach the goal by collecting a key and opening a door, unless the door is already opened. Every action has a cost, and we want to reach the goal with the minimum cost possible. For part A, we have a collection of 7 distinct environments. We are required to compute optimal policies for each environment, and the use it to determine an optimal sequence for the agent to navigate the map and reach its goal. For part B, we have 36 8x8 environments, where the key can be present in any of 3 (pre-defined) locations and so can the goal. Also, there are two doors, which can be in any of open/close states. We are required to compute a SINGLE policy that works for all the 36 environments.

We begin by setting up the problem as a Markov Decision Process (MDP). We then define our state space, inputs, motion model and cost function to define our problem. We then use dynamic programming applied deterministic shortest path algorithm to compute our optimal policies.

II. Problem formulation

The problems (A & B) are formulated as a Markov Decision Process (MDP). The MDP for our problem is a Markov chain with a cost function defined by the tuple $(\mathcal{X}, \mathcal{U}, p_f, T, l, q, \gamma)$ where,

$$\begin{aligned}
\mathcal{X} &: \text{is a } \textbf{discrete}\text{/continuous set of states}\\
\mathcal{U} &: \text{is a } \textbf{discrete}\text{/continuous set of controls}\\
p_f &: \textbf{deterministic}\text{/stochastic motion model}\\
T &: \textbf{finite}\text{/infinite time horizon}\\
l(\mathbf{x}, \mathbf{u}) &: \text{function that specifies cost of control } \mathbf{u} \in \mathcal{U}\\
&\quad \text{to state } \mathbf{x} \in \mathcal{X}\\
q(\mathbf{x_T}) &: \text{terminal cost of being in state } \mathbf{x_T} \text{ at time } \mathbf{T}\\
\gamma &: \in [0, \mathbf{1}] \text{ is the discount factor}
\end{aligned}$$

and items in bold correspond to our problem formulation.

Assumptions:

1) No negative cycles assumption: Optimal path need not have more than $|\mathcal{V}|$ elements

Objective for part A: given discrete control inputs $\mathcal{U} = [\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}]$ and a mini-grid environment, compute the optimal policy to be used for autonomous navigation of the agent to reach a given goal position by collecting a key and unlocking a door.

Objective for part B: given discrete control inputs $\mathcal{U} = [\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}]$ and variable environment information (like variable key/goal position and multiple doors which can be both opened/closed), pre-compute the optimal policy to be used for autonomous navigation of the agent to reach a given goal position (single policy must work on all environment variations).

A. State space $\mathcal{X}$

Both of our problems have a discrete state space.
For **part A**, the state space is considered to be,

$$\mathcal{X}_{\mathcal{A}} = \{\text{agent}_x, \text{agent}_y, \text{agent}_{dir}, \text{key}_{stat}, \text{door}_{stat}\}$$

where,

agent_x	:agent x positions for 8x8 grid $\in [0 \cdots 7]$
agent_y	:agent y positions for 8x8 grid $\in [0 \cdots 7]$
agent_{dir}	:agent orientations (encoded in .py file) $\in [0 \cdots 3]$
key_{stat}	:key not carrying/carrying $\in [0, 1]$
door_{stat}	:door open/close $\in [0, 1]$

For **part B**, the state space is considered to be,

$$\mathcal{X}_{\mathcal{B}} = \{\text{agent}_x, \text{agent}_y, \text{agent}_{dir}, \text{key}_{stat}, \text{door1}_{stat}, \text{door2}_{stat}, \text{key}_{loc}, \text{goal}_{loc}\}$$

where,

agent_x	:agent x positions for 8x8 grid $\in [0 \cdots 7]$
agent_y	:agent y positions for 8x8 grid $\in [0 \cdots 7]$
agent_{dir}	:agent orientations (encoded in .py file) $\in [0 \cdots 3]$
key_{stat}	:key not carrying/carrying $\in [0, 1]$
door1_{stat}	:door1 open/close $\in [0, 1]$
door2_{stat}	:door2 open/close $\in [0, 1]$
key_{loc}	:key location out of 3 possible states $\in [0, 1, 2]$
goal_{loc}	:goal location out of 3 possible states $\in [0, 1, 2]$

B. *Input controls $\mathcal{U}$*

Given are a set of discrete input controls $\mathcal{U} = [\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}]$, which are used to move forward, turn left, turn right, pick key and unlock door respectively.

C. *Motion model p_f*

The motion model is given in (Alg. #1 0). We have implemented a deterministic motion model, which checks for invalid moves (where $p_f = 0$) and valid moves where probability of transition $p_f = 1$.

Algorithm 1 Motion Model Compute next state (x_{t+1}) given current state and input (x_t, u_t)

```
1: if at goal then $x_{t+1}$ = $x_t$                                  ▷ Stay if at goal
2: else if wall ahead and act == MF then $x_{t+1}$ = $x_t$ ▷ Don't move if wall ahead
3: else if (key not ahead or key already picked) and act == PK then $x_{t+1}$ = $x_t$   ▷ Don't pick key
4: else if (key not picked or door not ahead or door unlocked) and act == UD then $x_{t+1}$ = $x_t$   ▷ Don't unlock door
5: else if (key ahead and key not picked) or (door ahead and door locked) act == MF then $x_{t+1}$ = $x_t$   ▷ Don't move forward if key not picked or door is locked
6:
7: else move according to usual motion model i.e. MF, PK, UD, TR, TL
```

D. *Time horizon T*

This is a finite time horizon MDP, since there always exists a path to the goal, which is attained in finite time. The actual time depends on the environments and no. of steps required to reach the goal.

E. *Cost function $l(\mathbf{x}, \mathbf{u})$*

The cost function was developed (Alg #2 0) to have high costs for all invalid moves, to deter the optimal policy from choosing those inputs. All valid moves are given a cost = 1. Cost = 0 always at the goal.

Algorithm 2 Cost function Compute cost given current state and input (x_t, u_t)

```
1: if at goal then cost = 0                                  ▷ Stay if at goal
2: else if wall ahead and act == MF then cost = $\infty$   ▷ Invalid move
3: else if (key not ahead or key already picked) and act == PK then cost = $\infty$   ▷ Don't pick key
4: else if (key not picked or door not ahead or door unlocked) and act == UD then cost = $\infty$   ▷ Don't unlock door
5: else if (key ahead and key not picked) or (door ahead and door locked) act == MF then cost = $\infty$   ▷ Don't move forward if key not picked or door is locked
6:
7: else cost = 1
8: return cost
```

F. *Terminal cost $q(\mathbf{x_T})$*

Terminal cost $q(\mathbf{x_T})$ is the cost incurred at the final time step of the problem. We want our agent to be at the goal in the final time step, and nowhere else. This is why all the goal positions have $q = 0$, while all non-goal positions have $q = \infty$.

G. *Discount factor γ*

Discount factor $\gamma = 1$ for our implementation.

III. Technical Approach

We are trying to solve a finite time horizon optimal control problem for an MDP. We will use the Deterministic Shortest Path (DSP) algorithm, which is solved using Dynamic programming. The finite-horizon optimal control problem in an MDP $(\mathcal{X}, \mathcal{U}, p_f, T, \ell, \mathfrak{q}, \gamma)$ with initial state $\mathbf{x}$ at time t is:

$$\min_{\pi_{t:T-1}} V_t^{\pi}(\mathbf{x}) := \mathfrak{q}\left(\mathbf{x}_T\right) + \sum_{\tau=t}^{T-1} \ell\left(\mathbf{x}_\tau, \pi_\tau\left(\mathbf{x}_\tau\right)\right) \mid \mathbf{x}_t = \mathbf{x} \tag{1}$$

$$\text{s.t. } \mathbf{x}_{\tau+1} \sim p_f\left(\cdot \mid \mathbf{x}_\tau, \pi_\tau\left(\mathbf{x}_\tau\right)\right), \quad \tau = t, \ldots, T-1$$
$$\mathbf{x}_\tau \in \mathcal{X}, \pi_\tau\left(\mathbf{x}_\tau\right) \in \mathcal{U}$$

Our goal is to determine a policy that minimizes this value function. We use dynamic programming applied to the DSP approach (Alg. #3 0) for this problem.

Algorithm 3 Deterministic Shortest Path via Dynamic Programming

```
1: Input: vertices $\mathcal{V}$, start $s \in \mathcal{V}$, goal $\tau \in \mathcal{V}$, and costs $c_{ij}$ for $i, j \in \mathcal{V}$
2: T=$| \mathcal{V} | -1$
3: $V_T(\tau) = V_{T-1}(\tau) = \ldots = V_0(\tau) = 0$    ▷ goal had 0 terminal cost
4: $V_T(i) = \infty, \quad \forall i \in \mathcal{V} \backslash \{\tau\}$    ▷ non-goal positions initialized with $\infty$ terminal cost
5: $V_{T-1}(i) = c_{i,\tau}, \quad \forall i \in \mathcal{V} \backslash \{\tau\}$
6: $\pi_{T-1}(i) = \tau, \quad \forall i \in \mathcal{V} \backslash \{\tau\}$
7: for t=(T-2), ..., 0 do
8:     $Q_t(i,j) = c_{i,j} + V_{t+1}(j), \quad \forall i \in \mathcal{V} \backslash \{\tau\}, j \in \mathcal{V}$    ▷ $c_{i,j}$ derived from cost function, $V_{t+1}$ was calculated at previous time step
9:     $V_t(i) = \min_{j \in \mathcal{V}} Q_t(i,j), \quad \forall i \in \mathcal{V} \backslash \{\tau\}$
10:    $\pi_t(i) = \arg\min_{j \in \mathcal{V}} Q_t(i,j), \quad \forall i \in \mathcal{V} \backslash \{\tau\}$
11:    if $V_t(i) = V_{t+1}(i), \forall i \in \mathcal{V} \backslash \{\tau\}$ then
12:        break
13:
```

For every time iteration, we calculate the optimal input to all states. This is done by minimizing the stage cost and finding the optimal input. We say that we have converged to a solution when the value function becomes steady i.e. $V_t(i) = V_{t+1}(i), \forall i \in \mathcal{V} \backslash \{\tau\}$. We are thus thus said to have achieved the corresponding optimal policy.
After determining the policy for our entire state space, we run the policy through any given random environment to determine the specific optimal sequence.

IV. Results

The optimal control policy for each environment was successfully implemented for both part A and B. The computation time in part A is quite low (in milliseconds), due to 1024×5 computations at each time horizon (1024 states and 5 inputs). We see that majority of environments converge to the optimal V and π in around 10-15 time steps, with the larger 8x8 environment taking > 20 steps occasionally.

The computation time in part B is higher, owing to an increase in no. of states. The states have increased from 1024 to $1024 \times 2 \times 3 \times 3 = 1024 \times 18 = 18432$. This is an 18x increase in the number of computations per time step, which leads to higher processing requirements.

A. Part A

We compute the optimal policy and generate the sequence for successfully reaching the goal for each of the 7 environments. Shown below are plots of each environment, before and after applying the optimal control input sequence. (Figures 1 - 14)

B. Part B

We compute **ONE SINGLE** policy, which is supposed to work on each of the 36 environments. We have randomly

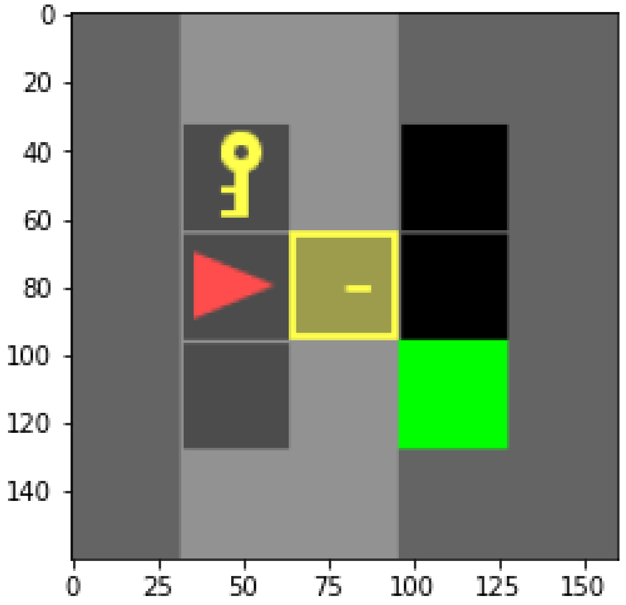

Fig. 1: **doorkey-5x5-normal. Optimal sequence is ['TL', 'PK', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF']**

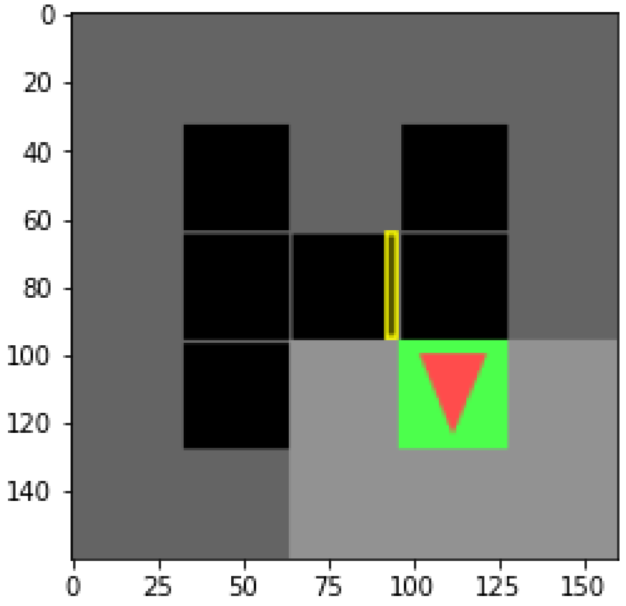

Fig. 2: **doorkey-5x5-normal reached goal**

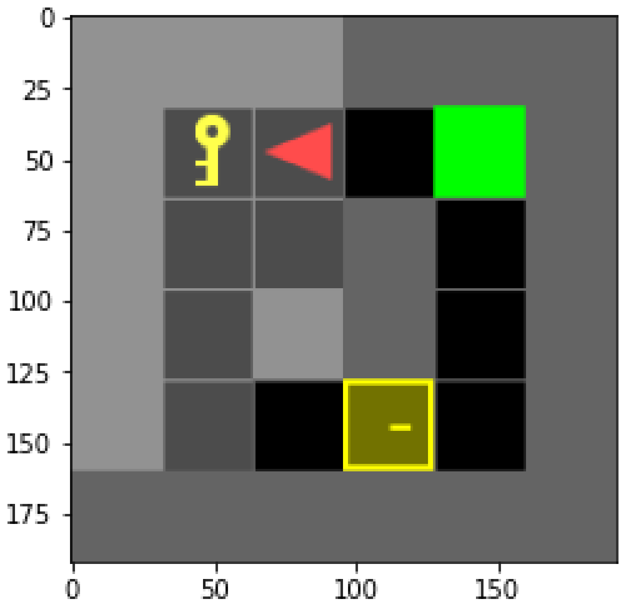

Fig. 3: **doorkey-6x6-direct. Optimal sequence is ['TL', 'TL', 'MF', 'MF']**

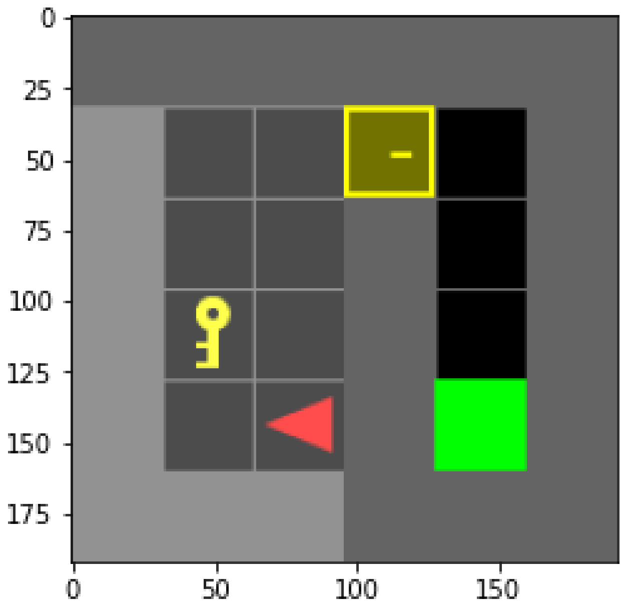

Fig. 5: **doorkey-6x6-normal. Optimal sequence is ['MF', 'TR', 'PK', 'MF', 'MF', 'MF', 'TR', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF']**

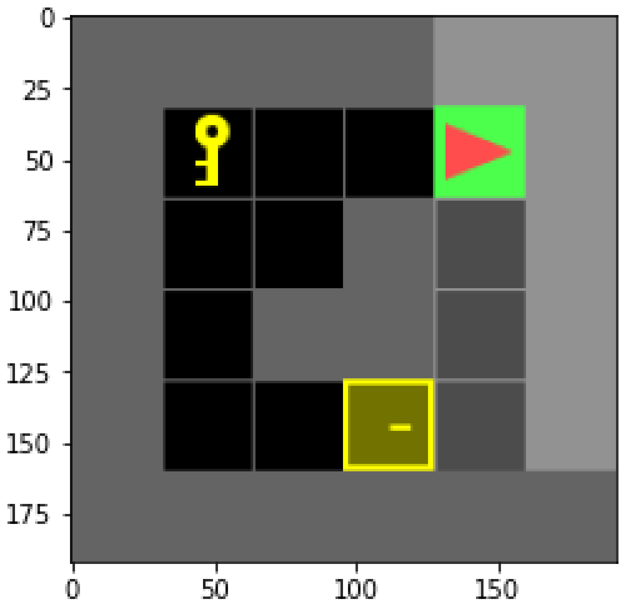

Fig. 4: **doorkey-6x6-direct reached goal**

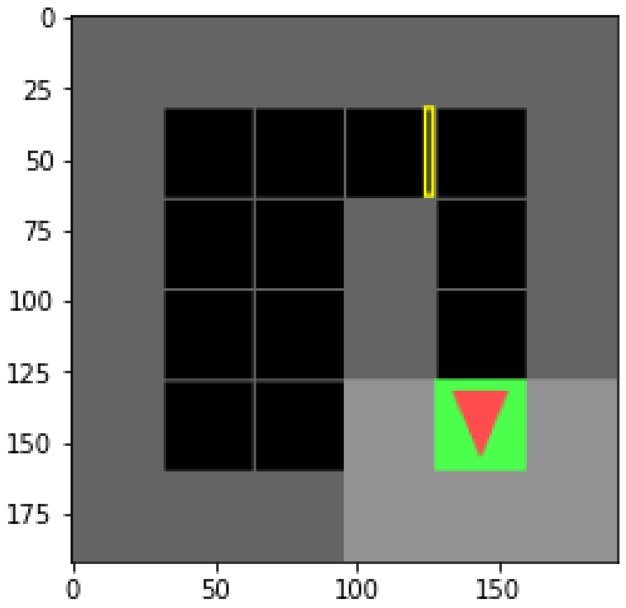

Fig. 6: **doorkey-6x6-normal reached goal**

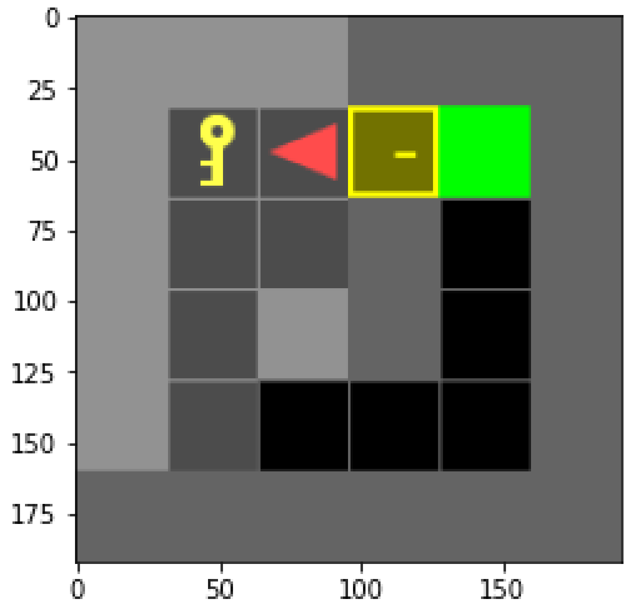

Fig. 7: **doorkey-6x6-shortcut. Optimal sequence is ['PK', 'TL', 'TL', 'UD', 'MF', 'MF']**

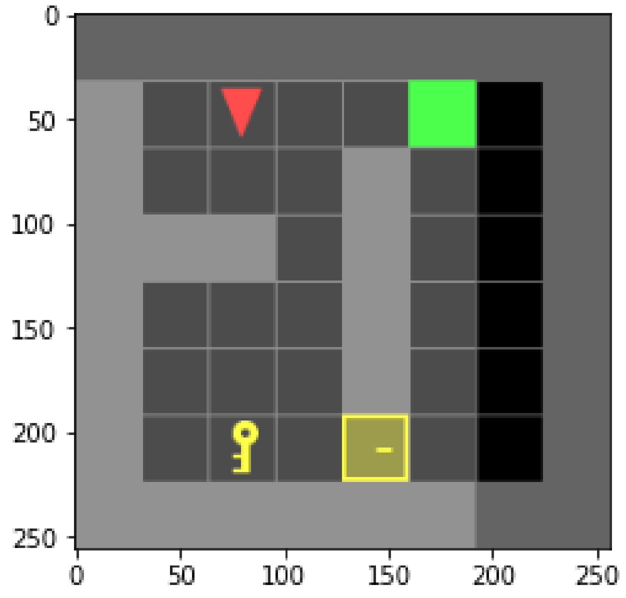

Fig. 9: **doorkey-8x8-direct. Optimal sequence is ['TL', 'MF', 'MF', 'MF']**

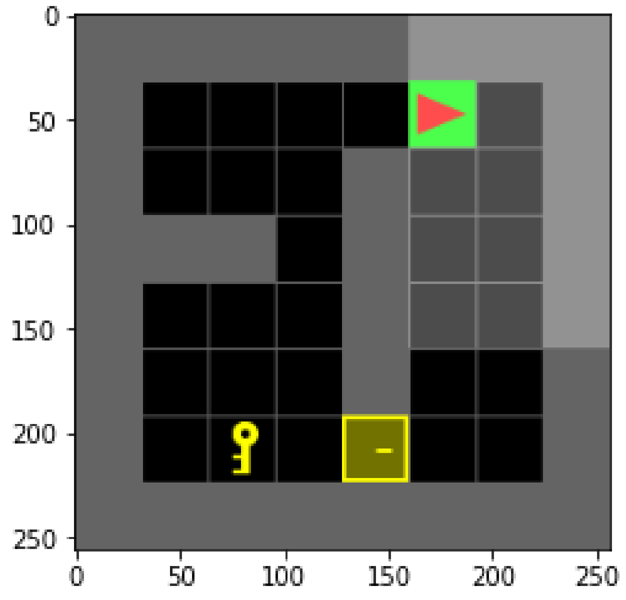

Fig. 10: **doorkey-8x8-direct reached goal**

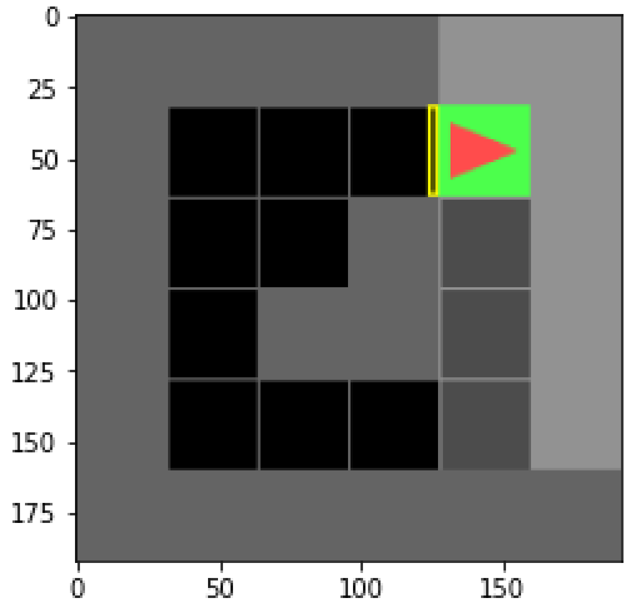

Fig. 8: **doorkey-6x6-shortcut reached goal**

selected some environments with different fundamental structures i.e. one door open, both doors open, both doors closed, shortcut, etc. (Figures 15 - 24). An interesting condition could be analysed for a few environments which have one door open and another one closed. Usually the agent would pick the key and unlock the door if the unlocked door happens to be the one nearest to the goal. But if the cost of PK/UD > MF, then for some cases it would prefer to go through the door that is farther away, rather than picking up the key and unlocking the door nearer to the goal.

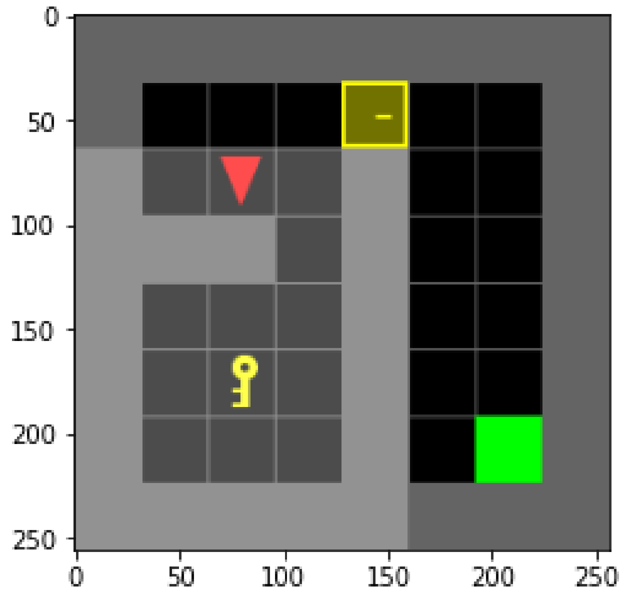
Fig. 11: **doorkey-8x8-normal. Optimal sequence is [’TL’, ’MF’, ’TR’, ’MF’, ’MF’, ’MF’, ’TR’, ’PK’, ’TR’, ’MF’, ’MF’, ’MF’, ’MF’, ’TR’, ’UD’, ’MF’, ’MF’, ’MF’, ’TR’, ’MF’, ’MF’, ’MF’, ’MF’, ’MF’]**

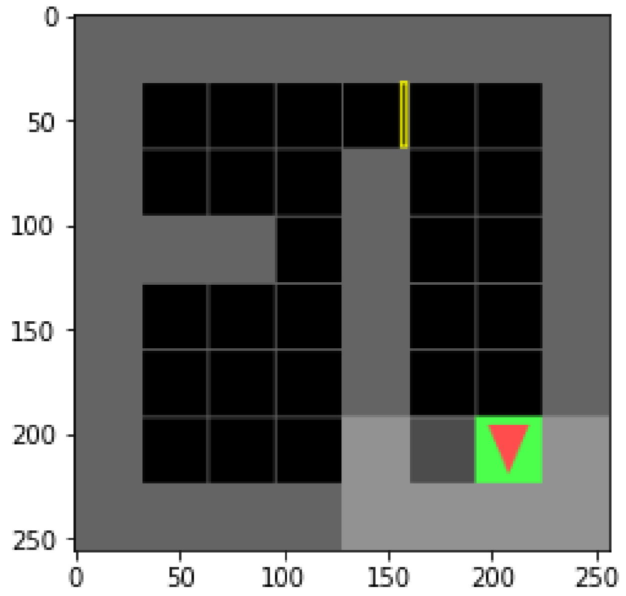
Fig. 12: **doorkey-8x8-normal reached goal**

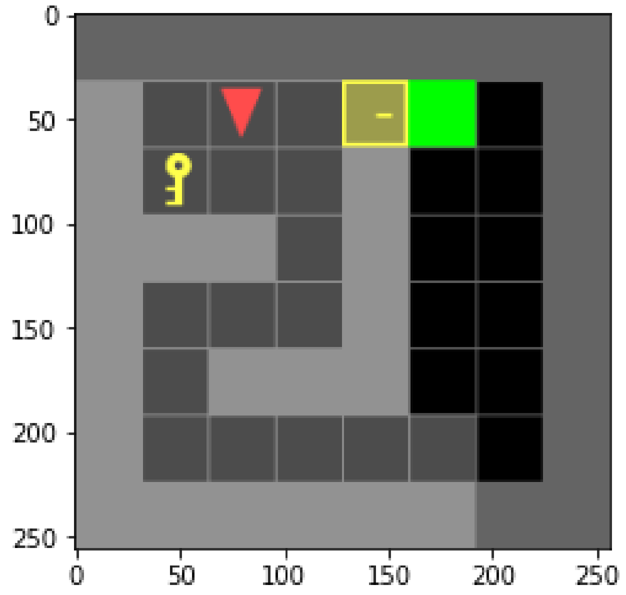
Fig. 13: **doorkey-8x8-shortcut. Optimal sequence is [’MF’, ’TR’, ’PK’, ’TR’, ’MF’, ’TR’, ’MF’, ’UD’, ’MF’, ’MF’]**

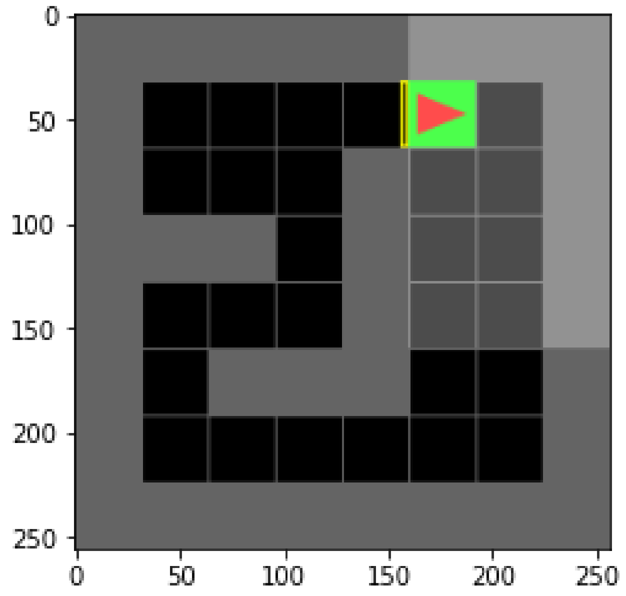
Fig. 14: **doorkey-8x8-shortcut reached goal**

V. Conclusions

We have successfully implemented Dynamic programming to calculate optimal policies for a range of environments. We also pre-computed a policy for an environment with changing key/goal positions and door statuses. Thus we realized that a plethora of problems can be solved and unknown variables can be dealt with by including them in our MDP state space. Even though we increase the state-space (and thus the computations significantly), we get a more robust optimal policy which can

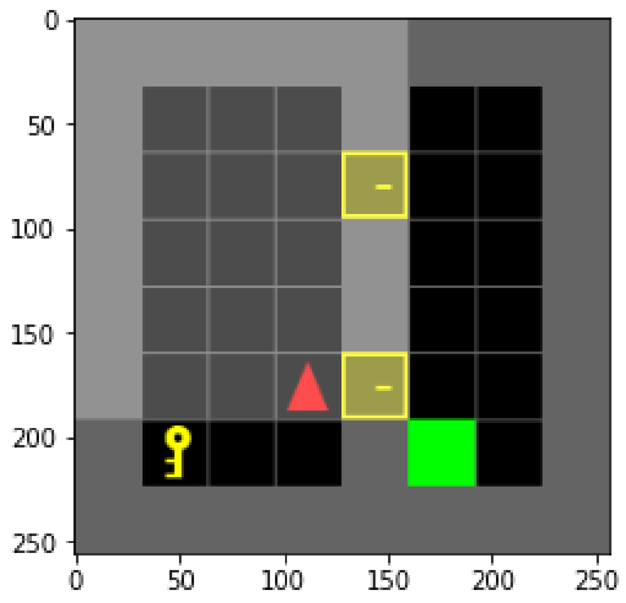

Fig. 15: **DoorKey-8x8_36. Optimal sequence is ['TL', 'MF', 'MF', 'TL', 'PK', 'TL', 'MF', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF']**

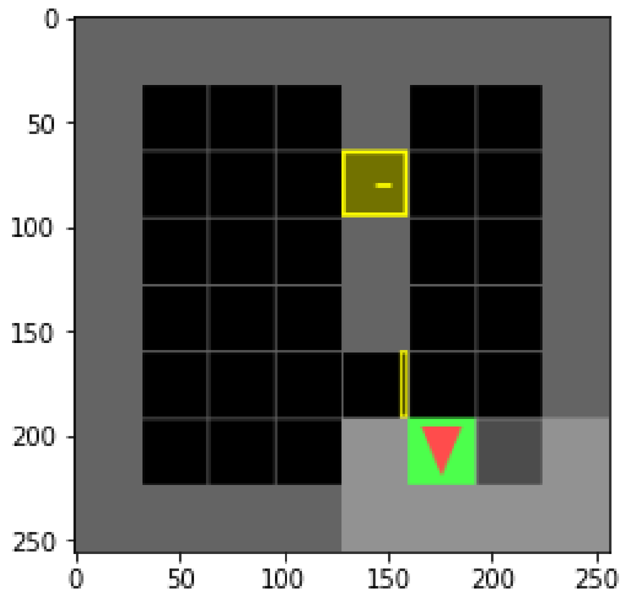

Fig. 16: **DoorKey-8x8_36 reached goal**

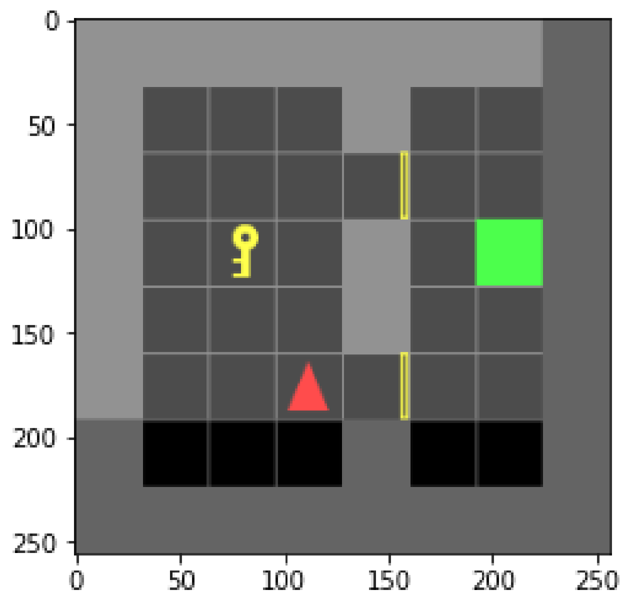

Fig. 17: **DoorKey-8x8_17. Optimal sequence is ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']**

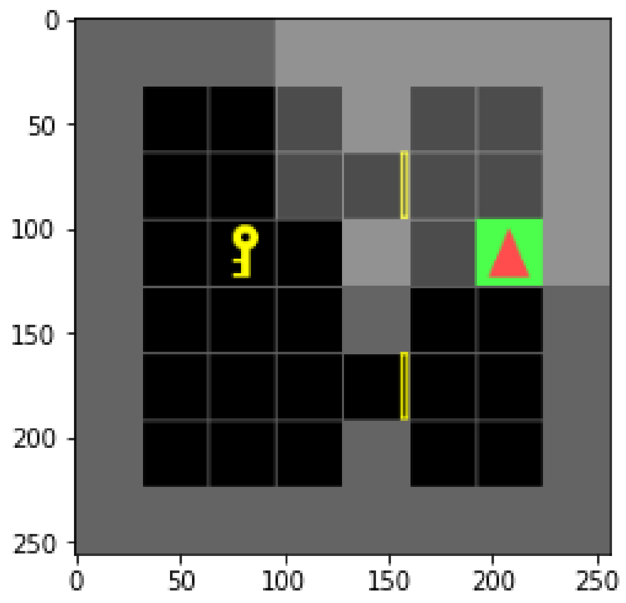

Fig. 18: **DoorKey-8x8_17 reached goal**

incorporate variations.

REFERENCES

[1] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

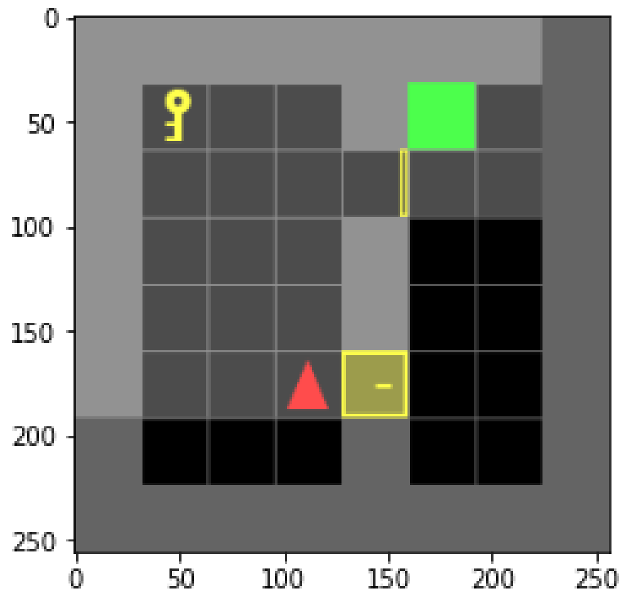

Fig. 19: **DoorKey-8x8_02. Optimal sequence is ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']**

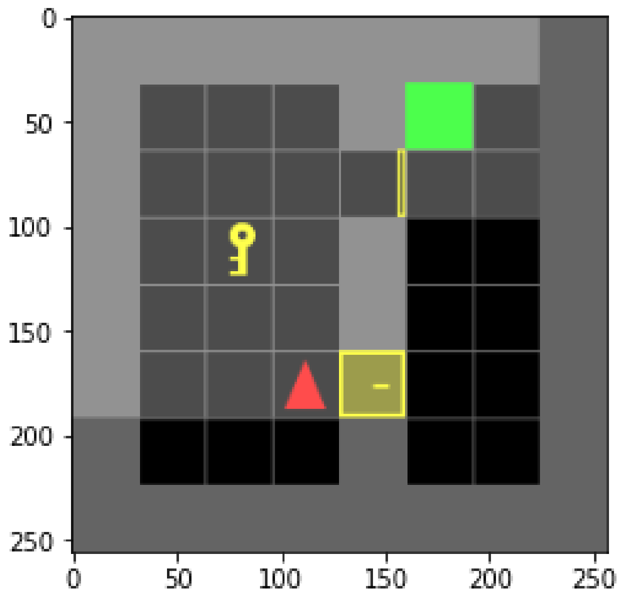

Fig. 21: **DoorKey-8x8_06. Optimal sequence is ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']**

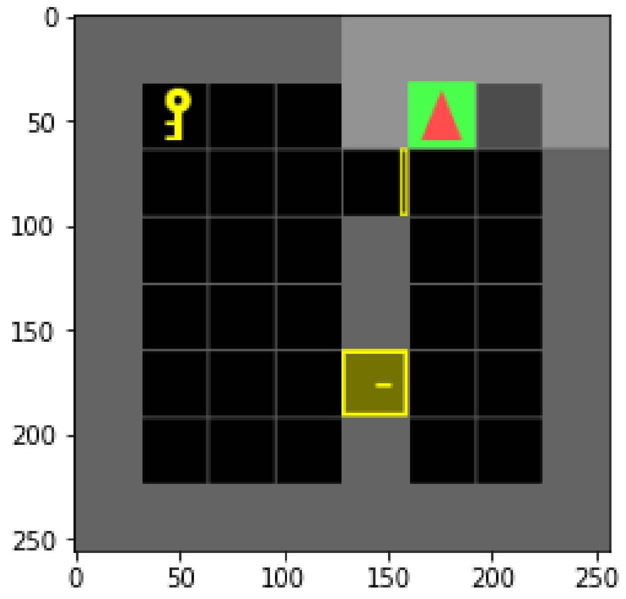

Fig. 20: **DoorKey-8x8_02 reached goal**

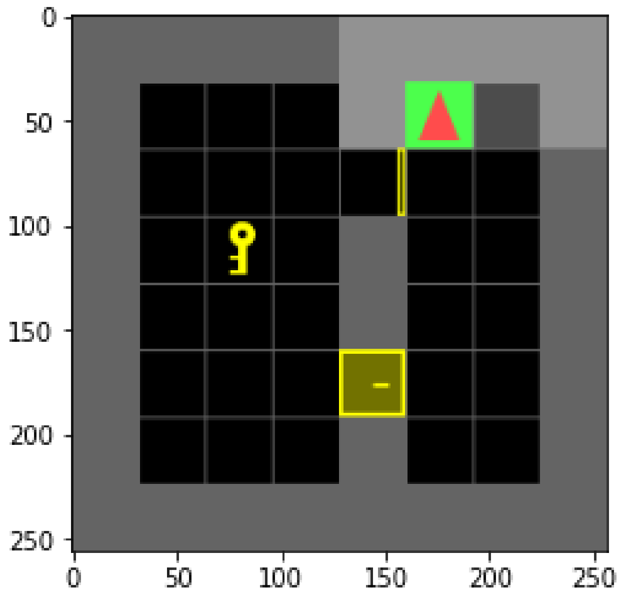

Fig. 22: **DoorKey-8x8_06 reached goal**

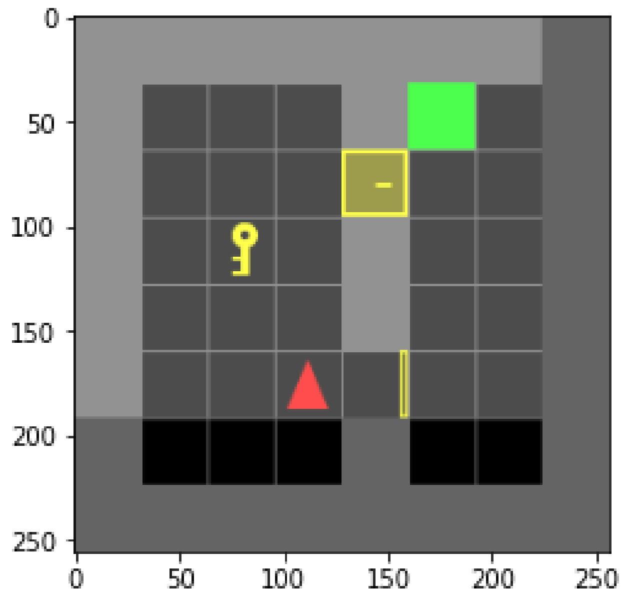

Fig. 23: **DoorKey-8x8_07. Optimal sequence is ['TR', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF']**

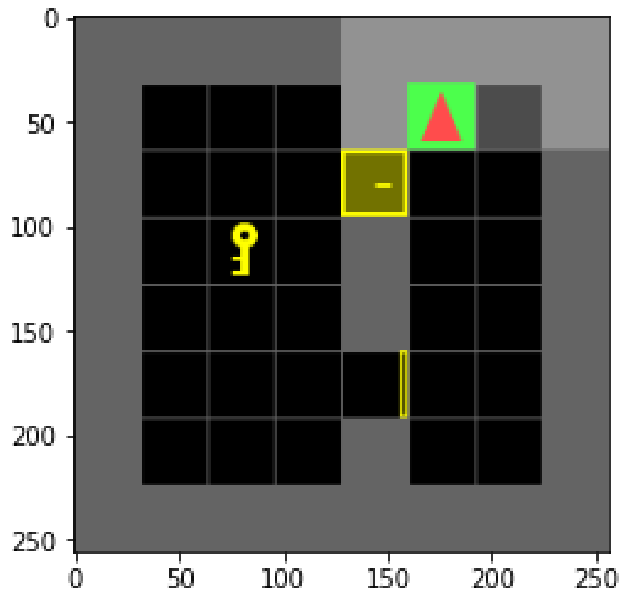

Fig. 24: **DoorKey-8x8_07 reached goal**