Solving the Door-Key Problem via Dynamic Programming

Allen Zeng
UCSD
azeng@ucsd.edu

Abstract

This report models the Door-Key problem in a 2D grid as a Markov Decision Process (MDP). By appropriately formulating the MDP, a dynamic programming algorithm can be used to solve the Door-Key problem. So given any starting state, an optimal sequence of actions can be found that will lead the agent to the goal state, including picking up a key and unlocking doors if necessary. Optimal control sequences computed by this dynamic programming algorithm are shown.

1 Introduction

Reinforcement learning (RL) is a sub-field of machine learning in which intelligent agents learn to take actions in an environment in order to maximize their rewards or minimize their costs. For real world applications, perhaps a robot must move to a certain location while minimizing its battery discharge. In some cases, agents must explore an unknown environment while simultaneously managing their cumulative reward or costs of their actions. In this report, we assume that the agent has complete knowledge of its environment. Given this information, we formulate a Markov Decision Process (MDP) which can be used to determine the optimal sequence of actions our agent must perform to reach its goal state.

Specifically, we are interested in moving an agent on a 2D grid from its initial position to a goal position. We make use of OpenAI's Gym toolkit [1] to simulate an environment consisting of the single agent, the goal position, walls, doors, and a key. The agent may not move into a grid cell containing a wall or a closed door. The agent may pick up a key, and use it to open doors that are closed. Otherwise, the agent may only turn in $90°$ increments and move forwards. The objective is to determine the shortest sequence of actions that moves the agent to the goal position. This problem can be formulated in a MDP, and dynamic programming can be used to solve this problem.

2 Problem Formulation

In this report, we aim to calculate an optimal control sequence that will move an agent to a goal location in a 2D Door-Key grid environment. An agent may face in one of four directions: up, down, left, right. An agent may stand on any empty cell, a cell with a key, a cell with an open door, and the goal position. An agent cannot reside on a cell containing a wall or a closed door. An agent has up to 5 possible actions at any position/state: *move forward* (MF), *turn left* (TL), and *turn right* (TR), *pickup key* (PK), and *unlock door* (UD). Given that an agent can only stand on valid squares (not wall or closed door), an agent can always TL or TR. An agent can only perform MF if the cell in front of it is not a wall and is not a closed door. An agent can only perform PK if a key is in the cell in front of the agent. An agent can only perform UD if the agent is holding a key and the cell in front of the agent contains a closed door.

We solve two very closely related problems in this report, one about a "Known Map" and one about a "Random Map" [2]. In the Known Map case, there are 7 environments of varying size

$\{5 \times 5, 6 \times 6, 8 \times 8\}$, each with one agent starting position, one key, one door, and one goal position. For some of these environments retrieving the key is necessary, but for some others it is not.

In the Random Map case, there are 36 unique 8×8 environments. Denoting the top-left cell as $(0, 0)$, bottom-left as $(0, 7)$, top-right as $(7, 0)$, and bottom-right as $(7, 7)$, the agent always starts at $(3, 5)$ facing upwards. There is always a vertical wall at column 4 and doors at $(4, 2)$ and $(4, 5)$, each of which may be initially opened or closed. The key is randomly located at one of three possible positions $\{(1, 1), (2, 3), (1, 6)\}$. The goal is randomly located in one of three possible positions $\{(5, 1), (6, 3), (5, 6)\}$.

Both of these may be formulated as a MDP with parameters: state $x \in X$, control/action $u \in U(x)$, initial state $x_0 \in X$, motion model F where $x_{t+1} = F(x_t, u_t)$, time horizon T, stage cost $L(x, u)$, terminal cost $q(x)$, and discount factor γ. Note that in our problem, the initial position is known so we specify the initial state as x_0 rather than as a probability mass function p_0 over X [3]. Similarly, actions are always applied successfully so the motion model is a function $F(x, u)$ rather than a conditional probability mass function $p_f(\cdot \mid x_t, u_t)$.

Since the two problems are similar, we first specify the MDP parameters that are in common. Then we specify the MDP parameters which are different between the two cases, namely X, U, F.

For both the Known Map and Random Map problems, the time horizon T is set to `(board_width-2) * (board_height-2) * 5`. This is based on prior knowledge that there is always a 1-cell wide wall surrounding the environment, and that there are at most 5 possible moves at any board position. We define the cost functions as always non-negative (and in fact always positive except at the goal location), so a state will never be visited twice. Meaning, if the problem is equivalently cast as a deterministic shortest path problem [4], there are no negative cycles. The dynamic programming algorithm fully solves the problem in T (or less) iterations.

For both problems, We define the stage cost $L(x, u)$ and terminal cost $q(x, u)$ as follows, using notation [4]:

$$L(x, u) := \begin{cases} 0 & \text{if } x = \tau \\ 1 & \text{otherwise} \end{cases}$$

$$q(x, u) := \begin{cases} 0 & \text{if } x = \tau \\ \infty & \text{otherwise} \end{cases}$$

Where τ is any goal state. More precisely, goal state τ is any state with the same board position as the goal position.

Also for both problems, we specify the discount factor $\gamma = 1$ meaning there is no discount. The costs at all time steps contribute equally to the cumulative cost.

Now, we specify X, U, F for the Known Map case. Here, our task is to compute a separate optimal control policy on each of the 7 provided environments [2]. All of these environments have several things in common, they each one agent starting position, one key, one door, and one goal position. So, we define our state space as

$$X = \{0, ..., m-1\} \times \{0, ..., n-1\} \times \{0, ..., d-1\} \times \{0, ..., h-1\} \times \{0, ..., c-1\}$$

where m is the board width, n is the board height, $(d = 4)$ accounts for the number of directions the agent may face, $(h = 2)$ accounts for whether a key is held by the agent, $(c = 2)$ accounts for whether the door is closed. Although the key position, door position, and goal position is part of the state, we do not explicitly include them as part of the state space because their locations are static in each of the target environments. Instead, the key position and door position are implicitly encoded as part of the control space, and the goal is implicitly encoded in the stage and terminal cost functions above. So for all 7 environments, $x_0 = (a_x, a_y, dir, has\_key, door\_closed)$ where (a_x, a_y) is the initial agent position, dir is the initial agent direction mapped into integers $\{0, 1, 2, 3\}$, $has\_key = 0$, $door\_closed = 1$.

For the Known Map case, the control space $U(X)$ is based on the rules of the game stated in the first paragraph of this section. The agent has at most 5 moves: MF, TL, TR, PK, UD. And for some $x \in X$, this list of possible moves $u = U(x)$ may be shorter. For example, if an agent is facing a wall, it can only TL or TR. $F(x, u)$ assumes that $u = U(x)$ contains only valid moves for a given x. Then for valid x, u, MF adds the vector representation of the agent direct to the agent's current position. TL and TR changes the agent's direction accordingly. PK changes the has_key variable of the state to 1. UD changes the $door\_closed$ variable to 0.

Now, we specify X, U, F for the Random Map case. Here, our task is to compute a single control policy even though the two door statuses, key position, and goal position may vary. We specify the state to account for these variables. We define our state space as

$$X = \{0, ..., m-1\} \times \{0, ..., n-1\} \times \{0, ..., d-1\} \times \{0, ..., h-1\} \times \\ \{0, ..., c_1 - 1\} \times \{0, ..., c_2 - 1\} \times \{0, ..., k-1\} \times \{0, ..., g-1\}$$

where m is the board width, n is the board height, $(d = 4)$ accounts for the number of directions the agent may face, $h = 2$ accounts for whether a key is held by the agent, $(c_1 = 2)$ accounts for whether the top door is closed, $(c_2 = 2)$ accounts for whether the bottom door is closed, $(k = 3)$ accounts for the three possible key locations, $(g = 3)$ accounts for the three possible goal locations. Unlike before, we do explicitly model the possible locations for the key and the goal here. The goal positions are still encoded in the stage and terminal cost functions as above. The door positions are implicitly encoded as part of the control space. In this part, $x_0 = (a_x, a_y, dir, has\_key, top\_door\_closed, bottom\_door\_closed, key\_location, goal\_location)$. Where, top_door_closed indicates whether the door at $(4, 2)$ is initially closed and $bottom\_door\_closed$ indicates whether the door at $(4, 5)$ is initially closed. The $key\_location$ is the key position mapped into integers $\{0, 1, 2\}$. The $goal\_location$ is the goal position mapped into integers $\{0, 1, 2\}$. The other variables are the same as in the Known Map case.

For the Random Map, the control space $U(X)$ is similar to the Known Map case, and is based on the rules of the game stated in the first paragraph of this section. The only change is that there are two possible doors that can be opened, and it is assumed that the key can open either door. Again, $F(x, u)$ assumes that $u = U(x)$ contains only valid moves for a given x. F here is also similar to the Known Map case. The only change is that UD has two possible conditional effects. If UD is used when facing the door at $(4, 2)$, the top_door_closed variable is set to 0. If UD is used when facing the door at $(4, 5)$, the $bottom\_door\_closed$ variable is set to 0.

With these MDP parameters $\{x \in X, u \in U(x), x_0 \in X, F, T, L(x, u), q(x), \gamma\}$ specified, we can use dynamic programming to solve either of the stated problems.

3 Technical Approach

Algorithm 1 Dynamic Programming

```
1: **Input**: MDP $(\mathcal{X}, \mathcal{U}, p_0, p_f, T, \ell, \mathsf{q}, \gamma)$
2:
3: $V_T(\mathbf{x}) = \mathsf{q}(\mathbf{x}), \quad \forall \mathbf{x} \in \mathcal{X}$
4: **for** $t = (T-1) \dots 0$ **do**
5:     $Q_t(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(\cdot|\mathbf{x}, \mathbf{u})} [V_{t+1}(\mathbf{x}')], \quad \forall \mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}(\mathbf{x})$
6:     $V_t(\mathbf{x}) = \min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} Q_t(\mathbf{x}, \mathbf{u}), \quad \forall \mathbf{x} \in \mathcal{X}$
7:     $\pi_t(\mathbf{x}) = \arg\min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} Q_t(\mathbf{x}, \mathbf{u}), \quad \forall \mathbf{x} \in \mathcal{X}$
8: **return** policy $\pi_{0:T-1}$ and value function $V_0$
```

Dynamic programming algorithm, reproduced from [5].

Using the specified MDP, we apply a dynamic programming algorithm (DPA) to solve for the optimal control policy $\pi_t(x)$. The algorithm from [5] was used as an implementation reference, and is reproduced in Algorithm 1. The only difference in our implementation is in line 5. Since we operate in discrete space and actions are always performed successfully, we instead do (using our own notation):

$$Q_t(x, u) = L(x, u) + \gamma V_{t+1}(F(x, u)), \quad \forall x \in X, u \in U(x)$$

Note that the state space X, for both of the two problems specified in Section 2, enumerates all possible states as a list of integers. We can therefore use a state x as an index into tensors which store

$V_{0:T}(x)$ and $\pi_{0:T-1}$. For the Known Map case, we keep track of the value function V and policy π in tensors each with shape $(m \times n \times d \times h \times c \times T)$. For the Random Map case, we keep track of the value function V and policy π in tensors each with shape $(m \times n \times d \times h \times c_1 \times c_2 \times k \times g \times T)$. The value of an element in V is the value at that corresponding state x and time t. The value of an element in π is the optimal action at that corresponding state x and time t.

After running the dynamic programming algorithm and returning policy π, we use the initial state x_0 and motion model F to record the optimal sequence of actions. We repeated apply

$$u_t^* \leftarrow \pi_t(x_t)$$
$$x_{t+1} \leftarrow F(x_t, u_t^*)$$

and record $S^* = \{u_0^*, u_1^*, ..., u_G^*\}$ until we reach a state which has the agent position equal to the goal position. Where G is the minimum number of steps required to reach the goal position, given the environment and obstacles. The sequence S^* is the optimal sequence of actions to move the agent to the goal position. This dynamic programming algorithm is optimal for the finite-horizon optimal control problem [5].

4 Results

Using the specified MDP formulation and the dynamic programming algorithm (DPA), optimal control sequences were produced for both the Known Maps and Random Maps cases. Visualizations of the initial states and corresponding optimal action sequences of the Known Maps are shown in Section 4.1, Page 5. Visualizations of the initial states and corresponding optimal action sequences of the Random Maps are shown in Section 4.2, Page 7. [1]

As stated previously, for each environment in Known Maps a separate optimal control policy is calculated. Following the policy from the initial state, an optimal sequence of actions is recorded which leads the agent to the goal position. Upon inspections of Figures 1–7, all of the calculated sequences appear to be optimal. The agent always progress towards the goal with no wasted moves, and only picks up the key to unlock the door if necessary. For environments with size 5×5 or 6×6, the DPA took less than 0.1 seconds to complete. For environments with size 8×8, the DPA took about 0.34 seconds to complete. In these small environments, and small state and control spaces, the DPA appears to run very fast and produces correct results.

In the Random Maps case, a single control policy is calculated which accounts for different potential door statuses, key positions, and goal positions. Upon inspecting Figures 8–43, the calculated control sequences all appear to be optimal. The agent always progress towards the goal with no wasted moves, and only picks up a key to unlock a door if necessary. With this larger state space, the DPA completes in about 7.411 seconds. As expected, the DPA takes longer to complete but is still very fast. After calculating the control policy once, it can be applied to all 36 possible random environments.

Although DPA runs well in this 2D setting, it is clear that the algorithm will slow down substantially as the board size, possible key locations, possible door locations, and possible goal locations increase. And, the memory footprint will increase correspondingly. Note that different starting positions have no effect on the runtime of DPA, since it computes an optimal $\pi(x)$ for all x. You can start anywhere and follow $\pi(x)$ until you reach the goal position.

As mentioned in Section 3, we store the value function V and policy π as multi-dimensional tensors. While the implementation is straightforward to understand, there is some wasted memory. We include the border wall and the inner wall locations in these tensors for ease of indexing. Since an agent cannot be in the same position as a wall, the indices that correspond to wall locations point to wasted memory. An alternative data structure which excludes these wall locations may have better memory efficiency. For this small 2D problem however, the inefficient tensor representation was not an issue.

[1] GIF animations of these control sequences are also uploaded as part of the assignment code submission.

4.1 Part A “Known Map” Figures

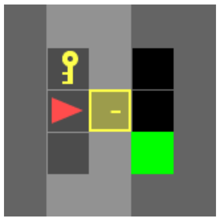

Figure 1: Environment `Doorkey-5x5-normal`, solved by: [’TL’, ’PK’, ’TR’, ’UD’, ’MF’, ’MF’, ’TR’, ’MF’]

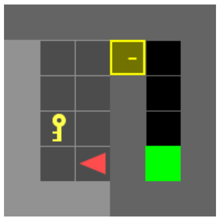

Figure 2: Environment `Doorkey-6x6-normal`, solved by: [’MF’, ’TR’, ’PK’, ’MF’, ’MF’, ’MF’, ’TR’, ’MF’, ’UD’, ’MF’, ’MF’, ’TR’, ’MF’, ’MF’, ’MF’]

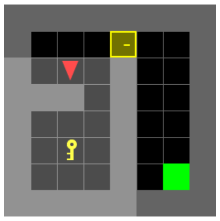

Figure 3: Environment `Doorkey-8x8-normal`, solved by: [’TL’, ’MF’, ’TR’, ’MF’, ’MF’, ’MF’, ’TR’, ’PK’, ’TR’, ’MF’, ’MF’, ’MF’, ’MF’, ’TR’, ’UD’, ’MF’, ’MF’, ’MF’, ’TR’, ’MF’, ’MF’, ’MF’, ’MF’, ’MF’]

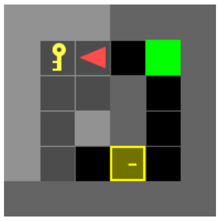

Figure 4: Environment `Doorkey-6x6-direct`, solved by: [’TL’, ’TL’, ’MF’, ’MF’]

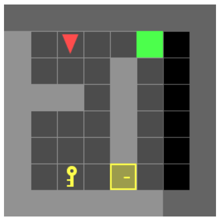

Figure 5: Environment `Doorkey-8x8-direct`, solved by: ['TL', 'MF', 'MF', 'MF']

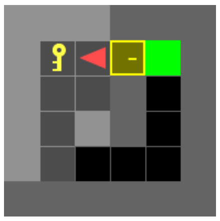

Figure 6: Environment `Doorkey-6x6-shortcut`, solved by: ['PK', 'TL', 'TL', 'UD', 'MF', 'MF']

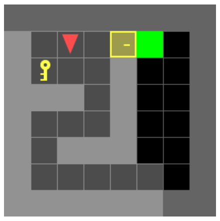

Figure 7: Environment `Doorkey-8x8-shortcut`, solved by: ['MF', 'TR', 'PK', 'TR', 'MF', 'TR', 'MF', 'UD', 'MF', 'MF']

4.2 Part B "Random Map" Figures

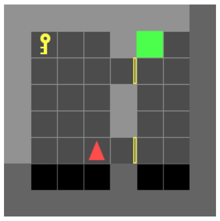

Figure 8: Environment `DoorKey-8x8_01`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']

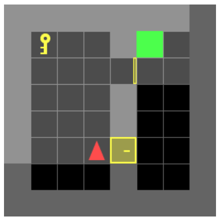

Figure 9: Environment `DoorKey-8x8_02`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']

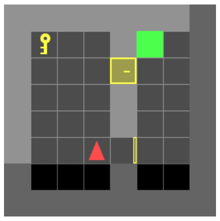

Figure 10: Environment `DoorKey-8x8_03`, solved by: ['TR', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF']

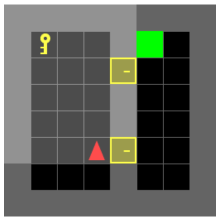

Figure 11: Environment `DoorKey-8x8_04`, solved by: ['MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'PK', 'TL', 'MF', 'TL', 'MF', 'UD', 'MF', 'MF', 'TL', 'MF']

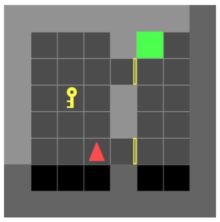

Figure 12: Environment `DoorKey-8x8_05`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']

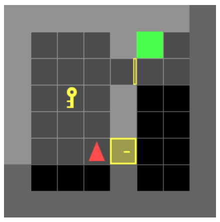

Figure 13: Environment `DoorKey-8x8_06`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']

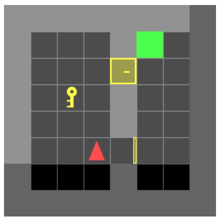

Figure 14: Environment `DoorKey-8x8_07`, solved by: ['TR', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF']

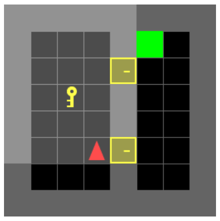

Figure 15: Environment `DoorKey-8x8_08`, solved by: ['MF', 'MF', 'TL', 'PK', 'TR', 'MF', 'TR', 'UD', 'MF', 'MF', 'TL', 'MF']

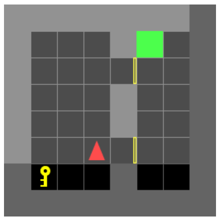

Figure 16: Environment `DoorKey-8x8_09`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']

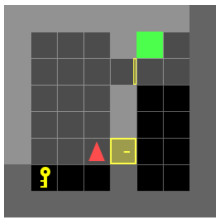

Figure 17: Environment `DoorKey-8x8_10`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF']

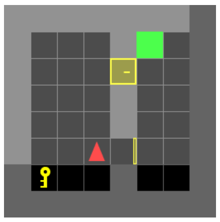

Figure 18: Environment `DoorKey-8x8_11`, solved by: ['TR', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF']

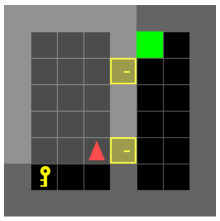

Figure 19: Environment `DoorKey-8x8_12`, solved by: ['TL', 'MF', 'MF', 'TL', 'PK', 'TL', 'MF', 'MF', 'UD', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF']

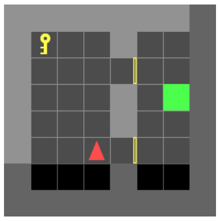

Figure 20: Environment `DoorKey-8x8_13`, solved by: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

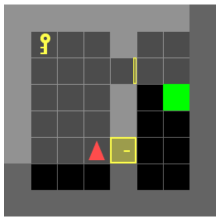

Figure 21: Environment `DoorKey-8x8_14`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'TR', 'MF']

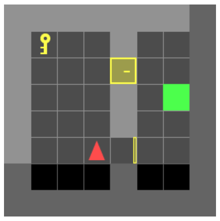

Figure 22: Environment `DoorKey-8x8_15`, solved by: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

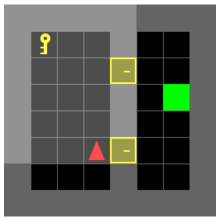

Figure 23: Environment `DoorKey-8x8_16`, solved by: ['MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'PK', 'TL', 'MF', 'TL', 'MF', 'UD', 'MF', 'MF', 'MF', 'TR', 'MF']

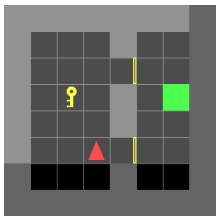

Figure 24: Environment `DoorKey-8x8_17`, solved by: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

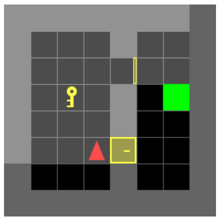

Figure 25: Environment `DoorKey-8x8_18`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'TR', 'MF']

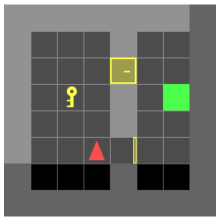

Figure 26: Environment `DoorKey-8x8_19`, solved by: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

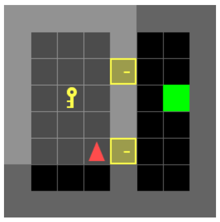

Figure 27: Environment `DoorKey-8x8_20`, solved by: ['MF', 'MF', 'TL', 'PK', 'TR', 'MF', 'TR', 'UD', 'MF', 'MF', 'MF', 'TR', 'MF']

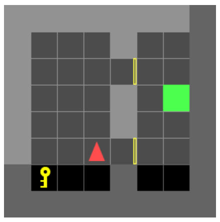

Figure 28: Environment `DoorKey-8x8_21`, solved by: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

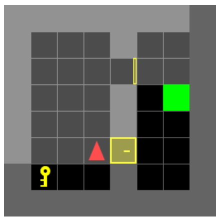

Figure 29: Environment `DoorKey-8x8_22`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'TR', 'MF']

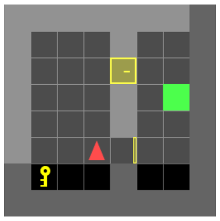

Figure 30: Environment `DoorKey-8x8_23`, solved by: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

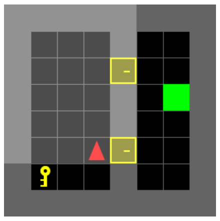

Figure 31: Environment `DoorKey-8x8_24`, solved by: ['TL', 'MF', 'MF', 'TL', 'PK', 'TL', 'MF', 'MF', 'UD', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

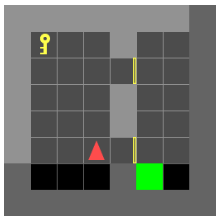

Figure 32: Environment `DoorKey-8x8_25`, solved by: ['TR', 'MF', 'MF', 'TR', 'MF']

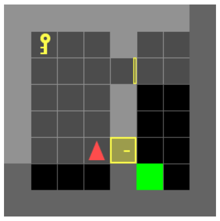

Figure 33: Environment `DoorKey-8x8_26`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'MF']

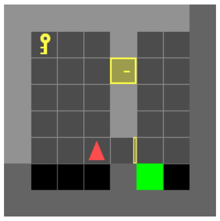

Figure 34: Environment `DoorKey-8x8_27`, solved by: ['TR', 'MF', 'MF', 'TR', 'MF']

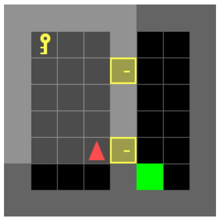

Figure 35: Environment `DoorKey-8x8_28`, solved by: ['MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'PK', 'TL', 'MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF']

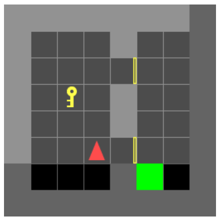

Figure 36: Environment `DoorKey-8x8_29`, solved by: ['TR', 'MF', 'MF', 'TR', 'MF']

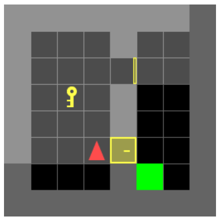

Figure 37: Environment `DoorKey-8x8_30`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'MF']

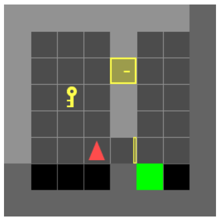

Figure 38: Environment `DoorKey-8x8_31`, solved by: ['TR', 'MF', 'MF', 'TR', 'MF']

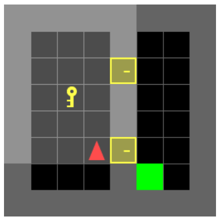

Figure 39: Environment `DoorKey-8x8_32`, solved by: ['MF', 'MF', 'TL', 'PK', 'TL', 'MF', 'MF', 'TL', 'UD', 'MF', 'MF', 'TR', 'MF']

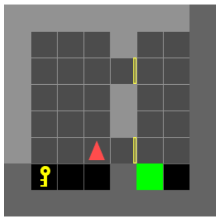

Figure 40: Environment `DoorKey-8x8_33`, solved by: ['TR', 'MF', 'MF', 'TR', 'MF']

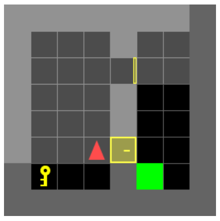

Figure 41: Environment `DoorKey-8x8_34`, solved by: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'MF']

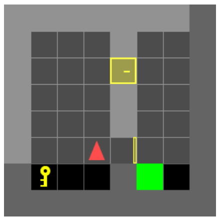

Figure 42: Environment `DoorKey-8x8_35`, solved by: ['TR', 'MF', 'MF', 'TR', 'MF']

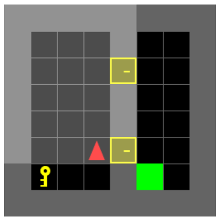

Figure 43: Environment `DoorKey-8x8_36`, solved by: ['TL', 'MF', 'MF', 'TL', 'PK', 'TL', 'MF', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF']

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Nikolay Atanasov. Project 1: Dynamic programming. In *ECE276B: Planning & Learning in Robotics*, 2021.

[3] Nikolay Atanasov. Lecture 3: Markov decision processes. In *ECE276B: Planning & Learning in Robotics*, 2021.

[4] Nikolay Atanasov. Lecture 5: Deterministic shortest path. In *ECE276B: Planning & Learning in Robotics*, 2021.

[5] Nikolay Atanasov. Lecture 4: Dynamic programming. In *ECE276B: Planning & Learning in Robotics*, 2021.