ECE 276B Project 1: Dynamic Programming

Jean Gorby Calicdan
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
jcalicda@ucsd.edu

***Abstract*—This project investigates optimal planning in a discrete gridworld setting, where an agent must navigate from an initial position to a goal location in the presence of obstacles, doors, and keys. The environment introduces both deterministic structure (known maps) and stochastic variation (random maps with varying door and key configurations). The task is formalized as a finite-horizon Markov Decision Process (MDP), with the objective of minimizing total action cost under motion constraints. We design and implement a Dynamic Programming (DP) approach that computes state-dependent policies using backward induction over the time horizon. For known maps, we compute map-specific optimal policies. For the random map scenario, we precompute a general policy over the joint state space—including goal and key indices, door states, and possession of the key—that adapts to all valid map instantiations. Key contributions include robust handling of invalid actions, dynamic value function pruning for computational efficiency, and policy generalization to unseen environments. Results demonstrate that the proposed DP framework reliably computes minimal-cost action sequences across diverse scenarios, including cases where multiple doors must be unlocked to reach the goal.**

***Index Terms*—Dynamic Programming, Markov Decision Process (MDP), Optimal Control, Policy Generalization, Door-Key Environment, Finite Horizon Planning**

I. Introduction

Autonomous navigation is a fundamental capability in robotics, requiring agents to reason over spatial constraints and action costs to reach a specified goal. This project investigates the planning problem in a constrained gridworld known as the Door Key environment, where an agent must navigate to a goal while potentially unlocking doors using a key. Such settings abstract critical elements of real-world robotic tasks, including manipulation, obstacle avoidance, and conditional reasoning over action preconditions.

The task is modeled as a finite-horizon Markov Decision Process (MDP), where the agent incurs a positive cost for each action and must select a control policy that minimizes the cumulative cost of reaching the goal. To solve this, we implement a dynamic programming approach using backward induction over the full state-time space. For known maps, we construct environment-specific policies, while for randomized maps, we precompute a single generalizable policy over an expanded state space that encodes door states, goal/key indices, and agent possession of the key. This enables robust policy transfer across unseen environments with variable structural configurations.

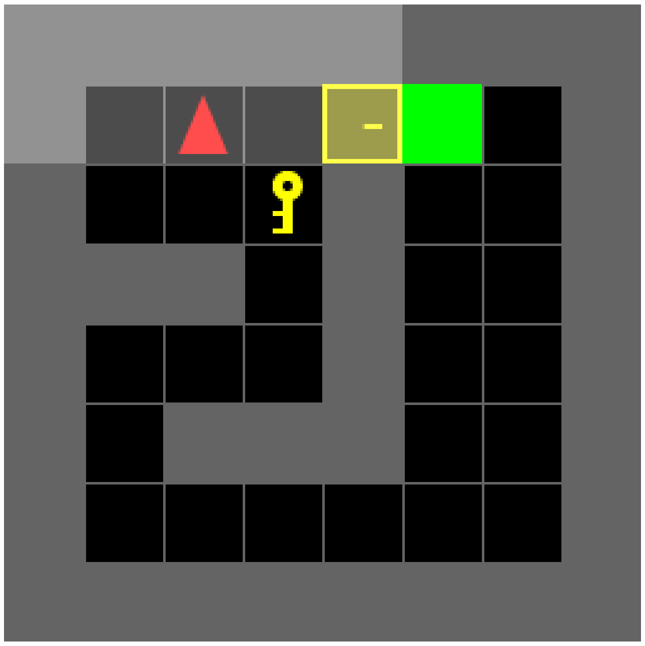
Fig. 1: Example grid environment

II. Problem Formulation

In this project, we study the problem of computing an optimal control policy for a robot navigating through a grid-based environment. The robot's task is to reach a designated goal square while minimizing the total cost of its actions. The environment includes static obstacles (walls), a locked door that must be opened, and a key that enables this operation. The robot must reason not only about how to get to the goal but also whether it needs to retrieve the key, unlock the door, and in what order to take specific actions in order to complete the task efficiently.

The robot operates in a discrete $h \times w$ grid world where each cell may contain an object such as a wall, key, door, goal, or be empty. The robot is initially placed at a known position and orientation. It can perform five discrete actions: move forward (MF), turn left (TL), turn right (TR), pick up a key (PK), and unlock a door (UD). These actions incur a cost. The robot is unable to pass through walls or locked doors, and it cannot occupy the same square as the key or the door unless certain preconditions are met. For example, the robot must be adjacent and facing the key to pick it up, or adjacent and facing a closed door to unlock it.

The robot's goal is to find a minimum-cost sequence of actions that leads it to the goal square. The environment is deterministic and fully observable, and the control policy should reflect knowledge of how the robot interacts with specific elements in the environment. A valid policy must satisfy the structural constraints of the grid world while exploiting valid control sequences that allow the agent to unlock doors and reach the goal efficiently. Two types of environments are considered in this project: known and random.

In mathematical terms, we want to solve the following optimization problem:

$$\min_{\pi_{t:T-1}} V_t^{\pi}(\mathbf{x}) := \mathbb{E}_{\mathbf{x}_{t+1:T}}\left[q(\mathbf{x}_T) + \sum_{\tau=t}^{T-1} \ell(\mathbf{x}_\tau, \pi_\tau(\mathbf{x}_\tau)) \,\middle|\, \mathbf{x}_t = \mathbf{x} \right]$$

$$\text{s.t.} \quad \mathbf{x}_{\tau+1} \sim p_f(\cdot \mid \mathbf{x}_\tau, \pi_\tau(\mathbf{x}_\tau)), \quad \tau = t, \dots, T-1$$

$$\mathbf{x}_\tau \in \mathcal{X}, \quad \pi_\tau(\mathbf{x}_\tau) \in \mathcal{U}$$

In **Part A**, the environment is predetermined and known to the robot. There are seven such environments, each with different configurations of walls, the door, key, and goal. The agent must generate an optimal policy tailored to each specific environment. The robot has full knowledge of the positions of the goal and the key, and it must determine when and how to unlock the door if necessary.

In **Part B**, the environment is generated randomly. The grid is fixed at 10 × 10, with a vertical wall at column 5 and two possible doors at coordinates $(5, 3)$ and $(5, 7)$. These doors may be either open or closed. The robot always starts at position $(4, 8)$ facing up, but the goal, key, and door states are randomized for each instance. Specifically, the goal may appear at one of three locations:

$$G := \{[6, 1], [7, 3], [6, 6]\},$$

the key may appear at one of:

$$K := \{[2, 2], [2, 3], [1, 6]\},$$

and the door states for each door are binary:

$$D_1, D_2 \in \{0, 1\},$$

where 0 indicates a closed door and 1 indicates an open door.

The robot must construct a single generalized policy that works across all 36 random configurations of the environment, regardless of which goal or key position is used or whether each door is open or closed. This introduces a challenging trade-off between specificity and generality: the robot must encode its policy over an expanded state space that includes door states and object indices. The result should be a policy that is robust to all valid randomized instantiations of the grid environment.

To formalize both Part A and Part B, we define each setting as a Markov Decision Process (MDP) with tuple:

$$(\mathcal{X}, \mathcal{U}, f, x_0, T, \ell, q),$$

where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the control space, f is the (deterministic) motion model, x_0 is the initial state, T is the time horizon, $\ell(x, u)$ is the stage cost, and $q(x)$ is the terminal cost. The details of these components are defined separately for the known and random environment settings in the subsections that follow.

A. Known Environment MDP

In the known maps, the agent operates in a fixed gridworld layout with a single locked door and a single key.

- **State Space** $\mathcal{X}$:

$$\mathcal{X} \subset \mathbb{Z}^5, |\mathcal{X}| = h \times w \times 4 \times 2 \times 2$$

where $\mathbf{x}_t \in \mathcal{X}$ can be described as:

$$\mathbf{x}_t := \begin{bmatrix} x_t^1 \\ x_t^2 \\ x_t^3 \\ x_t^4 \\ x_t^5 \end{bmatrix} = \begin{bmatrix} \textit{robot\_x} \\ \textit{robot\_y} \\ \textit{robot\_direction} \\ \textit{door\_state} \\ \textit{key\_state} \end{bmatrix}$$

$$\begin{aligned} x_t^1 &\in \{0, 1, \dots, h-1\}, && \text{(robot x-coordinate)} \\ x_t^2 &\in \{0, 1, \dots, w-1\}, && \text{(robot y-coordinate)} \\ x_t^3 &\in \{0, 1, 2, 3\}, && \text{(robot orientation)} \\ x_t^4 &\in \{0, 1\}, && \text{(door state)} \\ x_t^5 &\in \{0, 1\}, && \text{(whether robot has key} \end{aligned}$$

and h, w correspond with the height and width of the current environment, *robot_x* and *robot_y* correspond with the robots horizontal and vertical coordinates, respectively, *robot_direction* corresponds with the direction that the robot is facing (mapping in Table 1), *door_state* corresponds to whether the door is open or not, and *key_state* corresponds to whether the robot has the key or not.

Mapping of Direction Vectors to Orientation Indices		
Direction Vector	**Meaning**	**Mapped Index**
$[-1,\ 0]$	Left	0
$[0,\ -1]$	Up	1
$[1,\ 0]$	Right	2
$[0,\ 1]$	Down	3

Table 1: Direction index encoding used in the motion model.

- **Control Space** $\mathcal{U}$:

$$\mathcal{U} = \{\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}\}$$

- **Motion Model** f:

$$f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$$

$$f(x, u) = \begin{cases} x_{t+1} & \text{if valid\_action}(x, u) = 1 \\ x_t & \text{if valid\_action}(x, u) = 0, \\ & \text{or the robot is already at the goal} \end{cases}$$

- **Initial State** x_0:

$$x_0 = (x_r^0, y_r^0, d^0, 0, 0)$$

- **Planning Horizon** T: $T = 100$
- **Stage Cost** $\ell(x,u)$:

$$\ell(u) = \begin{cases} 3 & \text{if } u \in \{\text{MF}, \text{TL}, \text{TR}\} \\ 1 & \text{if } u \in \{\text{PK}, \text{UD}\} \\ \infty & \text{if the action is invalid} \end{cases}$$

- **Terminal Cost** $q(x)$:

$$q(x) = \begin{cases} 0 & \text{if } (x_r, y_r) = (x_g, y_g) \\ \infty & \text{otherwise} \end{cases}$$

The valid_action(x,u) routine determines whether an action is feasible given the agent's current position, orientation, and the environment configuration, and will be described in the Technical Approach section.

B. Random Environment MDP

- **State Space** $\mathcal{X}$:

$$\mathcal{X} \subset \mathbb{Z}^8, \quad |\mathcal{X}| = h \times w \times 4 \times 3 \times 3 \times 2 \times 2 \times 2$$

where $\mathbf{x}_t \in \mathcal{X}$ is defined as:

$$\mathbf{x}_t := \begin{bmatrix} x_t^1 \\ x_t^2 \\ x_t^3 \\ x_t^4 \\ x_t^5 \\ x_t^6 \\ x_t^7 \\ x_t^8 \end{bmatrix} = \begin{bmatrix} \textit{robot\_x} \\ \textit{robot\_y} \\ \textit{robot\_direction} \\ \textit{goal\_index} \\ \textit{key\_index} \\ \textit{door1\_state} \\ \textit{door2\_state} \\ \textit{key\_state} \end{bmatrix}$$

$$x_t^1 \in \{0, 1, \dots, h-1\}, \qquad \text{(robot x-coordinate)}$$
$$x_t^2 \in \{0, 1, \dots, w-1\}, \qquad \text{(robot y-coordinate)}$$
$$x_t^3 \in \{0, 1, 2, 3\}, \qquad \text{(robot orientation)}$$
$$x_t^4 \in \{0, 1, 2\}, \qquad \text{(goal index)}$$
$$x_t^5 \in \{0, 1, 2\}, \qquad \text{(key index)}$$
$$x_t^6 \in \{0, 1\}, \qquad \text{(door 1 state)}$$
$$x_t^7 \in \{0, 1\}, \qquad \text{(door 2 state)}$$
$$x_t^8 \in \{0, 1\}, \qquad \text{(whether robot has key)}$$

The values for *goal_index* and *key_index* correspond to the following fixed coordinate sets:

$$G := \{[6,1], [7,3], [6,6]\}, \quad K := \{[2,2], [2,3], [1,6]\}$$

where $G[x_t^4]$ gives the active goal coordinates, and $K[x_t^5]$ gives the key location. The robot always spawns at location $[4, 8]$ facing upward.

- **Control Space** $\mathcal{U}$:

$$\mathcal{U} = \{\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}\}$$

- **Motion Model** f:

$$f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$$

$$f(x,u) = \begin{cases} x_{t+1} & \text{if valid\_action}(x,u) = 1 \\ x_t & \text{if valid\_action}(x,u) = 0, \\ & \text{or the robot is already at the goal} \end{cases}$$

- **Initial State** x_0:

$$x_0 = (4, 8, 1, g_i, k_i, d_1, d_2, 0)$$

where g_i and k_i denote the goal and key index respectively, and $d_1, d_2 \in \{0,1\}$ are the initial open/closed states of door 1 and door 2.

- **Planning Horizon** T: $T = 200$
- **Stage Cost** $\ell(x,u)$:

$$\ell(u) = \begin{cases} 3 & \text{if } u \in \{\text{MF}, \text{TL}, \text{TR}\} \\ 1 & \text{if } u \in \{\text{PK}, \text{UD}\} \\ \infty & \text{if the action is invalid} \end{cases}$$

- **Terminal Cost** $q(x)$:

$$q(x) = \begin{cases} 0 & \text{if } (x_r, y_r) = G[x_t^4], \\ & \quad \text{and for any valid } x_t^5, x_t^6, x_t^7, x_t^8 \\ \infty & \text{otherwise} \end{cases}$$

The valid_action(x,u) routine determines whether an action is feasible given the agent's current position, orientation, and the environment configuration, and will be described in the Technical Approach section. Valid state values for the terminal costs is discussed more in the Discussion section.

III. Technical Approach

By formulating the navigation problem as a finite-horizon MDP, we can apply a deterministic dynamic programming algorithm (DPA) to compute the optimal control policy π^* that minimizes total cost over a horizon T. In this section, we describe our technical pipeline for both known and random map scenarios. Each case uses a similar high-level framework but differs in how the motion model and state space are defined and how dynamic programming is executed.

A. Known Environment

1) Motion Model Implementation: For the known environment setting, we define a deterministic transition function $f(x,u)$ using the helper function `motion_model_known()`, which computes the next state x' given a current state x and action u. This function calls `check_valid_action_known()` to determine whether an action is admissible. The criteria for validity include staying within map boundaries, not colliding with walls or closed doors, and ensuring correct orientation when interacting with the key or door. If the action is valid, the robot transitions to a new state; otherwise, it remains in place.

The logic for determining whether an action is valid is given in Table II.

Table II: Conditions for Valid Actions in Known Environment

Action	Validity Conditions
Move Forward (MF)	Predict new position based on orientation: • 0 (Left): $x' = x - 1$ • 1 (Up): $y' = y - 1$ • 2 (Right): $x' = x + 1$ • 3 (Down): $y' = y + 1$ Invalid if: • New position is outside map bounds. • New position is a closed door ($x'[3] = 0$ and matches `door_pos`). • New position is in `inner_walls`.
Pick Up Key (PK)	Invalid if: • Key already picked up ($x[4] = 1$) • Robot is not adjacent to the key • Robot is not facing the key: – Below key, not facing up ($x[2] \neq 1$) – Above key, not facing down ($x[2] \neq 3$) – Right of key, not facing left ($x[2] \neq 0$) – Left of key, not facing right ($x[2] \neq 2$) Valid only if adjacent and facing the key.
Unlock Door (UD)	Invalid if: • Robot does not have the key ($x[4] = 0$) • Door is already open ($x[3] = 1$) • Robot is not adjacent to the door • Robot is not facing the door correctly (must be left of door and facing right) Valid only if robot is to the left of the door, facing right, and has the key.
Turn Left (TL), Turn Right (TR)	These actions are always valid. They update the robot's orientation without changing its position.

2) Dynamic Programming: We solve the optimal control problem using a finite-horizon backward induction approach. At the final time step T, we initialize the value function using the terminal cost $V_T(x) = q(x)$. We then iterate backwards in time, computing the cost-to-go $Q_t(x, u)$ and optimal value $V_t(x)$ at each state for every possible action. The optimal action is stored in the policy $\pi_t(x)$.

Algorithm 1 Deterministic Dynamic Programming for Known Map

```
0: **Input:** MDP $(\mathcal{X}, \mathcal{U}, f, T, \ell, q)$
0: $V_T(x) = q(x), \quad \forall x \in \mathcal{X}$
0: **for** $t = T-1$ to 0 **do**
0:    **for all** $x \in \mathcal{X}$ **do**
0:       **for all** $u \in \mathcal{U}$ **do**
0:          $x' = f(x, u)$
0:          $Q_t(x, u) = \ell(x, u) + V_{t+1}(x')$
0:       **end for**
0:       $V_t(x) = \min_{u \in \mathcal{U}} Q_t(x, u)$
0:       $\pi_t(x) = \arg\min_{u \in \mathcal{U}} Q_t(x, u)$
0:    **end for**
0:    **if** $V_t = V_{t+1}$ for all $x$ **then**
0:       **break**
0:    **end if**
0: **end for**
0: **return** policy $\pi_{0:T-1}$ and value function $V_0$ =0
```

This procedure is deterministic and guarantees convergence to the optimal policy as long as the value function stops changing between iterations. In practice, all known maps converged in under 30 steps even though we conservatively set $T = 100$. This is described in Algorithm 1.

B. Random Environment

1) Motion Model Implementation: In the random environment setting, the agent always starts at $(4, 8)$ facing up, but the goal and key positions, as well as the door states, are randomized for each trial. To handle this complexity, we expand the state space to 8 dimensions: agent position, direction, goal index, key index, door 1 state, door 2 state, and key possession. The motion model is implemented in `motion_model_random()`, which handles transitions in this higher-dimensional state space.

As in the known case, `check_valid_action_random()` verifies action feasibility. This function incorporates logic to check door unlock conditions, key pickup constraints, and map boundaries. It distinguishes between doors at $(5, 3)$ and $(5, 7)$ and checks whether each is open or closed. These checks ensure that the motion model properly accounts for the randomized environment structure.

The logic for determining whether an action is valid is given in Table III.

2) Dynamic Programming: To avoid recomputing policies for each of the 36 random configurations, we construct a single general policy over the entire state space using `precompute_general_policy()`. This function initializes a value function $V(x, t)$ over the 8D state space described earlier and performs backward induction from $t = T - 1$ to 0 using the deterministic motion model and stage cost.

Table III: Conditions for Valid Actions in Random Environment

Action	Validity Conditions
Move Forward (MF)	Predict new position based on orientation: • 0 (Left): $x' = x - 1$ • 1 (Up): $y' = y - 1$ • 2 (Right): $x' = x + 1$ • 3 (Down): $y' = y + 1$ Action is **invalid** if: • New position is outside the bounds of the 10×10 grid. • New position is a closed door: – Door 1: $x'[5] = 0$ and new position is at $(5, 3)$ – Door 2: $x'p[6] = 0$ and new position is at $(5, 7)$ • New position is in `inner_walls`. Otherwise, the action is valid.
Pick Up Key (PK)	Action is **invalid** if: • Robot already has the key ($x[7] = 1$) • Robot is not adjacent to the key • Robot is not facing the key: – Below key but not facing up ($x[2] \neq 1$) – Above key but not facing down ($x[2] \neq 3$) – Right of key but not facing left ($x[2] \neq 0$) – Left of key but not facing right ($x[2] \neq 2$) Action is **valid** only if robot is adjacent to and facing the key.
Unlock Door (UD)	Action is **invalid** if: • Robot is not holding the key ($x[7] = 0$) • Robot is not adjacent to either door at the correct unlock position (left of the door) • Robot is not facing the door correctly (must be facing right) • Door is already open: – Door 1: $x[5] = 1$ – Door 2: $x[6] = 1$ Action is **valid** only if: • Robot is left of the door (at $x = 4$) and • Facing right ($x[2] = 2$) and • Holding the key and door is closed
Turn Left (TL), Turn Right (TR)	These actions are always valid. They update the robot's orientation in-place without affecting position or interacting with any objects.

The terminal value function is initialized by setting $q(x) = 0$ if $(x_r, y_r) = G[x^4]$ for any valid index combination, and ∞ otherwise. Then, the same logic from the known map DPA is applied: for every state-action pair, we compute $x' = f(x, u)$, assign cost-to-go $Q_t(x, u)$, and select the action minimizing Q.

Once computed, this general value function and policy $\pi_t(x)$ are stored and used across all random environments by indexing into the correct initial state defined by the sampled map. This approach allows `partB()` to run inference using a single optimized policy while supporting generalization over all valid random configurations.

Algorithm 2 describes the extraction of the optimal sequence for each random map after the value function is pre-computed. The value function is pre-computed using Algorithm 1 but with the addition of a larger state space.

Algorithm 2 Policy Extraction for Random Map Instance

```
Input: Random map $M$, value function $V$, time horizon $t$
Get environment info: $x_0 = (x, y, d, g_i, k_i, d_1, d_2, 0)$
Initialize policy trajectory $\pi^* = [\ ]$
while $V_{t-1}(x_0) > 0$ do
    for all $u \in \mathcal{U}$ do
        $x' = f(x_0, u)$
        $Q[u] = V_{t-1}(x')$
    end for
    if all $Q[u] = \infty$ then
        terminate (no valid moves)
    end if
    $u^* = \arg\min Q$
    Append $u^*$ to $\pi^*$
    $x_0 = f(x_0, u^*)$
end while
return optimal sequence $\pi^*$ =0
```

IV. Discussion

A. Known Environment MDP

The biggest roadblock in implementing and solving the known environment MDP problem was simply defining the state space and stage costs. Initially, I was under the impression that the states simply meant the location of the robot on the grid and their orientation. I planned to complement this with smart stage costs definition such that picking up the key and opening the door could result in a reward (negative cost). I believe that this would probably solve the problem in environments where the door is closed and they must open the door. However, I realized that this would always return a suboptimal path when there is a shortcut, as the robot is always rewarded for picking up the key and opening the door.

After realizing that this would be rather ineffective, I thought about encoding the key state and door state within the state space. I was initially doubtful about incorporating the door state within the state space, as this felt rather ”cheaty” in the sense that this prior knowledge gives it an advantage. However, I realized that the problem is that of an MDP and not a POMDP, which means we have full knowledge of the environment at all times so this is a valid approach.

Other issues were simply minor implementation issues, such as forgetting zero indexing for the grid, which resulted in my check valid function returning a valid move whenever the robot goes into a wall, since the comparison I was doing didn't account for zero indexing.

B. Random Environment MDP

My biggest issues with the random environment MDP can be split into two main issues which ended up with the same solutions.

The first obstacle I ran into was that my robot would terminate whenever it hit one of the possible goal locations, even if the goal for that current environment is elsewhere (typically if it is further away). The reason for was the definition of my terminal costs. In the previous problem, it was sufficient to define the terminal costs as 0 whenever the state coordinates are within the goal coordinates and independent of the other state values. This works for the previous problem as there was only ever one goal location and the policy is found for each map. Moving on to the random environment problem, I kept the same definition for my stage costs and found the issue described previously. The solution for this was to also include the goal location associated with each goal's coordinates. In the mathematical sense, this can be expressed as below.

$$q(x) = \begin{cases} 0 & \text{if } (x_r, y_r) = G[x_t^4] = (x_g, y_g), \\ & \quad \text{and for any valid } x_t^5, x_t^6, x_t^7, x_t^8 \\ \infty & \text{otherwise} \end{cases}$$

In words, the terminal cost is only defined for x and y coordinates associated with that specific goal.

The second obstacle I ran into was that the robot would seemingly always find the optimal path whenever there is at least one door open. However, as soon as both doors are closed initially, then the robot fails to find any valid moves. It turns out, the issue comes again from the definition of the terminal costs. I first started encoding the door states for the terminal states as this account for situations where only one door needs to be open to find the optimal path. The second change comes from a mistake where I was declaring only states for which the robot is not holding a key ($x_t^8 = 0$)) as terminal. This was a big mistake because this meant that the robot can only terminate when both doors are closed and the robot is not holding a key. This is impossible because the robot can never open a door without holding a key, thus it will never find a path. This was not an issue in the previous case because a door open means the robot never has to pick up a key and thus, this was a valid terminal state. It is also important to account for the key index associated with that specific environment as this ensures that the optimal path is taken for each random environment.

$$q(x) = \begin{cases} 0 & \text{if } (x_r, y_r) = G[x_t^4] \text{ (goal index match),} \\ & \quad \text{for any } x_t^5 \in \{0, 1, 2\} \text{ where } K := \{[2,2],[2,3],[1,6]\} \\ & \quad x_t^6, x_t^7 \in \{(0,1),(1,0),(1,1)\} \text{ (door states),} \\ & \quad x_t^8 \in \{0, 1\} \text{ (key possession)} \\ \infty & \text{otherwise} \end{cases}$$

V. Results

A. Known Environment MDP

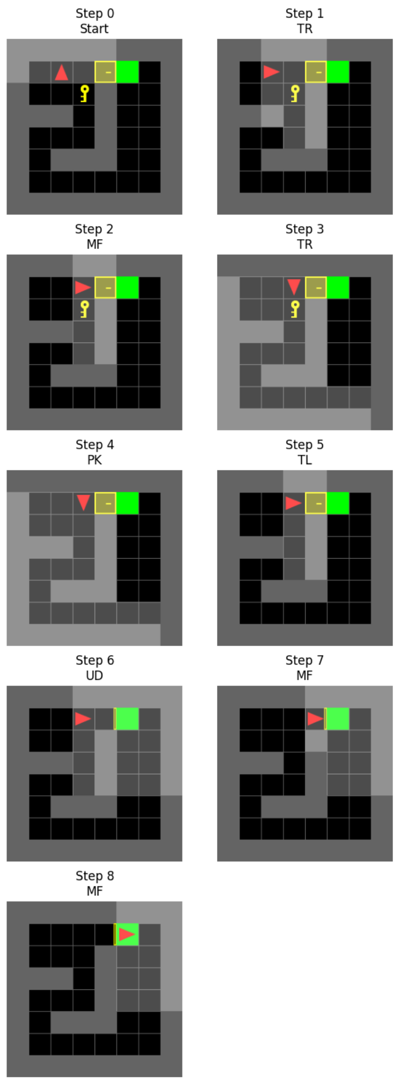

Fig. 2: Optimal Control Sequence for Doorkey-8x8-shortcut

Here is an example of solving a known environment MDP problem. The GIFs and control sequences for each known map can be found in the code submission.

B. Random Environment MDP

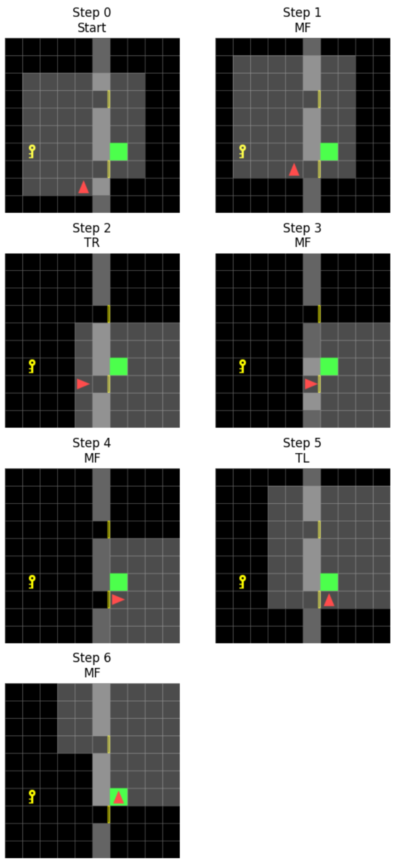

Fig. 3: Optimal Control Sequence for Doorkey-10x10-33

Here is an example of solving a random environment MDP problem. The GIFs and control sequences for each random map can be found in the code submission.

In general, the implementation of dynamic programming for both known environments and random environments successfully generates an optimal control sequence.