Project 1: Solving Door & Key Problem using Dynamic Programming

Khagesh Bhardwaj
Jacob School of Engineering
University of California
San Diego, California 92093
kbhardwaj@ucsd.edu

Guide: Prof. Nikolay Atanasov
Jacob School of Engineering
University of California
San Diego, California 92093
natanasov@ucsd.edu

***Abstract*—This paper addresses a important aspect of autonomous navigation systems through the Door & Key project, where a simulated agent—a red triangle—must efficiently reach a green square goal while navigating dynamic obstacles, such as doors that may require keys to unlock. We formulate this navigation challenge as a Markov Decision Process (MDP) and employ Dynamic Programming (DP) techniques to derive optimal policies. Our approach systematically optimizes the agent's path by minimizing costs associated with interaction with the environment. We evaluate our algorithms in both known and randomly generated environments, demonstrating that while known environments allow for rapid computation of policies (1-2 seconds), the complexity significantly increases in random environments, taking 25-30 seconds due to a larger state space. The study not only showcases the adaptability of the algorithm in various scenarios but also discusses potential applications in industrial robotics and autonomous vehicle navigation. Results confirm the effectiveness of the developed policies, underscoring the practical implications for real-world autonomous systems.**

I. Introduction

The Door & Key project addresses a fundamental challenge in autonomous navigation systems: enabling an agent to navigate efficiently through an environment with dynamic obstacles and varying goals. In this case, the challenge is embodied in a simulated environment where a red triangle (the agent) must reach a green square (the goal) while potentially needing to retrieve a key to unlock doors if that block its path as shown in Fig. 2.

A. Importance of the Problem

Autonomous navigation is crucial in numerous real-world applications, ranging from robotics in industrial settings to autonomous vehicles and drones in urban environments. The ability to navigate efficiently while interacting with objects in the environment is essential for the development of fully autonomous systems. This project not only explores these capabilities but also addresses the optimization of resource consumption, such as energy, which is vital for practical implementations.

B. High-Level Overview of Approach

The approach taken in this project involves formulating the navigation task as a Markov Decision Process (MDP). This formulation allows us to systematically explore the decision-making process required for the agent to achieve its goal with minimal cost. We implement Dynamic Programming (DP) techniques to solve this MDP, specifically focusing on designing and implementing algorithms that can find optimal policies for both known and randomly generated environments.

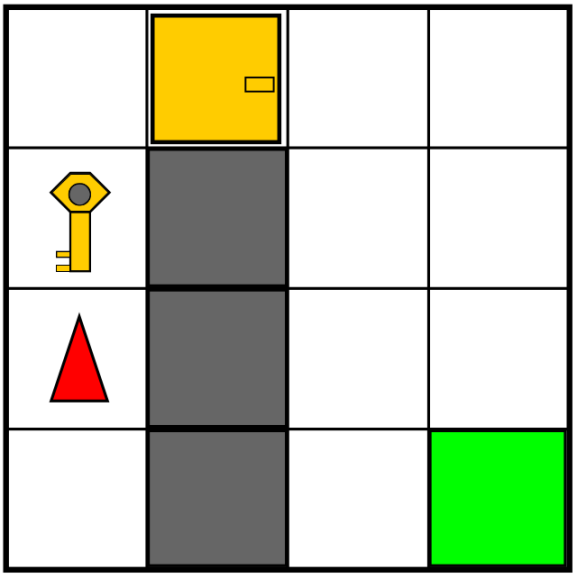

Fig. 1. Door & Key Environment

C. Outline

The outline of this report is structured into several sections. In Section II, we will define the problem explicitly as an MDP, detailing the state space, action space, motion model, and stage & terminal cost functions. Moving on to Section III, we then describe the technical algorithm developed to solve the MDP, including how the value function & policies are computed and applied in different scenarios. Following this, Section IV presents the results from applying these policies in both known and random map scenarios, supported by optimal path in the environment. The next section presents a discussion of the challenges and observations. The subsequent section extends gratitude in the acknowledgement, recognizing the

support of individuals who contributed to the project. The final section, References, lists the sources that played a crucial role in completing this task.

II. Problem Formulation

Objective: The objective of the problem is to provide an optimal policy for a given environment which enables the agent to navigate from the starting location to the goal optimally. This involves minimizing the total cost while efficiently reaching the goal.

Map Overview

1) In the **Known Map** scenario, the environment layout includes an agent, doors (which can be either open or closed), a key, walls, floors, and the goal. This environment is static and fully known, allowing the agent to utilize complete information to navigate and interact with elements like doors and keys to reach the goal. We are given with 7 known environments and have to provide optimal pocily for each individually.
2) The **Random Map** scenario features a dynamically generated 8×8 grid environment surrounded by walls, with specific elements varying:
   - **Agent's Starting Position and Orientation**: Located at (3, 5), facing up.
   - **Doors**: Positioned at (4, 2) and (4, 5), which can be either open or closed.
   - **Key Locations**: Possible placements at (1, 1), (2, 3), or (1, 6).
   - **Goal Locations**: Could be located at (5, 1), (6, 3), or (5, 6).
   - **Obstacle**: A vertical wall along column 4.

   This configuration leads to 36 potential environment variations. The challenge is to develop a SINGLE control policy that adapts to these changes and ensures efficient navigation to the goal under varying conditions.

Markov Decision Process (MDP

A Markov Decision Process (MDP) is defined using the framework of a Markov chain, represented by the tuple $(\mathcal{X}, \mathcal{U}, x_0, pf, T, \ell, q, \gamma)$, where:

- $\mathcal{X}$ is the **discrete state space**, containing all possible states in the environment.
- $\mathcal{U}$ is the **discrete control space**, including all possible actions the agent can take.
- pf is the **deterministic motion model**, mapping state-action pairs to subsequent states.
- T represents the **time horizon**, the total duration over which decisions are made.
- ℓ is the **stage cost**, incurred for taking action u from state x.
- q is the **terminal cost** at time T, associated with being in state x at the end of the decision process.

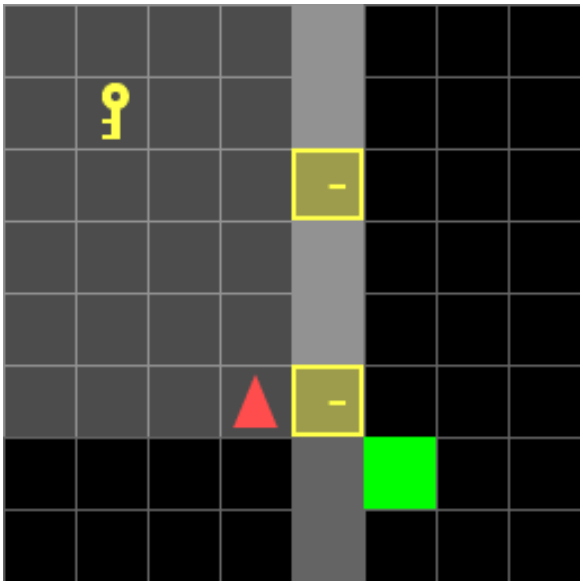

Fig. 2. One possible scenario from 8X8 random environment

- γ is the **discount factor**, which in our case is set to 1, emphasizing the equal importance of all future costs without discounting.

Assumptions: The model assumes that there are no negative cycles; all actions carry a positive cost. Consequently, the optimal path to the goal cannot involve more actions than the total number of states minus one ($|\nu| - 1$).

A. State Space Definition($\mathcal{X}$))

The state space for the MDP is the set of all possible states which the agent can attain in the environment. We categorize the problem into two scenarios: the known and the random environment.

Known Environment ($\mathcal{X}_{\text{known map}}$):
In Part A, the known environment, the state space is explicitly defined and includes the following components:

$$\mathcal{X}_{\text{known map}} = [\text{x, y, heading, key}_{\text{state}}, \text{door}_{\text{state}}]$$

where:

- x and y: Robot's location coordinates within the grid.
- heading: Direction the robot is facing; [0,1,2,3].
- key$_{state}$: Key availability ([0,1]).
- door$_{\text{state}}$: Door state (close/open) ([0,1]).

The dimension of the state space is $x \times y \times 4 \times 2 \times 2$.

Random Environment ($\mathcal{X}_{\text{random map}}$):
For the random environment, the state space is enhanced to account for the state of additional door, possible key location

and possible goal locations:

$\mathcal{X}_{\text{random map}}$ = [x, y, heading, $\text{key}_{\text{state}}$, $\text{door1}_{\text{state}}$, $\text{door2}_{\text{state}}$, key_{loc}, goal_{loc}]

where:

- x_{agent} and y_{agent}: Coordinates of the agent within 8x8 grid.
- heading: Heading direction; [0,1,2,3].
- key_{state}: Key availability ([0,1]).
- $\text{door1}_{\text{state}}$: Door 1 state (close/open) ([0,1]).
- $\text{door1}_{\text{state}}$: Door 1 state (close/open) ([0,1]).
- $\text{key}_{\text{location}}$: Index of the possible key locations; ([0,1,2]).
- $\text{goal}_{\text{location}}$: Index of the possible goal locations; ([0,1,2]).

This enhanced state space supports a more involved decision-making process to accommodate the variability and dynamic aspects of the key and goal locations.

B. Input Controls ($\mathcal{U}$)

The input controls for both the known and random environment scenarios are defined as follows:

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

where:

- MF (Move Forward): The agent advances one grid cell in the direction it is currently facing.
- TL (Turn Left): The agent rotates its orientation 90 degrees to the left.
- TR (Turn Right): The agent rotates its orientation 90 degrees to the right.
- PK (Pick Key): A special control allowing the agent to pick up a key if the key is in the next cell where its headed.
- UD (Unlock Door): Another special control that enables the agent to unlock a door if the door is in the next cell where agent is headed.

These controls constitute the action space $\mathcal{U}$ that the agent can execute to interact with the environment, each impacting the state of the system as per the defined motion model.

C. Motion Model (p_f)

The motion model for the MDP is deterministic, defined by transition probabilities pf which are:

$$pf = \begin{cases} 0 & \text{if the action is not allowed in the current state,} \\ 1 & \text{if the action is allowed and changes the state.} \end{cases}$$

This model simplifies the prediction of state transitions, ensuring that actions permitted by the environment deterministically lead to new states as formulated in algorithm 1.

Algorithm 1 Motion Model (Determine Next State)

procedure NEXTSTATE(current state, action, env, info)
  if action == MF **then**
    if $0 \leq x <$ grid dim and $0 \leq y <$ grid dim **then**
      if wall ahead **then**
        Do not move
      else if door ahead and locked **then**
        Do not move
      else
        Move ahead
      end if
    end if
  else if action == TL **then**
    Turn left
  else if action == TR **then**
    Turn right
  else if action == PK and (key ahead and agent does not have the key) **then**
    Pick the key
  else if action == UD and (have the key and door ahead) **then**
    Toggle the door state
  end if
  return NextState
end procedure

D. Intial State (x_0)

The initial state of the agent is derived from the environment using the 'info' attribute. In the known environment, the initial state varies, while in the random environment, it remains consistent. Specifically, for random maps, the agent starts at position (3,5) and is oriented upwards.

E. Time Horizon (T)

The MDP operates within a finite time horizon, assumed on the guarantee that a path to the goal exists within a limited number of actions. The specific length of this time horizon is dependent on the particular settings of the environment info.

F. Stage Cost, l

To streamline the analysis, we've standardized the cost for all actions taken by the agent and assumed zero cost when the agent is at the goal location. Building on this simplification, we have developed an algorithm 2 to calculate the stage cost based on the state x and control u used by the robot.

G. Terminal Cost q(x, T)

Terminal cost represents the cost incurred to reach the goal location at the final time step, q(x,T). Our aim is to ensure the agent arrives at the goal location as the time horizon ends. Accordingly, we define the terminal cost at the goal location as $q(\text{goal}, T) = 0$ and assign a value of ∞ to all other locations.

H. Discount Factor (γ)

We have taken discount factor as 1 for our decision process.

Algorithm 2 Calculate Stage Cost

```
procedure StageCost(x, y, goal location)
    if (x, y) = goal location then
        cost = 0
    else
        cost = 1
    end if
    return cost
end procedure
```

III. Technical Approach

In optimal control problems, we aim to determine the best sequence of actions to minimize the measure of long-term cost starting from an initial state, and/or collecting the key, unlocking the door and finally reaching the goal.

We modeled our problem as a Markov Decision Process (MDP) the key elements of the MDP are as follows:

- **State Space** (X): The set of all possible states.
- **Control Input Space** (U): The set of all possible control actions.
- **Initial State** (x_0): The initial states.
- **Motion MOdel** (p_f): Defines the agent's location moving from one state to another under a given action.
- **Horizon** (T): The number of time steps considered in the decision process.
- **Stage Cost** (ℓ): The immediate cost incurred from taking an action at a state.
- **Terminal Cost** (q): The cost associated with the final state at time T.
- **Discount Factor** (γ): A factor that discounts future costs, reflecting the value of immediate versus future rewards.

The Dynamic Programming algorithm is an approach used to solve the optimal control problem in a Markov Decision Process (MDP) framework. To formulate a dynamic programming algorithm, let's define it's key components:

A. Control Policy

A control policy π is defined as a function that maps a time step $t \in \mathbb{N}$ and a state $x \in X$ to a feasible control input $u \in U$:

$$\pi : \mathbb{N} \times X \to U$$

B. Value Function

The value function $V_t^{\pi}(x)$ associated with a policy π measures the expected long-term cost starting in state x at time t under policy π:

$$V_t^{\pi}(x)$$

C. Optimal Control Problem

The optimal control problem seeks to find a policy π^* that minimizes the expected long-term cost from an initial state x_0:

$$V_0^*(x_0) = \min_{\pi} V_0^{\pi}(x_0)$$

where π^* is the optimal policy, defined as:

$$\pi^* \in \arg\min_{\pi} V_0^{\pi}(x_0)$$

D. Dynamic Programming Algorithm

Dynamic programming offers a systematic approach to solve this problem by calculating the optimal value function and policy backwards in time:

- It handles non-linear, non-convex problems efficiently.
- It has a complexity that is polynomial in the number of states $|X|$ and number of actions $|U|$.
- It is significantly more efficient than brute-force methods that evaluate all possible policies.

The **Principle of Optimality** is essential in understanding dynamic programming approaches to solving decision-making problems. It states:

- Let $\pi_0^*, \ldots, \pi_{T-1}^*$ be an optimal control policy sequence.
- Consider a subproblem starting at time t and state x, defined as:

$$V_{\pi}^t(x) = \mathbb{E}_{t+1:T}\Bigg[\gamma^{T-t} q(x_T) + \sum_{\tau=t}^{T-1} \gamma^{\tau-t} \ell(x_\tau, \pi_\tau(x_\tau)) \mid x_t = x\Bigg]$$

  where, γ = 1,

$$V_{\pi}^t(x) = \mathbb{E}_{t+1:T}\left[q(x_T) + \sum_{\tau=t}^{T-1} \ell(x_\tau, \pi_\tau(x_\tau)) \mid x_t = x\right]$$

- The principle of optimality asserts that if π^* is optimal for the entire horizon, then the truncated policy $\pi_{t:T-1}^*$ must be optimal for the subproblem beginning at t with state x.

Utilizing the Principle of Optimality, we compute the value function backward in time, determining the optimal policy for each subproblem. Concatenating these optimal sub-policies yields the complete optimal policy for the entire decision-making horizon. The Dynamic Programming algorithm 3 is detailed, explaining the systematic approach for deriving the optimal policy and value function in a Markov Decision Process (MDP).

During each iteration over time, we compute the optimal control input for all states by minimizing the stage cost and identifying the optimal control action. Convergence to a solution is achieved when the value function stabilizes, meaning that $V_t(i) = V_{t+1}(i)$ for all states i in the set $V \setminus \{\tau\}$. At this point, it is said that the corresponding optimal policy has been determined.

Once the policy is established across the entire state space, it is then applied in any given random environment to determine the specific optimal sequence of actions.

IV. Results

In this section, we have calculated and presented the optimal policy for both known and random environments. The algorithm operates very quickly in the known environment,

Algorithm 3 Dynamic Programming for MDPs

Input: MDP $(\mathcal{X}, \mathcal{U}, x_0, p_f, T, \ell, q, \gamma)$
Initialize:
$V_T(\tau) = 0$ ▷ Goal state has zero terminal cost
$V_T(i) = \infty, \forall i \in X \setminus \{\tau\}$ ▷ Non-goal states initialized with infinite terminal cost
for $t = T - 1$ to 0 **do**
  for all $x \in X$ **do**
    for all $u \in U(x)$ **do**
      Calculate $Q_t(x, u) = \ell(x, u) + \gamma \sum_{x'} p_f(x' \mid x, u) V_{t+1}(x')$
    end for
    $V_t(x) = \min_{u \in U(x)} Q_t(x, u)$
    $\pi_t(x) = \arg\min_{u \in U(x)} Q_t(x, u)$
  end for
end for
return policy $\pi_{0:T-1}$ and value function V_0

requiring only 1-2 seconds to compute the complete policy. However, in the random environment, the computation time increases significantly, taking approximately 25-30 seconds to determine the optimal policy. This difference in computation time can be attributed to the increase in the number of states.

In the known environment, we handle a maximum of 8×8×4×2×2=1024 states and 5 control inputs. In contrast, the random environment involves 1024×2×3×3=18432 states and the same number of control inputs.

Ultimately, the algorithm successfully generates an optimal policy that guides the agent from the starting location to the goal location efficiently.

A. Part A

In this subsection, we discuss how the variable component, the environment input, is managed. All environments are stored in a dictionary, and we sequentially execute each one to determine optimal policies. The results, including the **optimal policies for all environments, are detailed in the captions** of Figures 2 through 15. These figures illustrate the initial and final states of the environment for each of the 7 known scenarios.

B. Part B

This subsection discusses four specific scenarios out of the 36 possible cases, focusing on variations in door states. We explore the following configurations: first, with both doors open (17, 18); second, with door 1 open and door 2 closed (19, 20); third, with door 1 closed and door 2 open (21, 22); and lastly, with both doors closed (23, 24).

V. Discussion

A. Part A

Part A consist of 7 different known environment scenarios. In each scenario, the agent navigates optimally, selecting the best path to the goal. The scenarios are categorized into three types: normal, direct, and shortcut.

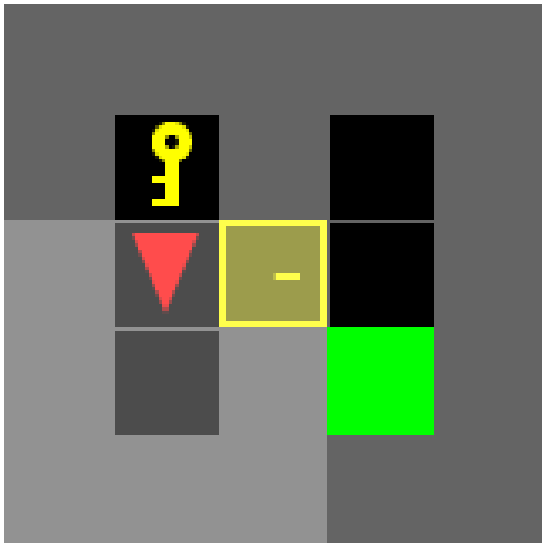

Fig. 3. Doorkey-5x5-normal; Optimum Policy: ['TL', 'TL', 'PK', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF']

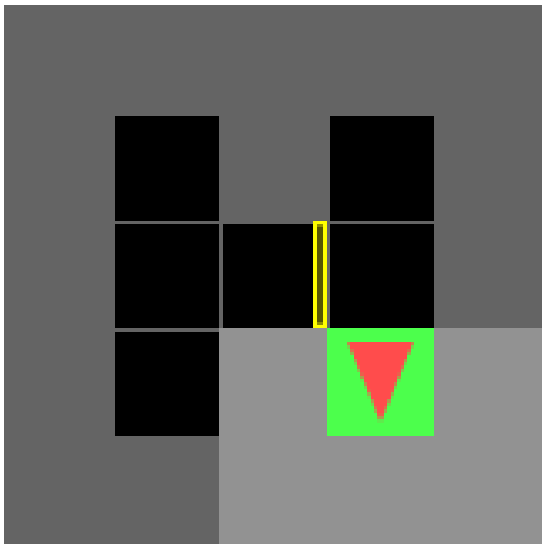

Fig. 4. Doorkey-5x5-normal; Agent reached the goal following the Optimum Policy

*1) **Normal**:* In the normal scenario, the agent must obtain a key before it can reach the goal. The results confirm the expected behavior: the agent consistently collects the key, opens the door, and then proceeds to the goal location.

*2) **Direct**:* In the direct scenario, a pathway exists that allows the agent to reach the goal directly without needing to collect a key. Observations show that the agent behaves accordingly—it heads straight for the goal without detouring to collect the key or open any doors.

*3) **Shortcut**:* In the shortcut scenario, the agent evaluates potential shortcuts to minimize its overall cost, as dictated by its value function. The agent must decide whether to head

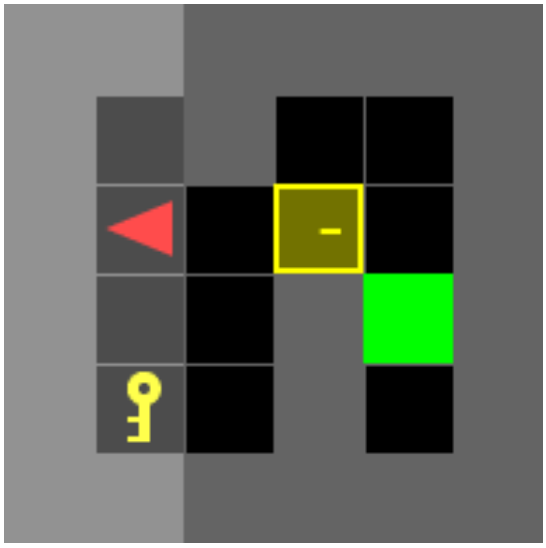

Fig. 5. Doorkey-6x6-normal; Optimum Policy: ['TL', 'MF', 'PK', 'TL', 'MF', 'TL', 'MF', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF']

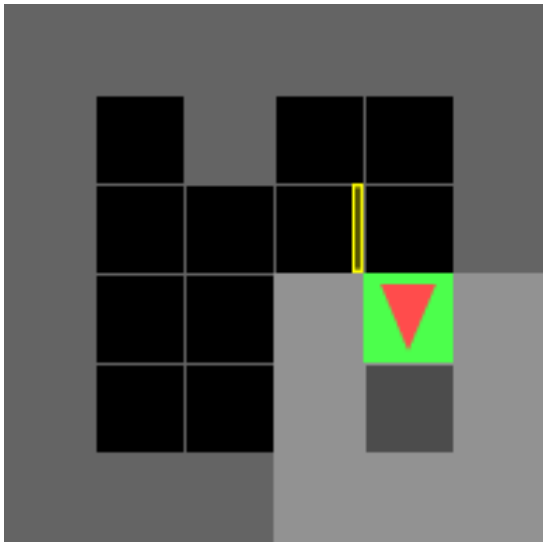

Fig. 6. Doorkey-6x6-normal; Agent reached the goal following the Optimum Policy

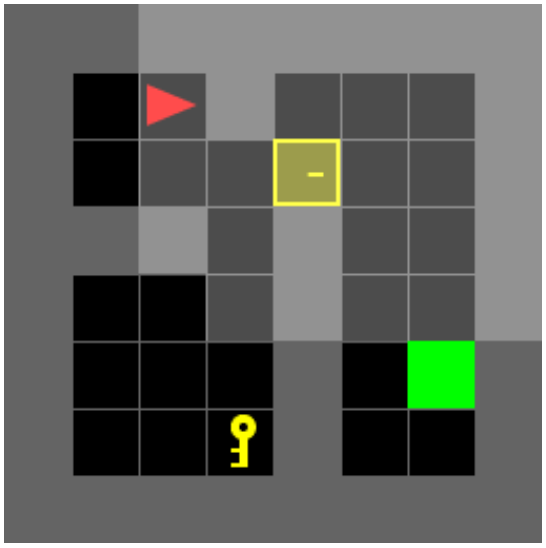

Fig. 7. Doorkey-8x8-normal; Optimum Policy: ['TR', 'MF', 'TL', 'MF', 'TR', 'MF', 'MF', 'MF', 'PK', 'TL', 'TL', 'MF', 'MF', 'MF', 'TR', 'UD', 'MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF']

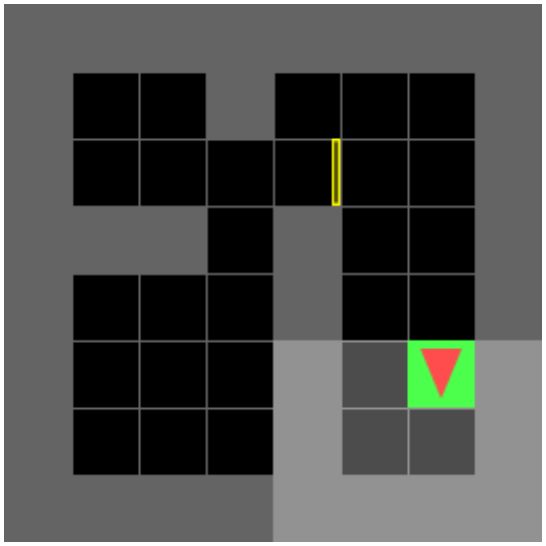

Fig. 8. Doorkey-8x8-normal; Agent reached the goal following the Optimum Policy

directly to the goal or to collect the key, open the door, and then proceed to the goal. Consistent with the principles of the Dynamic Programming algorithm, the agent opts for the route that reduces its total cost.

B. Part B

This subsection discusses four specific scenarios out of the possible 36 cases, focusing on variations in the states of the doors. We analyze the agent's behavior in each configuration to determine the efficiency of its navigation strategy.

In the first scenario, where **both doors are open**, the agent is expected to reach the goal in the fewest possible steps. This expectation is met, as there is no need for the agent to pick up the key or toggle any doors.

In the second scenario, **door 1 is open while door 2 is closed**. Despite the agent's proximity to the key and the locked door, it opts not to pursue that route. Instead, it takes a longer path through the open door to minimize the overall cost as it involves less number of actions.

The third scenario features **door 1 being locked and door 2 open**. Here, the agent bypasses the locked door entirely as expected, choosing instead to pass through door 2, avoiding the need to collect the key or unlock door 1.

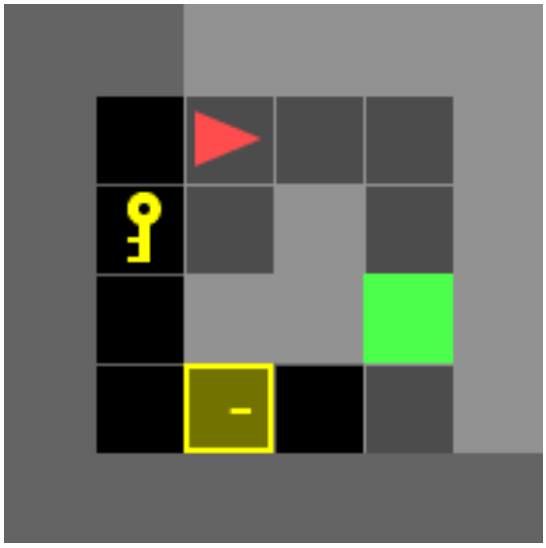

Fig. 9. Doorkey-6x6-direct; Optimum Policy: ['MF', 'MF', 'TR', 'MF', 'MF']

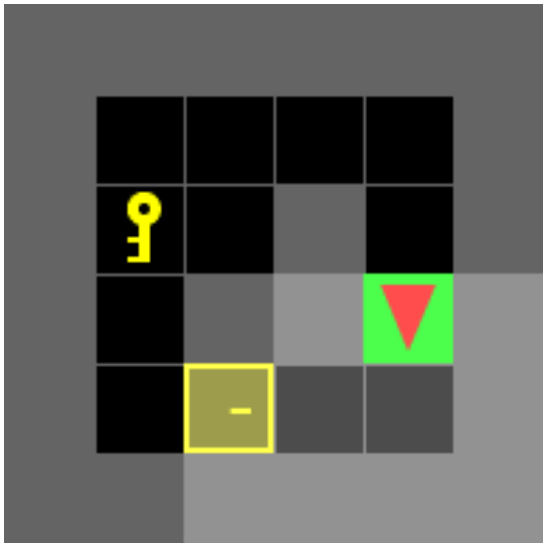

Fig. 10. Doorkey-6x6-direct; Agent reached the goal following the Optimum Policy

In the final scenario, where both doors are locked, the agent is required to pick up the key, unlock a door, and proceed to the goal. It successfully follows this sequence, reaching the goal as anticipated.

These four cases illustrate the agent's ability to adapt its strategy to different environmental conditions, ensuring optimal policy execution that minimizes the value function in each scenario. Other cases not detailed here behave similarly, sticking to strategies that yield the minimum value function and optimal outcomes.

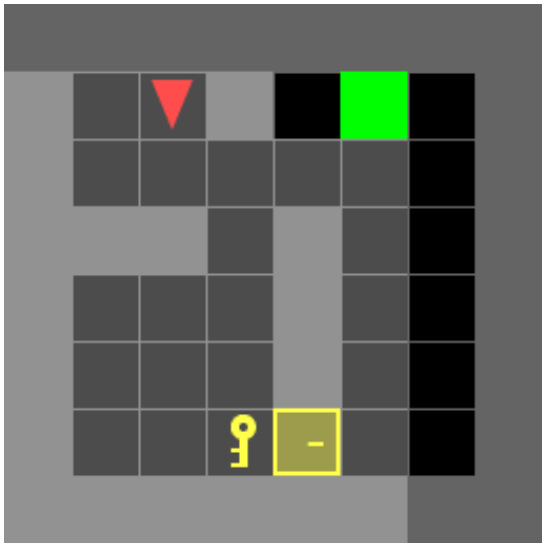

Fig. 11. Doorkey-8x8-direct; Optimum Policy: ['MF', 'TL', 'MF', 'MF', 'MF', 'TL', 'MF']

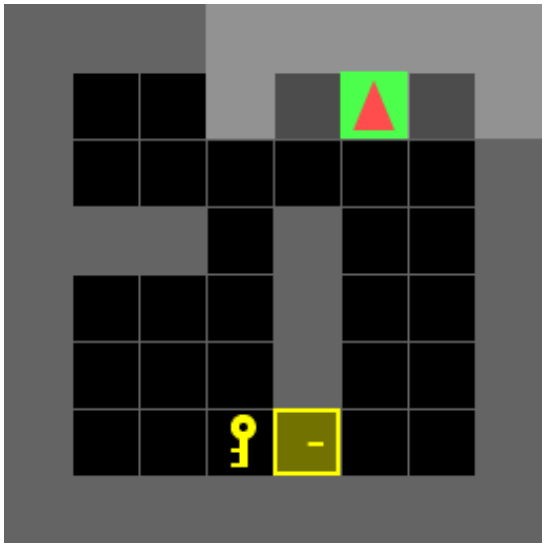

Fig. 12. Doorkey-8x8-direct; Agent reached the goal following the Optimum Policy

C. Conclusion

This project successfully applied a Dynamic Programming algorithm to determine optimal policies for the agent in both known and random environments. We have some key findings from the project, which includes:

1) The algorithm quickly computed optimal policies in known environments (1-2 seconds) and managed a significantly larger state space in random environments (25-30 seconds).
2) The agent adapted its strategy to minimize costs under varying environmental conditions, efficiently navigating through both open and locked doors.
3) In scenarios with open doors, the agent took the most direct path to the goal. When faced with closed doors, it

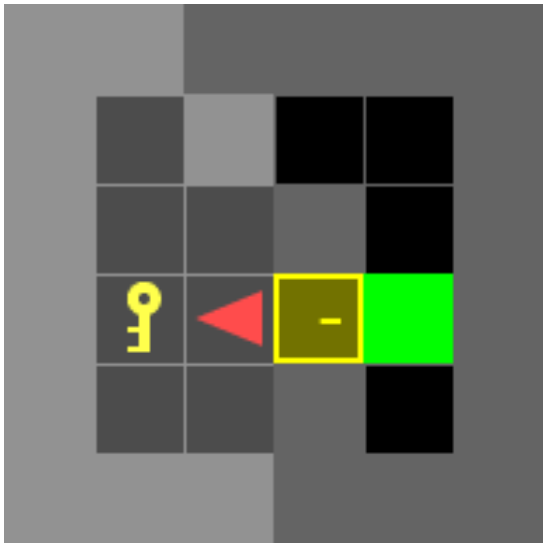

Fig. 13. Doorkey-6x6-shortcut; Optimum Policy: ['PK', 'TL', 'TL', 'UD', 'MF', 'MF']

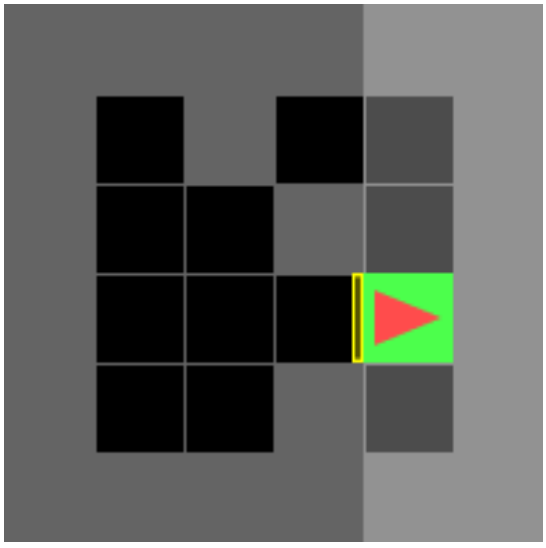

Fig. 14. Doorkey-6x6-shortcut; Agent reached the goal following the Optimum Policy

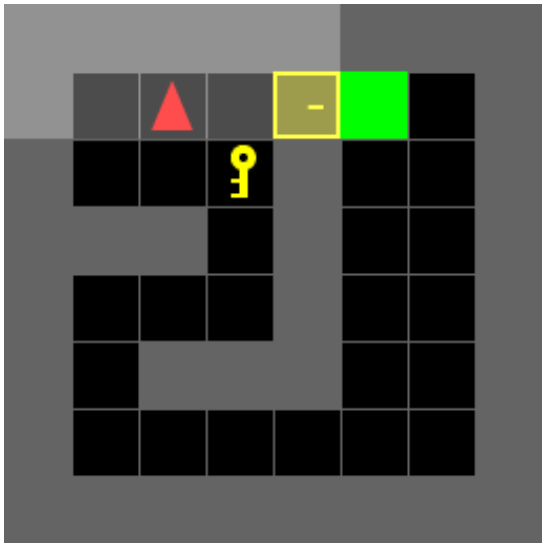

Fig. 15. Doorkey-8x8-shortcut; Optimum Policy: ['TR', 'MF', 'TR', 'PK', 'TL', 'UD', 'MF', 'MF']

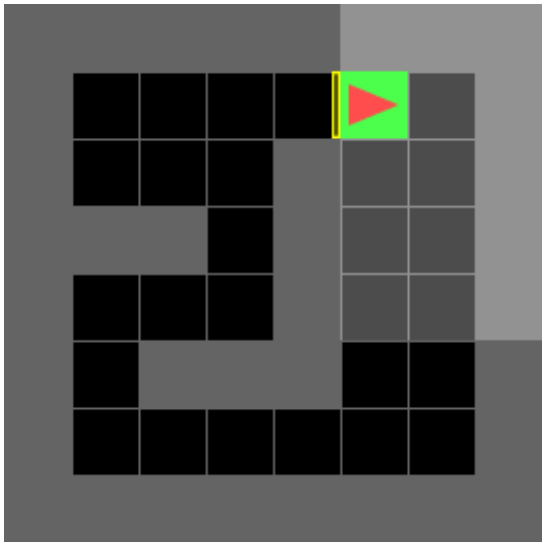

Fig. 16. Doorkey-8x8-shortcut; Agent reached the goal following the Optimum Policy

evaluated alternative routes to avoid unnecessary actions or utilized keys strategically.

In summary, the Dynamic Programming approach proved effective for autonomous navigation, demonstrating robustness and adaptability across different environment settings, which supports its potential for broader applications.

ACKNOWLEDGMENT

I would like to express my sincere appreciation to Professor *Nikolay A. Atanasov* for his invaluable guidance and *Zhirui Dai* for their continuous support throughout this project. Their expertise and assistance have been crucial to its success. I am grateful for the mentorship provided and the insights gained from their valuable feedback.

REFERENCES

[1] ECE276BLectureSlides

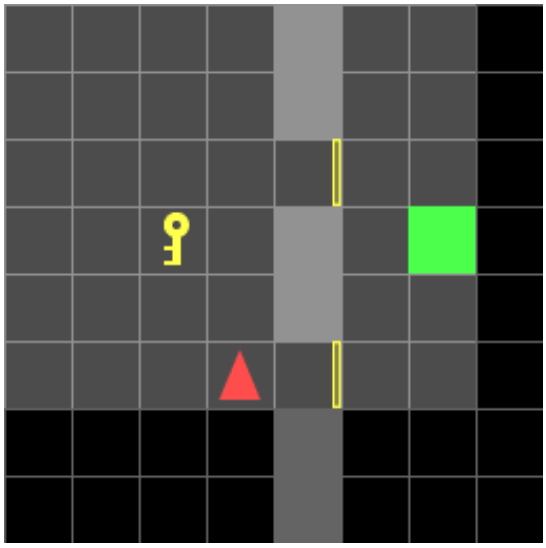

Fig. 17. Doorkey-8x8-17; Optimum Policy: ['TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF']

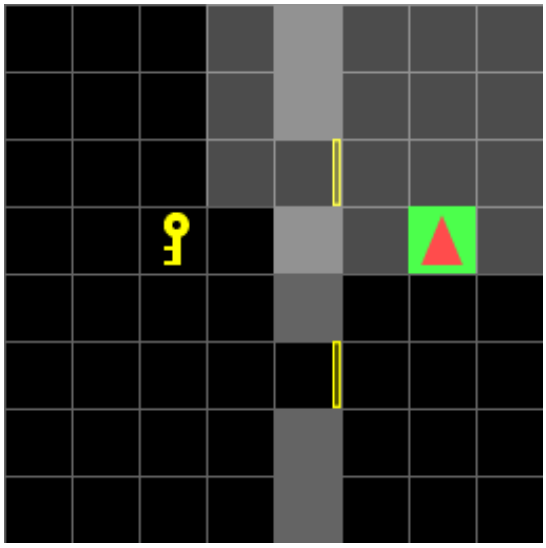

Fig. 18. Doorkey-8x8-17; Agent reached the goal following the Optimum Policy

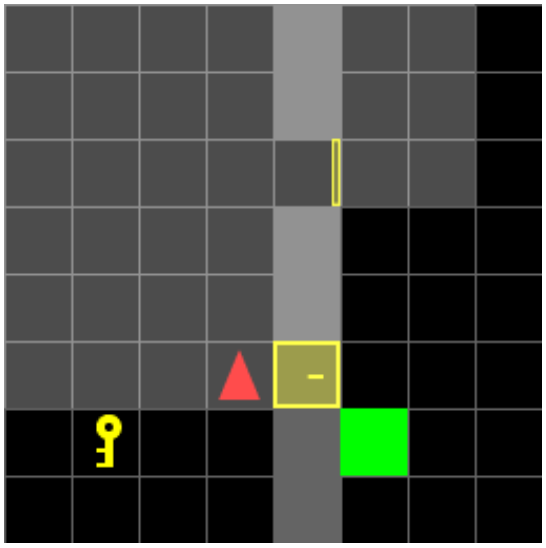

Fig. 19. Doorkey-8x8-34; Optimum Policy: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'MF']

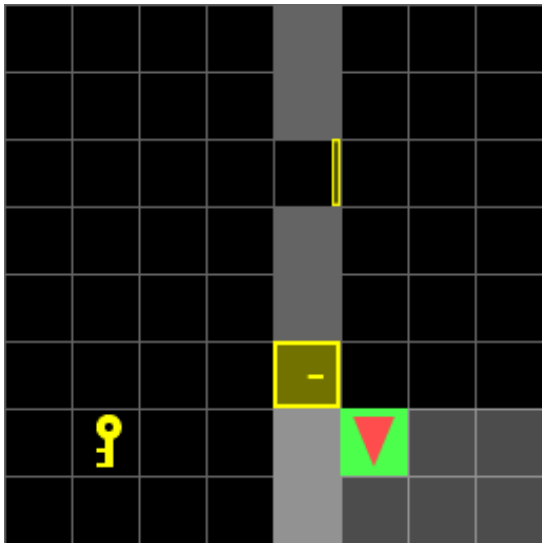

Fig. 20. Doorkey-8x8-34; Agent reached the goal following the Optimum Policy

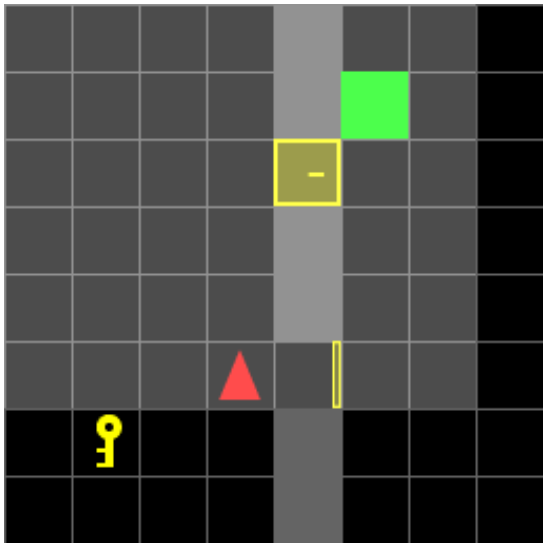

Fig. 21. Doorkey-8x8-27; Optimum Policy: ['TR', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF']

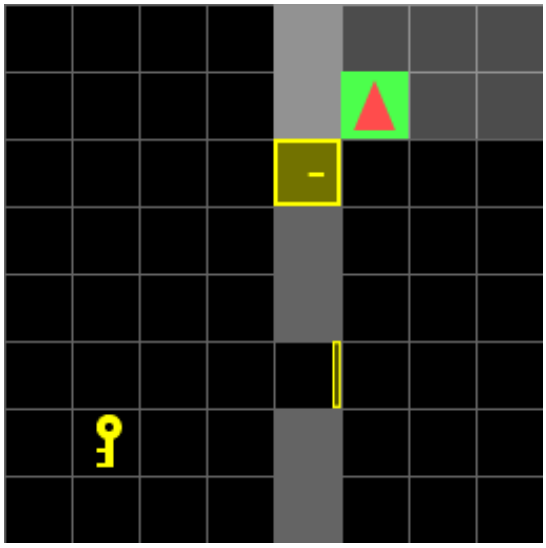

Fig. 22. Doorkey-8x8-27; Agent reached the goal following the Optimum Policy

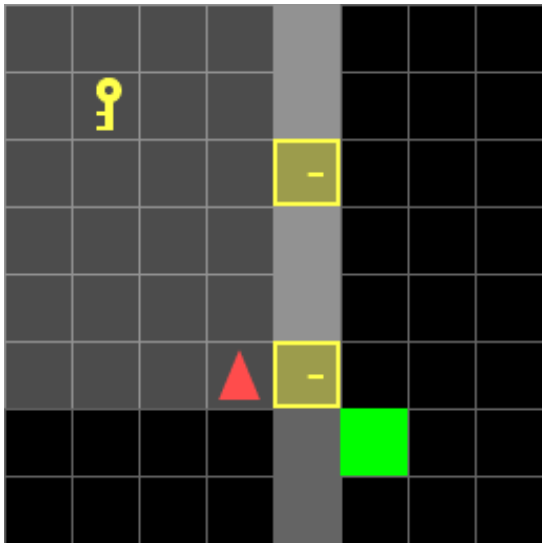

Fig. 23. Doorkey-8x8-12; Optimum Policy: ['MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'PK', 'TL', 'MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF']

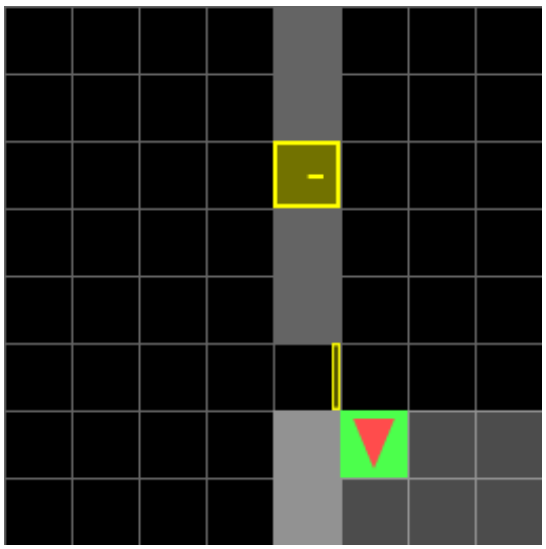

Fig. 24. Doorkey-8x8-12; Agent reached the goal following the Optimum Policy