Optimizing Autonomous Navigation through Dynamic Programming: Strategies for Door & Key Environments

Yixin Zhang

I. Introduction

This paper explores autonomous navigation within a Door & Key environment, focusing on the challenge of navigating an agent (represented as a red triangle) to a goal location (depicted as a green square). The environment may feature a door that obstructs the path to the goal. If the door is closed, the agent is required to pick up a key to unlock the door. This scenario is formulated as an optimal control problem, addressed through dynamic programming algorithms designed to find the path of minimal effort enabling the agent to reach the goal.

Dynamic programming offers substantial benefits in robotics, particularly for autonomous navigation and decision-making in unpredictable environments. Robots utilize these algorithms to adeptly maneuver through complex terrains and interact with dynamic obstacles, enhancing their efficiency in critical applications such as disaster response and industrial automation. Moreover, this method supports the development of autonomous vehicles and service robots, enabling them to perform tasks with high precision and adaptability. Whether guiding drones through variable agricultural landscapes or assisting in healthcare settings, dynamic programming equips robots with the capabilities necessary to operate independently and effectively in a variety of challenging scenarios.

We demonstrate the correctness and efficiency of our algorithms across two types of environments:

- **Known Map:** We compute specific control policies for each of the seven environments provided in the starter code, evaluating their performance on the corresponding environment.
- **Random Map:** A single control policy is computed, capable of adapting to any of the 36 randomly generated 8×8 environments, showcasing the scalability and adaptability of our approach.

II. Problem Formulation

The finite-horizon optimal control problem in a Markov Decision Process (MDP) $(\mathbb{X}, \mathbb{U}, p_0, p_f, T, \ell, q, \gamma)$ with initial state x at time t is defined as:

$$\min_{\pi_{0:T-1}} V_0^{\pi}(x) := \gamma^{T-t} q(x_T) + \sum_{\tau=0}^{T-1} \gamma^{\tau-t} \ell(x_\tau, \pi_\tau(x_\tau))$$

subject to the transition:

$$x_{\tau+1} \sim p_f(\cdot \mid x_\tau, \pi_\tau(x_\tau)),$$
$$x_\tau \in \mathbb{X}, \quad \pi_\tau(x_\tau) \in \mathbb{U}, \quad \tau = 0, \ldots, T-1$$

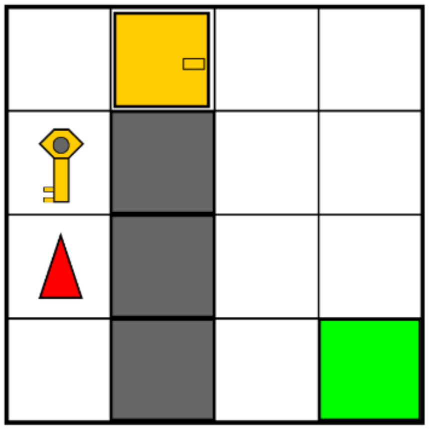


Fig. 1: Environment Overview

The specifics of the MDP are detailed in the subsections for Known and Random Environments.

A. Known Environment

1) State Space $\mathbb{X}$: The agent's state comprises the following components:

- Door status: Indicates whether the door is open or closed.
- Key status: Indicates whether the agent is carrying the key.
- Agent position: Includes all valid positions on the map, except those occupied by walls, represented as (i, j), where i is the height index and j is the width index.
- Agent direction: Represented by the vectors $(0,1)$, $(1,0)$, $(0,-1)$, $(-1,0)$, corresponding to the directions down, right, up, and left respectively.
- Done status: Determines whether the agent has reached the goal, set to True if the goal is reached.

2) Control Space $\mathbb{U}$: The valid control inputs are:

- Move Forward,
- Turn Left,
- Turn Right,
- Pickup Key,
- Unlock Door,

represented numerically in the implementation as 0, 1, 2, 3, and 4, respectively.

3) Initial State Distribution p_0: The initial state is set at the beginning of each new map instance.

4) Motion Model p_f: The motion model is deterministic:

$$x_{t+1} = f(x_t, u_t)$$

The motion model for our autonomous agent is deterministic and governs how the agent interacts with its environment based on its actions. The state transition function $f(x_t, u_t)$ is defined as follows:

- **Move Forward (MF = 0):** The agent moves forward in the direction it is facing:

  $$\text{new\_agent\_pos} = \text{agent\_pos} + \text{agent\_dir}$$

  - If the new position is within the wall list, the agent remains in its current position.
  - If the new position hits a closed door and the agent does not have the key, it remains in its current position.
  - If the new position reaches the goal, the agent moves to the new position and the mission is marked as completed.
  - Otherwise, the agent successfully moves to the new position.
- **Turn Left (TL = 1):** The agent changes its direction counterclockwise:

  $$\text{agent\_dir} = (\text{agent\_dir}[1], -\text{agent\_dir}[0])$$
- **Turn Right (TR = 2):** The agent changes its direction clockwise:

  $$\text{agent\_dir} = (-\text{agent\_dir}[1], \text{agent\_dir}[0])$$
- **Pickup Key (PK = 3):** The agent picks up the key if it is directly in front of it:

  $$\text{agent\_front} = \text{agent\_pos} + \text{agent\_dir}$$

  If the position in front contains a key, the key status is set to true.
- **Unlock Door (UD = 4):** The agent attempts to unlock a door if it is directly in front of it:

  $$\text{agent\_front} = \text{agent\_pos} + \text{agent\_dir}$$

  - If the position in front contains a door and the agent has a key, the door status is set to open.
  - If the position in front contains a door and the agent does not have a key, the door status remains unchanged.

.

5) Planning Horizon T: The maximum planning horizon T corresponds to the number of states, but the process terminates once the goal is reached.

6) Stage Cost ℓ: The stage cost ℓ is consistently 1, encouraging the agent to reach the goal efficiently by minimizing time.

7) Terminal Cost q: The terminal cost q is set to 0 if the agent reaches the goal and ∞ otherwise, incentivizing goal completion.

8) Discount Factor γ: Given the finite planning horizon, the discount factor γ is set to 1, emphasizing the importance of the entire trajectory in decision-making.

B. Random Environment

For the Random Environment, we are tasked with computing a single control policy applicable to any of the 36 random 8×8 environments. The environment includes a vertical wall at column 4 with two doors at positions (4, 2) and (4, 5). Each door can be either open or locked, requiring a key to be unlocked. The key is randomly placed in one of three possible positions: {(1,1), (2,3), (1,6)}, and the goal is randomly located at one of the positions {(5,1), (6,3), (5,6)}. The agent is initially spawned at (3,5) facing upwards. The randomness in each scenario arises from the door statuses, key position, and goal position.

1) State Space $\mathbb{X}$: The state space comprises the following components:

- Key status: Indicates whether the agent is carrying the key.
- Agent position: Includes all valid positions on the map, except those occupied by walls, represented as (i, j) where i is the height index and j is the width index.
- Agent direction: Represented by the vectors {(0, 1), (1, 0), (0, -1), (-1, 0)}, corresponding to the directions down, right, up, and left, respectively.
- Key position: Can be in one of the three positions {(1,1), (2,3), (1,6)}.
- Door status: Each door may be open or locked.
- Goal position: Can be in one of the three positions {(5, 1), (6, 3), (5, 6)}.

2) Control Space $\mathbb{U}$: The control inputs are consistent with those in the known environments.

3) Initial State Distribution p_0: The environment initialization determines one of the 36 possible states based on the randomized positions of the key and the goal, along with the status of each door.

4) Motion Model p_f: The motion model for our agent is deterministic and describes how the agent's state changes in response to its actions within the environment:

- **Move Forward (MF = 0):** The agent attempts to move in the direction it is currently facing:

  $$\text{new\_agent\_pos} = \text{agent\_pos} + \text{agent\_dir}$$

  The new position is evaluated as follows:
  - If the new position collides with a wall, the agent remains in the current position.
  - If the new position is a door that is closed (not currently open), the agent remains stationary.
  - If the new position is out of environmental bounds, the agent remains in the current position.
  - Otherwise, the agent successfully moves to the new position.
- **Turn Left (TL = 1):** The agent changes its direction counterclockwise:

  $$\text{agent\_dir} = (\text{agent\_dir}[1], -\text{agent\_dir}[0])$$

This action updates the agent's direction to the left relative to its current orientation.

- **Turn Right (TR = 2):** The agent changes its direction clockwise:

$$\text{agent\_dir} = (-\text{agent\_dir}[1], \text{agent\_dir}[0])$$

This action updates the agent's direction to the right relative to its current orientation.

- **Pickup Key (PK = 3):** The agent attempts to pick up a key if it is directly in front of it:

$$\text{agent\_front} = \text{agent\_pos} + \text{agent\_dir}$$

If the position directly in front of the agent contains the key, the key status is updated to true.

- **Unlock Door (UD = 4):** The agent attempts to unlock a door if it is directly in front and the agent has a key:

$$\text{agent\_front} = \text{agent\_pos} + \text{agent\_dir}$$

If the position in front contains a door and the agent has the key, the door status is updated to open, corresponding to the specific door's index in the door list.

5) Plan Horizon T, Stage Cost ℓ, Terminal Cost q, and Discount Factor γ: These parameters are identical to those defined in the Known Environment setting, facilitating consistency across different scenarios.

III. TECHNICAL APPROACH

We address the optimal control problem using dynamic programming, an algorithm specifically designed to compute both the optimal value function $V_0^*(x_0)$ and the optimal policy π^*. This method efficiently computes the value function and policy by processing iteratively in a backward manner. It excels at tackling non-linear and non-convex challenges. The complexity of this approach is polynomial, dependent on the number of states $|\mathbb{X}|$ and the number of actions $|\mathbb{U}|$, making it significantly more efficient than brute-force methods that evaluate all possible policies.

We divide the application of this algorithm into two phases: computation and inference. The computation phase involves using dynamic programming to calculate the optimal control policy. The inference phase involves utilizing the pre-computed policy to interact with the environment. For known environments, we perform both computation and inference for each scenario; however, for unknown environments, we compute the policy once and conduct inference in 36 different settings.

A. Computing Optimal Policies

The **Dynamic Programming: Computing Optimal Policies** algorithm provides a structured approach to solving the optimal control problem in MDP through backward iteration. Initially, it sets the terminal conditions where the value at the goal state is zero and assigns an infinite value to all other states. The algorithm then proceeds to iteratively compute the cost-to-go $Q_t(x,u)$ for each state-action pair, incorporating both the immediate cost $\ell(x,u)$ and the discounted future costs from the successor states. By updating the value function $V_t(x)$ and the policy $\pi_t(x)$ at each timestep to reflect the minimum expected cost among all possible actions. This process is repeated for each timestep, progressively refining the policy and value estimates until the value function stabilizes, indicating that the optimal policy π^* and the optimal value function V^* have been robustly determined.

Algorithm 1 Dynamic Programming: Computing Optimal Policies

```
1:  Input: MDP $(\mathbb{X}, \mathbb{U}, p_0, p_f, T, \ell, q, \gamma)$
2:  Initialize $T = |\mathbb{X}| - 1$
3:  Set $V_T(goal) = V_{T-1}(goal) = \cdots = V_0(goal) = 0$
4:  Set $V_T(x) = \infty$ for all $x \in \mathbb{X} \setminus \{goal\}$
5:  for $t = T-1$ down to 0 do
6:      Compute $Q_t(x,u) = \ell(x,u) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x,u)}[V_{t+1}(x')]$ for all $x \in \mathbb{X} \setminus \{goal\}, u \in \mathbb{U}(x)$
7:      Update $V_t(x) = \min_{u \in \mathbb{U}(x)} Q_t(x,u)$ for all $x \in \mathbb{X} \setminus \{goal\}$
8:      Determine $\pi_t(x) = \arg\min_{u \in \mathbb{U}(x)} Q_t(x,u)$ for all $x \in \mathbb{X} \setminus \{goal\}$
9:  end for
10: if $V_t(x) = V_{t+1}(x)$ for all $x \in \mathbb{X} \setminus \{goal\}$ then
11:     break
12: end if
13: return $\pi^* = \pi_t$ and $V^* = V_t$
```

B. Inference

The **Inferencing Optimal Policies** algorithm effectively utilizes the pre-computed optimal policy π^* to generate an optimal sequence of actions from a given initial state. This process involves iteratively selecting actions based on π^* and updating the agent's state according to a predefined motion model. As a result, the algorithm constructs a tailored sequence of actions that efficiently guide the agent towards its intended goal, navigating through the environment's dynamics and constraints. The execution continues seamlessly until the agent successfully reaches a terminal state, ensuring optimal performance in real-time operational contexts. This methodology not only enhances the precision of task execution but also significantly improves the adaptability of robotic systems in complex and evolving environments.

Algorithm 2 Inference

```
1: Input: Optimal policy $\pi^*$
2: Initialization: Obtain the initial state of the agent, $x_0$.
3: Optimal Policy Sequence Execution:
4: while state is not done do
5:     Select the action $u_t$ using the optimal policy $\pi^*(x_t)$.
6:     Update the state using the motion model: $x_{t+1} = f(x_t, u_t)$.
7:     Append the action $u_t$ to the optimal policy sequence.
8: end while
9: Return the optimal policy sequence.
```

IV. Results

The application of the Dynamic Programming algorithm demonstrated robust performance in both known and unknown environments. This algorithm successfully navigated the complexities of these settings, generating effective strategies for efficiently achieving goals. In known environments, the algorithm handled challenges with ease. However, in random environments with large state spaces, the computation proved to be intensive and slow, taking up to five minutes to complete. While it still produced correct results, the efficiency was not optimal. Overall, the results validate the efficacy of dynamic programming in structured environments, yet they also highlight its limitations in more volatile and unpredictable settings.

A. Known Environments

The optimal policies for various known environments are summarized as follows:

- `doorkey-5x5-normal.env`: [TL, TL, PK, TR, UD, MF, MF, TR, MF]
- `doorkey-6x6-direct.env`: [MF, MF, TR, MF, MF]
- `doorkey-6x6-normal.env`: [TL, MF, PK, TL, MF, TL, MF, TR, UD, MF, MF, TR, MF]
- `doorkey-6x6-shortcut.env`: [PK, TL, TL, UD, MF, MF]
- `doorkey-8x8-direct.env`: [MF, TL, MF, MF, MF, TL, MF]
- `doorkey-8x8-normal.env`: [TR, MF, TL, MF, TR, MF, MF, MF, PK, TL, TL, MF, MF, MF, TR, UD, MF, MF, MF, TR, MF, MF, MF]
- `doorkey-8x8-shortcut.env`: [TR, MF, TR, PK, TL, UD, MF, MF]
- `example-8x8.env`: [TR, MF, PK, TL, UD, MF, MF, MF, MF, TR, MF]

B. Random Environments

The results from testing optimal policies in 36 different random environments are detailed below:

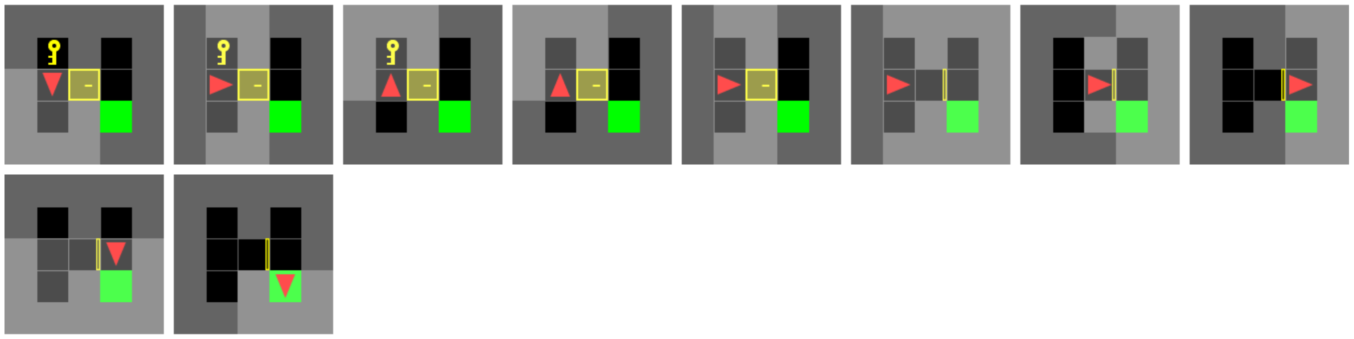
- Environment 1: [MF, MF, MF, TR, MF, MF, TL, MF]
- Environment 2: [MF, MF, MF, TR, MF, MF, TL, MF]
- Environment 3: [TR, MF, MF, TL, MF, MF, MF, MF]
- Environment 4: [MF, MF, MF, MF, TL, MF, PK, TL, MF, TL, MF, UD, MF, MF, TL, MF]
- Environment 5: [TR, MF, MF, MF, TL, MF, MF]
- Environment 6: [MF, MF, MF, TR, MF, MF, MF, TR, MF]
- Environment 7: [TR, MF, MF, MF, TL, MF, MF]
- Environment 8: [MF, MF, MF, MF, TL, MF, PK, TL, MF, TL, MF, UD, MF, MF, MF, TR, MF]
- Environment 9: [TR, MF, MF, TR, MF]
- Environment 10: [MF, MF, MF, TR, MF, MF, TR, MF, MF, MF, MF]
- Environment 11: [TR, MF, MF, TR, MF]
- Environment 12: [MF, MF, MF, MF, TL, MF, PK, TL, MF, MF, MF, MF, TL, MF, UD, MF, MF, TR, MF]
- Environment 13: [MF, MF, MF, TR, MF, MF, TL, MF]
- Environment 14: [MF, MF, MF, TR, MF, MF, TL, MF]
- Environment 15: [TR, MF, MF, TL, MF, MF, MF, MF]
- Environment 16: [MF, MF, TL, PK, TR, MF, TR, UD, MF, MF, TL, MF]
- Environment 17: [TR, MF, MF, MF, TL, MF, MF]
- Environment 18: [MF, MF, MF, TR, MF, MF, MF, TR, MF]
- Environment 19: [TR, MF, MF, MF, TL, MF, MF]
- Environment 20: [MF, MF, TL, PK, TR, MF, TR, UD, MF, MF, MF, TR, MF]
- Environment 21: [TR, MF, MF, TR, MF]
- Environment 22: [MF, MF, MF, TR, MF, MF, TR, MF, MF, MF, MF]
- Environment 23: [TR, MF, MF, TR, MF]
- Environment 24: [MF, MF, TL, PK, TL, MF, MF, TL, UD, MF, MF, TR, MF]
- Environment 25: [MF, MF, MF, TR, MF, MF, TL, MF]
- Environment 26: [MF, MF, MF, TR, MF, MF, TL, MF]
- Environment 27: [TR, MF, MF, TL, MF, MF, MF, MF]
- Environment 28: [TL, MF, MF, TL, PK, TL, MF, MF, UD, MF, MF, TL, MF, MF, MF, MF]
- Environment 29: [TR, MF, MF, MF, TL, MF, MF]
- Environment 30: [MF, MF, MF, TR, MF, MF, MF, TR, MF]
- Environment 31: [TR, MF, MF, MF, TL, MF, MF]
- Environment 32: [TL, MF, MF, TL, PK, TL, MF, MF, UD, MF, MF, MF, TL, MF, MF]
- Environment 33: [TR, MF, MF, TR, MF]
- Environment 34: [MF, MF, MF, TR, MF, MF, TR, MF, MF, MF, MF]
- Environment 35: [TR, MF, MF, TR, MF]
- Environment 36: [TL, MF, MF, TL, PK, TL, MF, MF, UD, MF, MF, TR, MF]

V. Conclusions

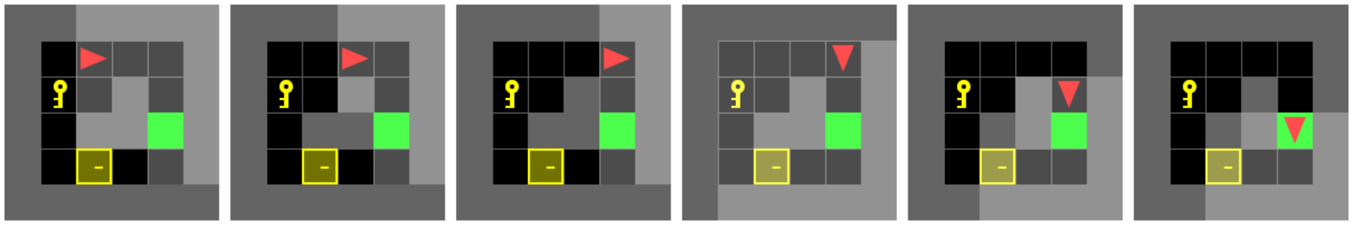
In conclusion, this study illustrates the effectiveness of dynamic programming algorithms in solving complex optimal control problems within autonomous navigation contexts, such as the Door & Key environment. By rigorously testing these algorithms in both known and randomly generated environments, we have demonstrated their ability to devise optimal navigation strategies that are both correct and efficient. The adaptability and scalability of our approach are particularly noteworthy, enabling robust performance across diverse and unpredictable settings.

Acknowledgment

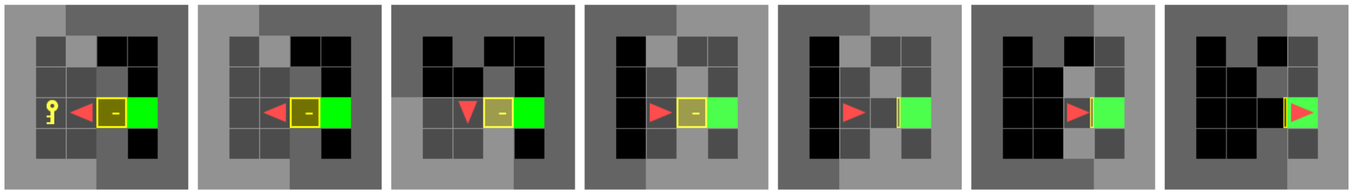
The author would also like to thank for the constructive advice from Professor Nikolay Atanasov, all teaching assistants, and everyone who shared their ideas on Piazza and during office hours.



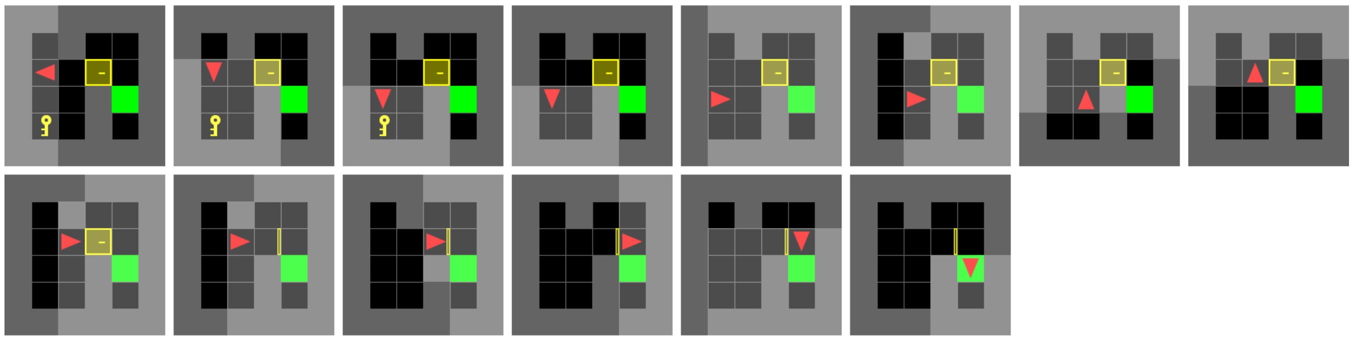
(a) doorkey-5x5-normal



(b) doorkey-6x6-direct

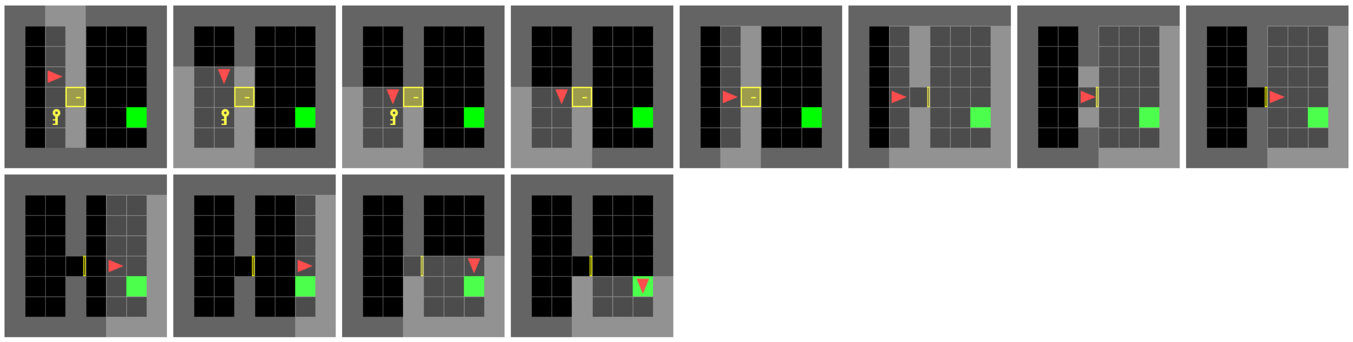


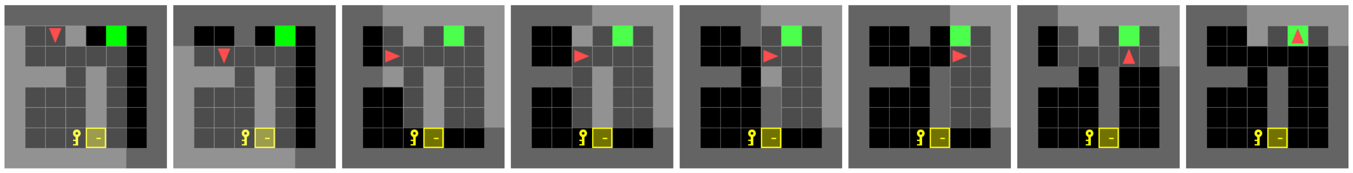
(c) doorkey-6x6-shortcut

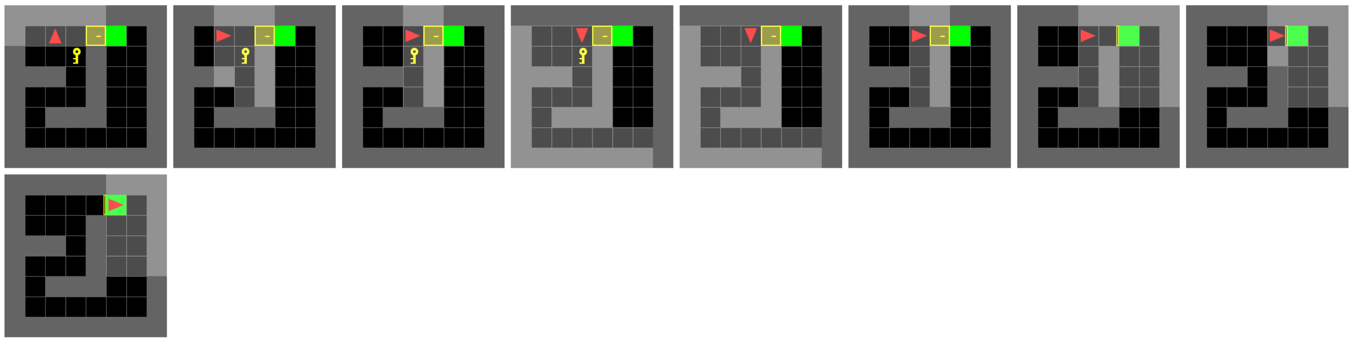


(d) doorkey-6x6-normal

Fig. 2: PartA: Known Envs, 5x5&6x6


(a) example-8x8


(b) doorkey-8x8-direct


(c) doorkey-8x8-shortcut

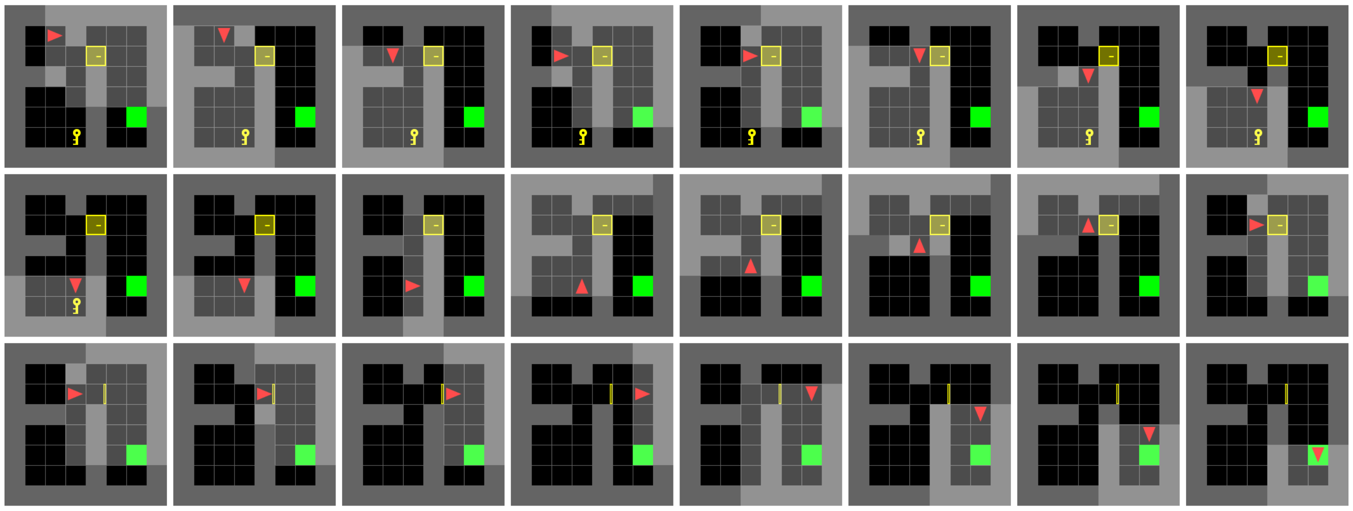
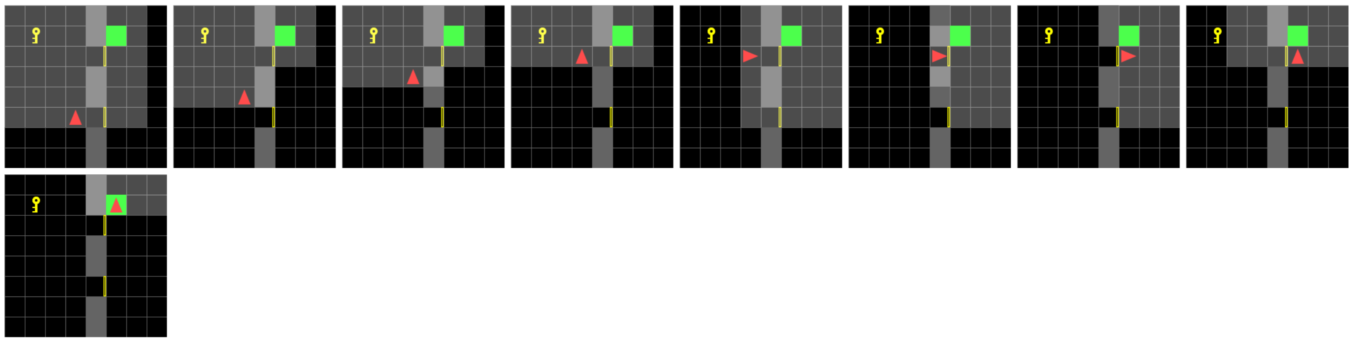
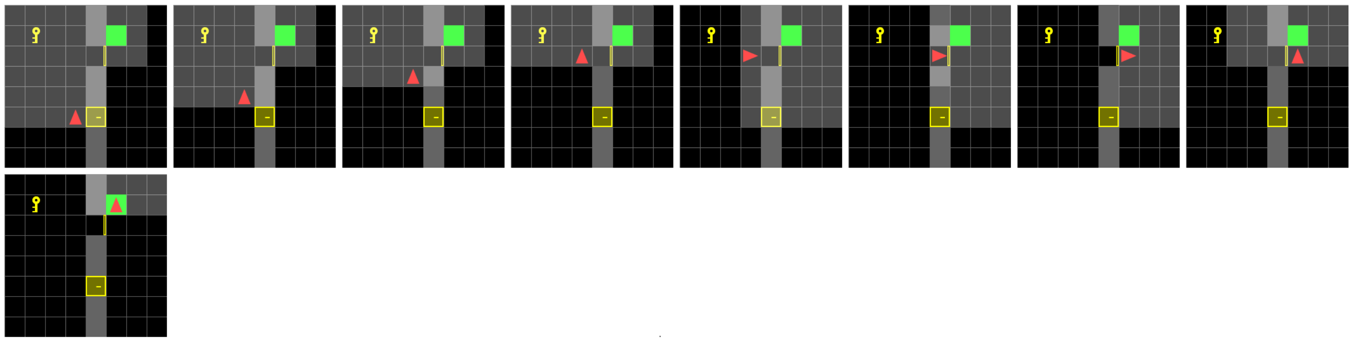
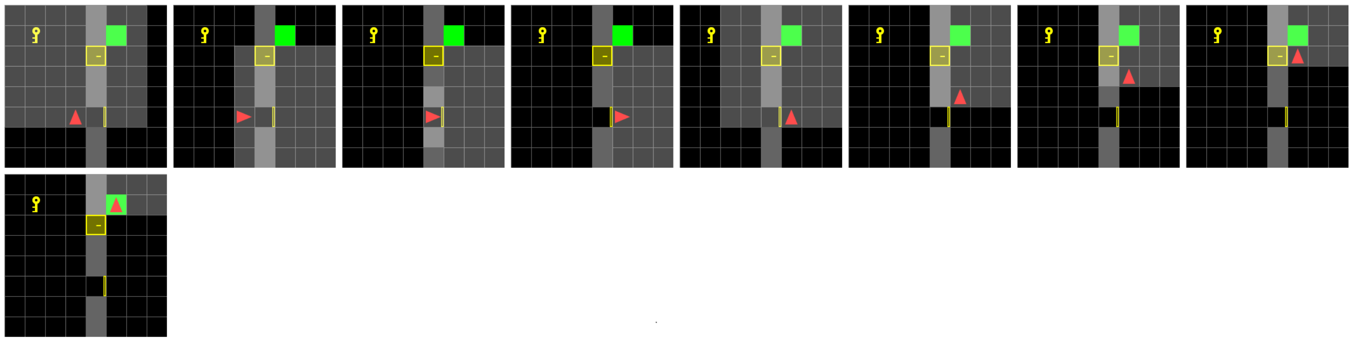

(d) doorkey-8x8-normal

Fig. 3: PartA: Known Envs, 8x8


(a) DoorKey-8x8-1


(b) DoorKey-8x8-2


(c) DoorKey-8x8-3

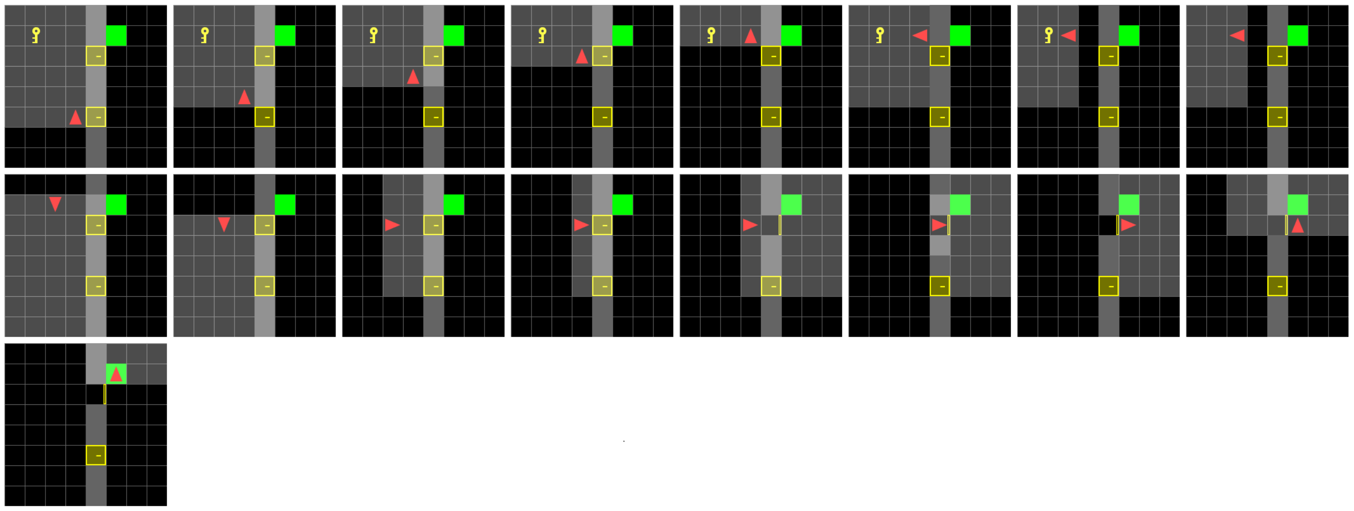
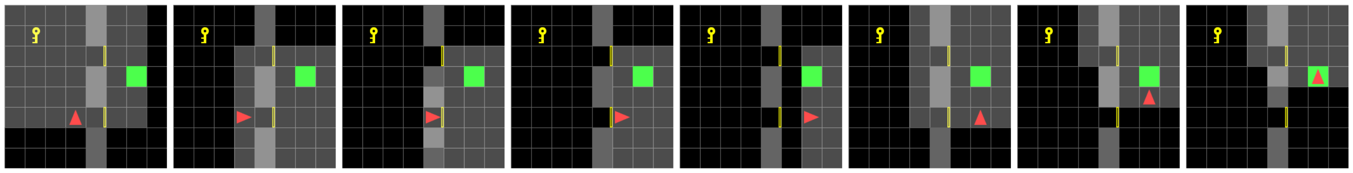
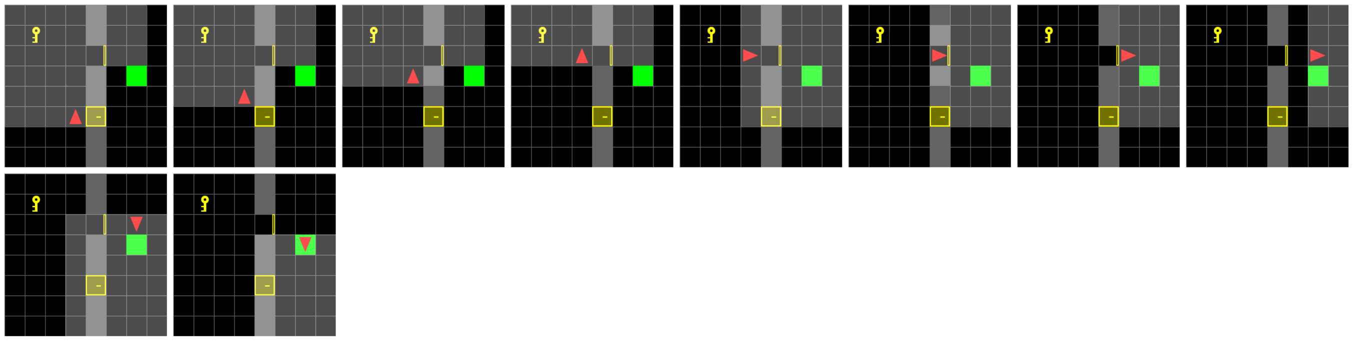

(d) DoorKey-8x8-4

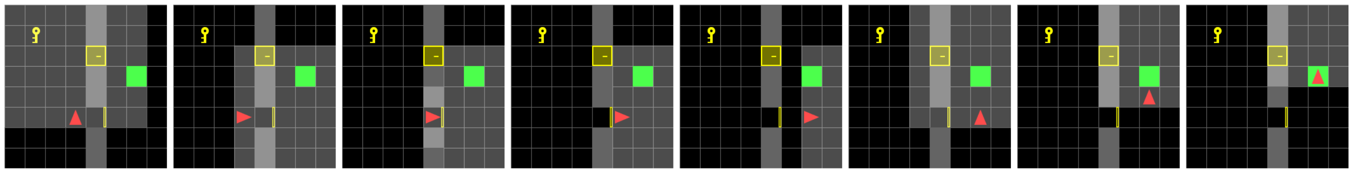
Fig. 4: PartB: Random Envs, 1-4



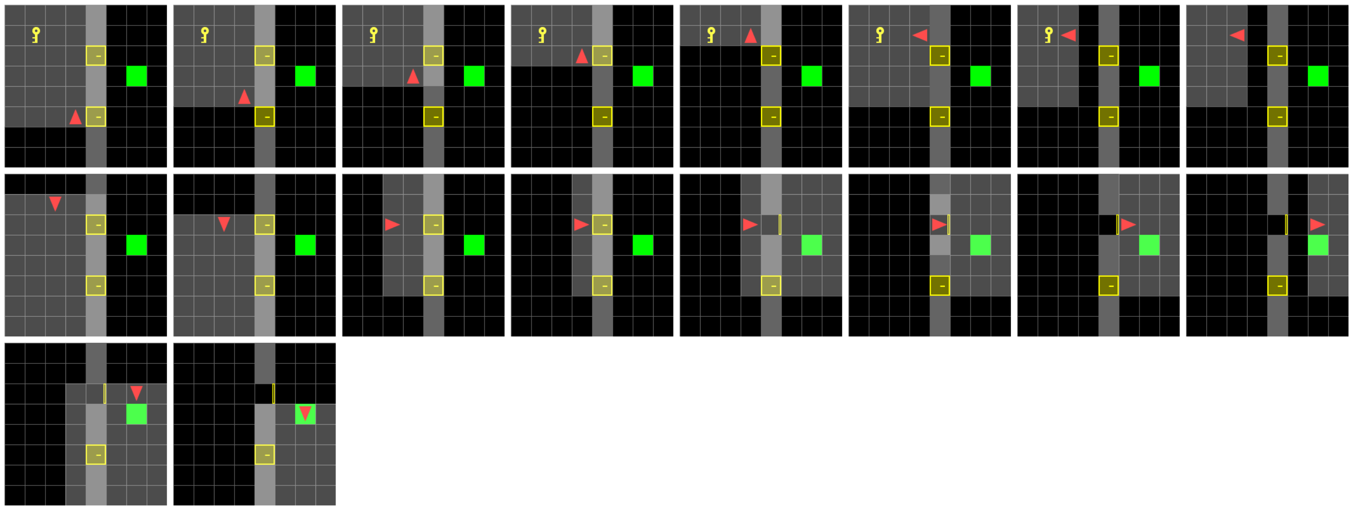
(a) DoorKey-8x8-5



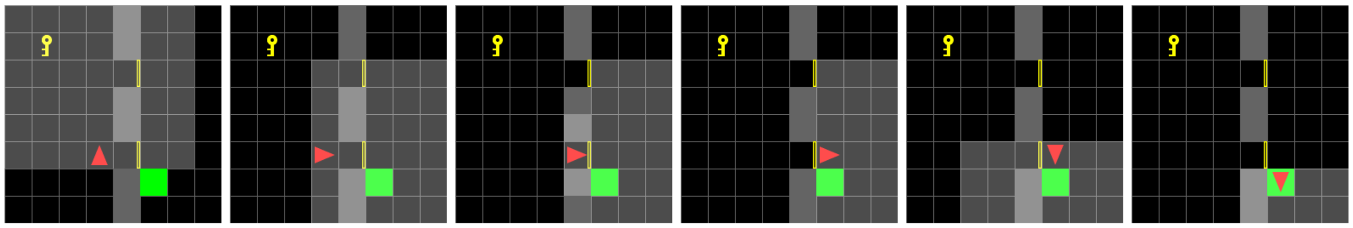
(b) DoorKey-8x8-6



(c) DoorKey-8x8-7

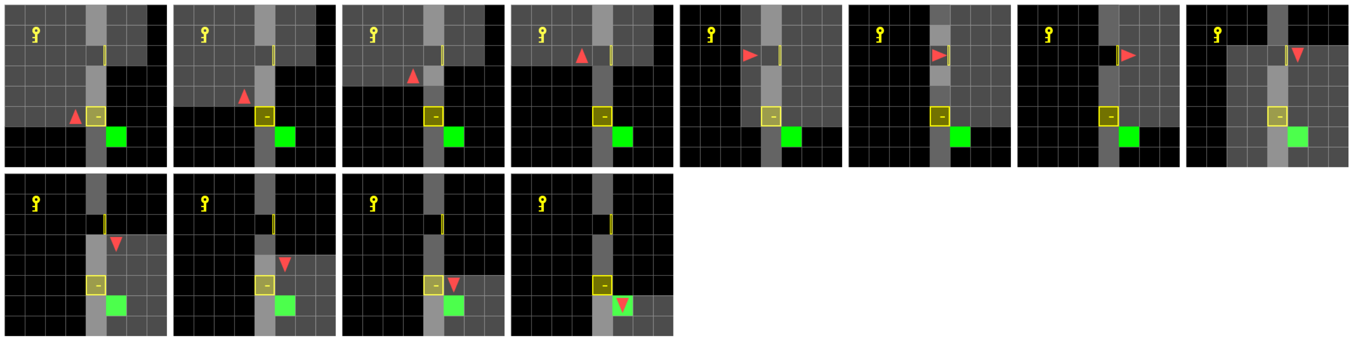


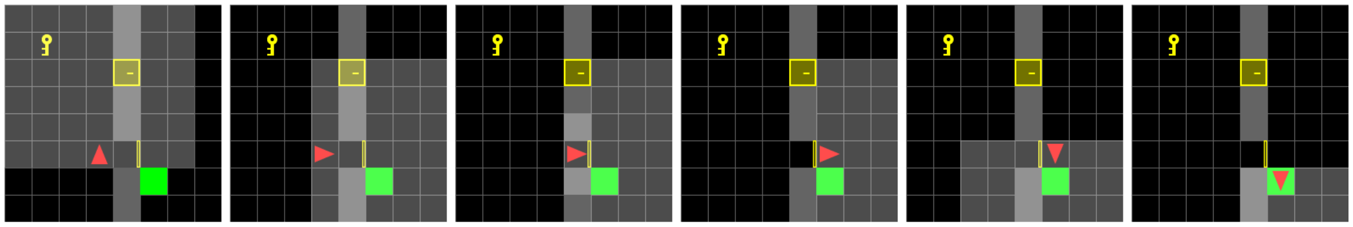
(d) DoorKey-8x8-8

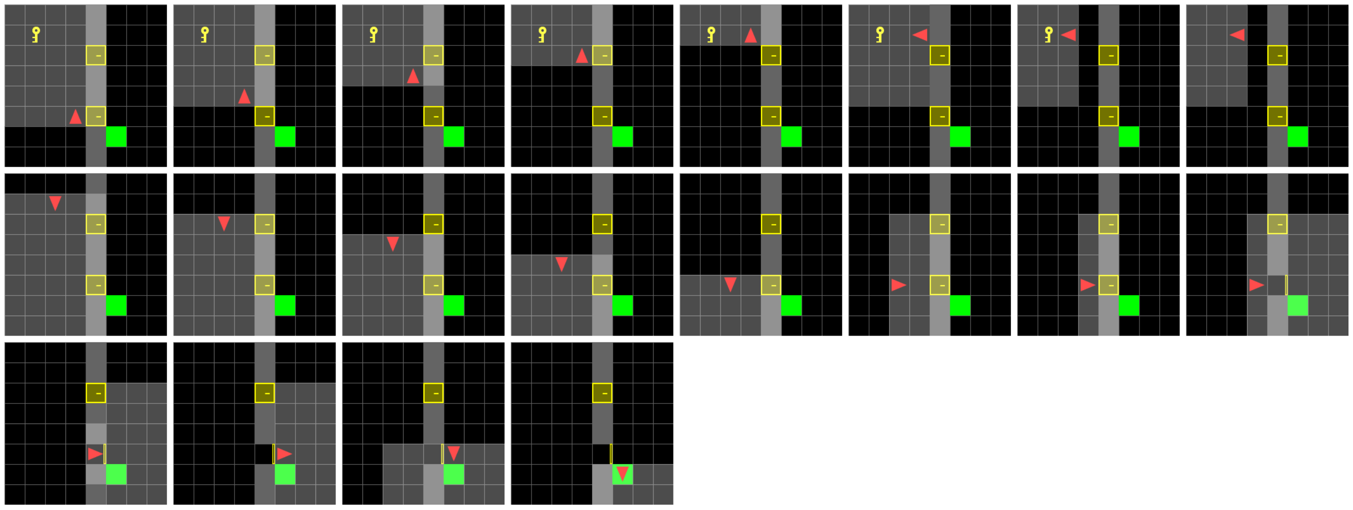


(e) DoorKey-8x8-9

Fig. 5: PartB: Random Envs, 5-9


(a) DoorKey-8x8-10


(b) DoorKey-8x8-11


(c) DoorKey-8x8-12

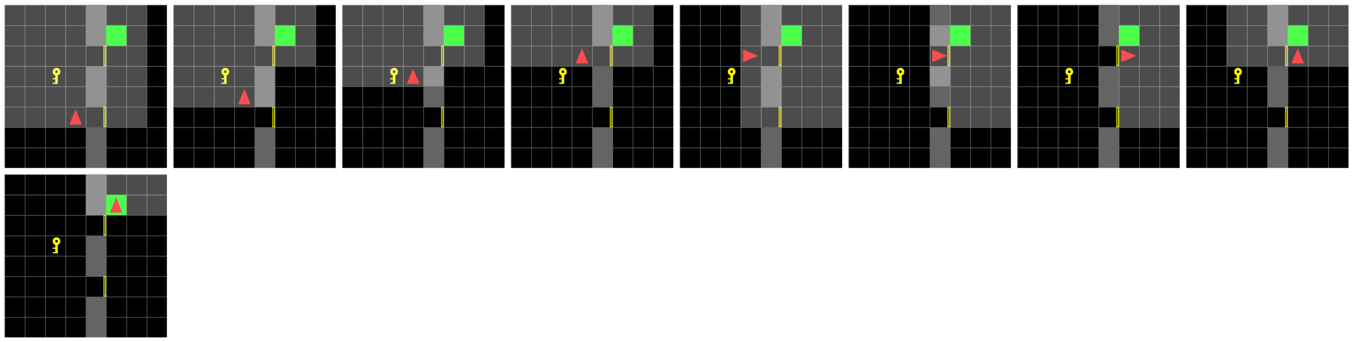
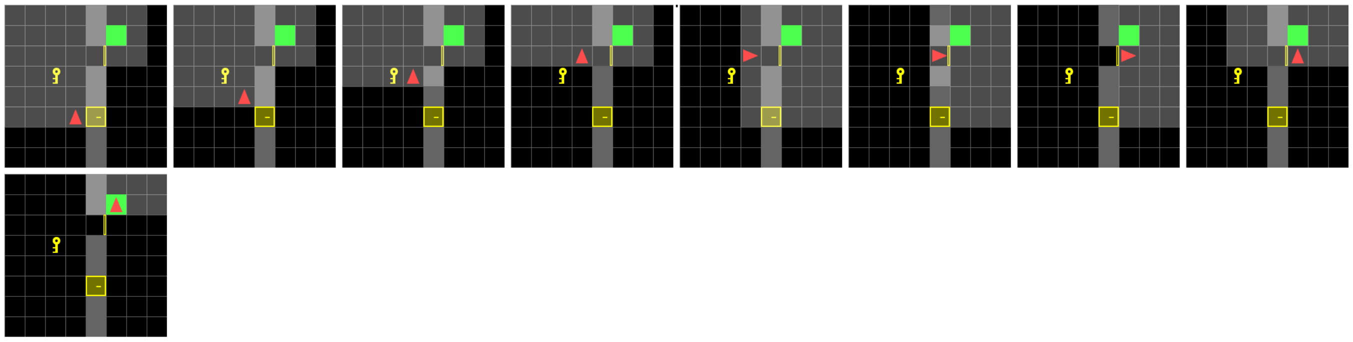
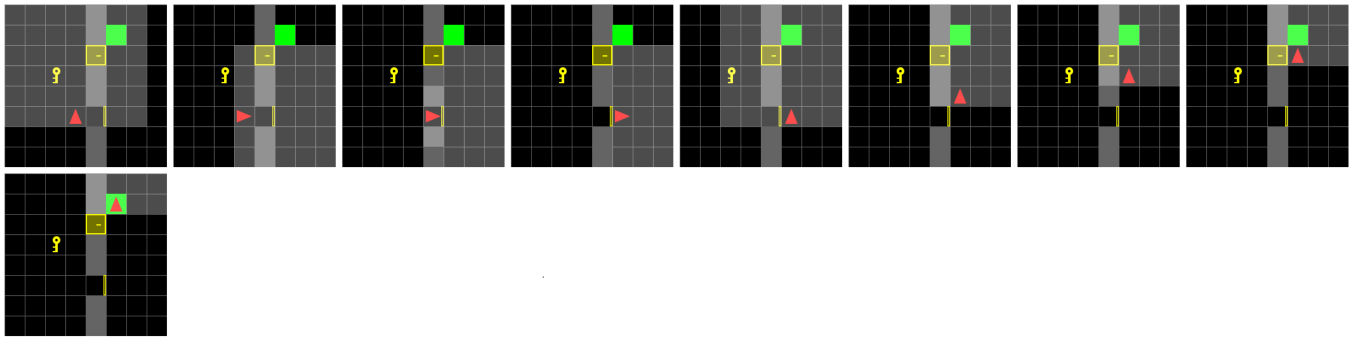

(d) DoorKey-8x8-13

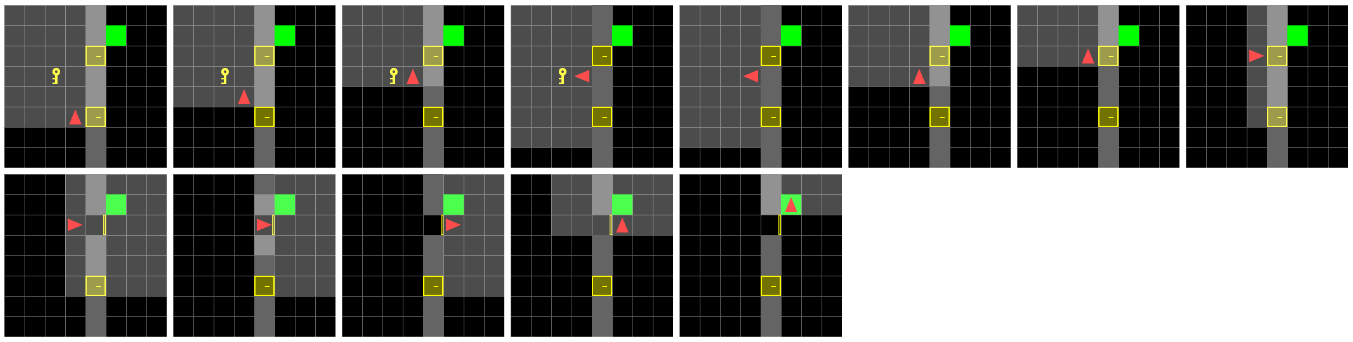
Fig. 6: PartB: Random Envs, 10-13



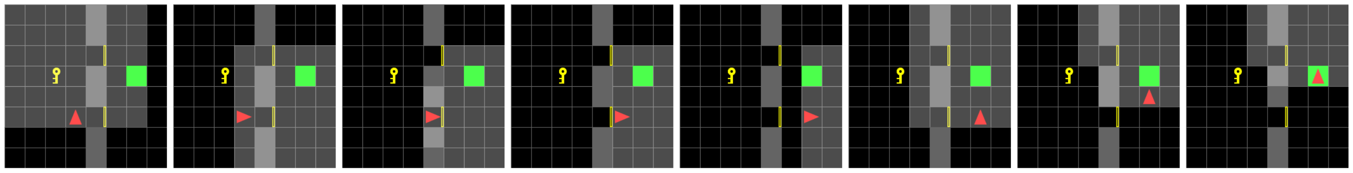
(a) DoorKey-8x8-14



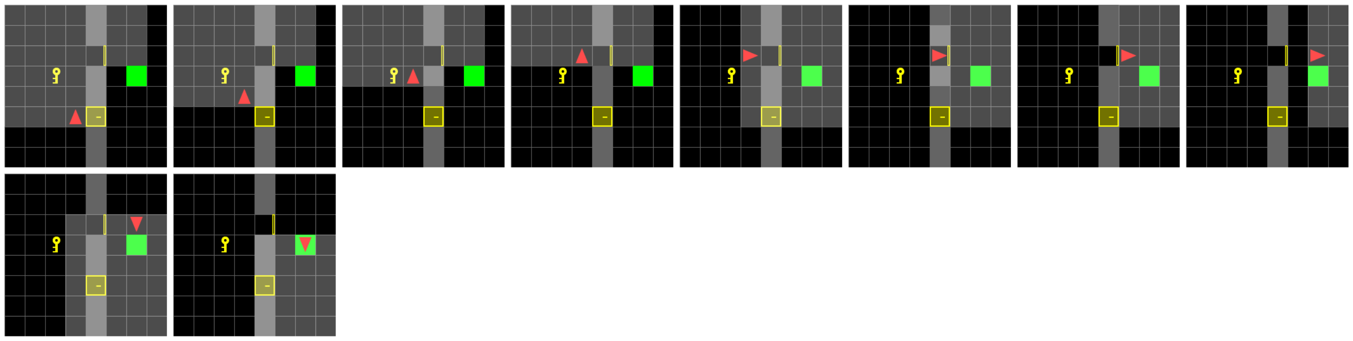
(b) DoorKey-8x8-15



(c) DoorKey-8x8-16

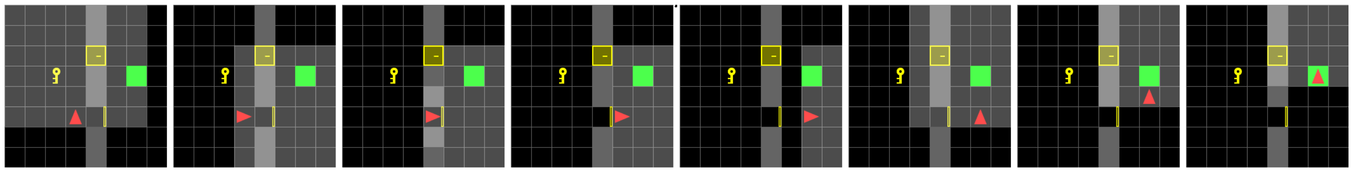


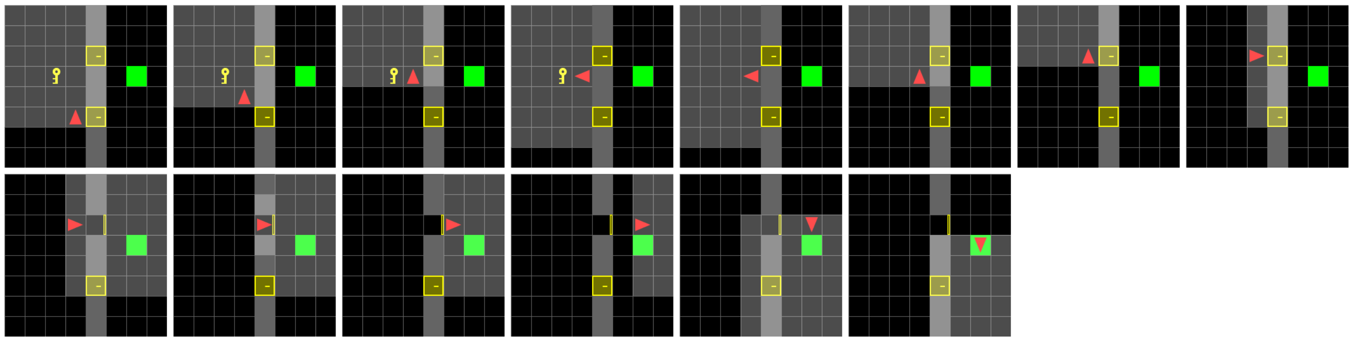
(d) DoorKey-8x8-17

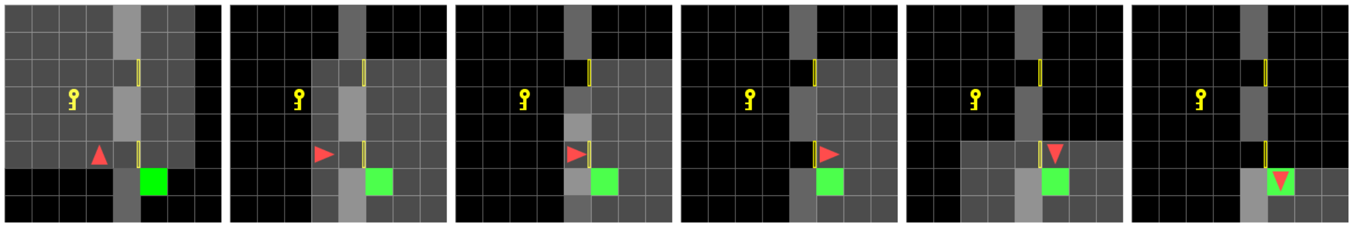


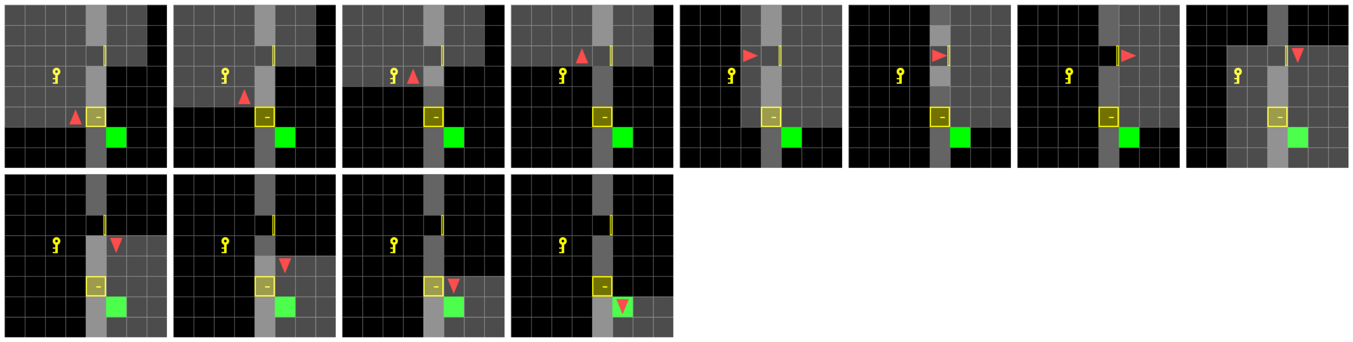
(e) DoorKey-8x8-18

Fig. 7: PartB: Random Envs, 14-18


(a) DoorKey-8x8-19


(b) DoorKey-8x8-20


(c) DoorKey-8x8-21


(d) DoorKey-8x8-22

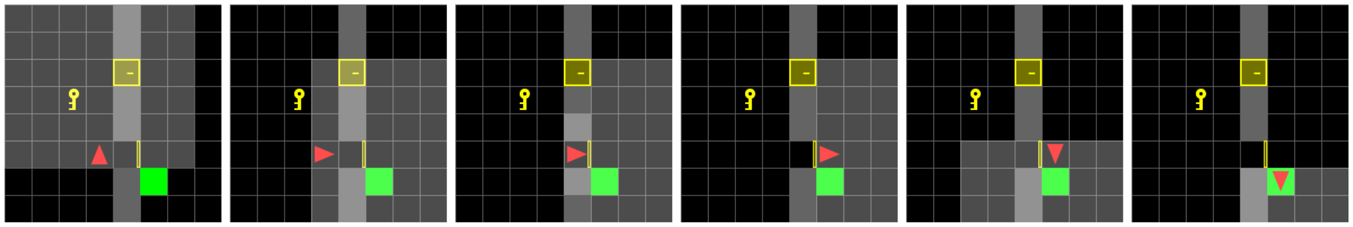
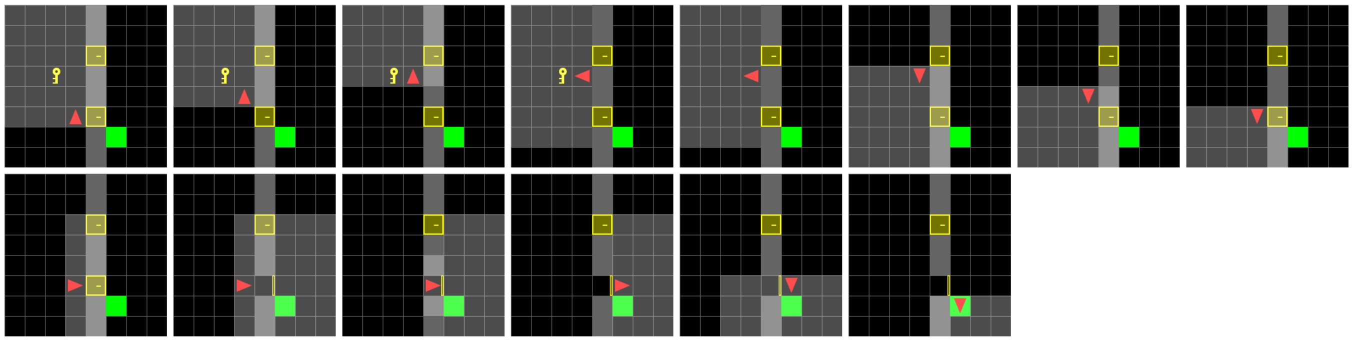
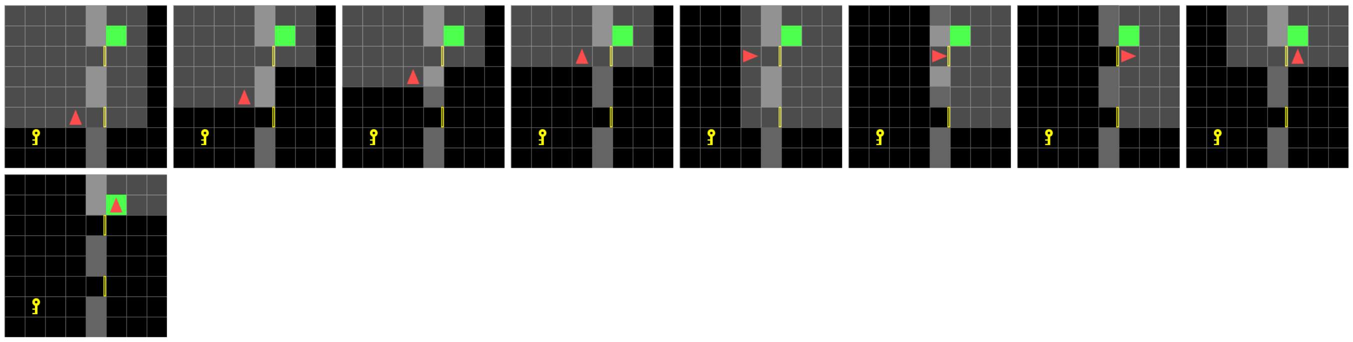

(e) DoorKey-8x8-23

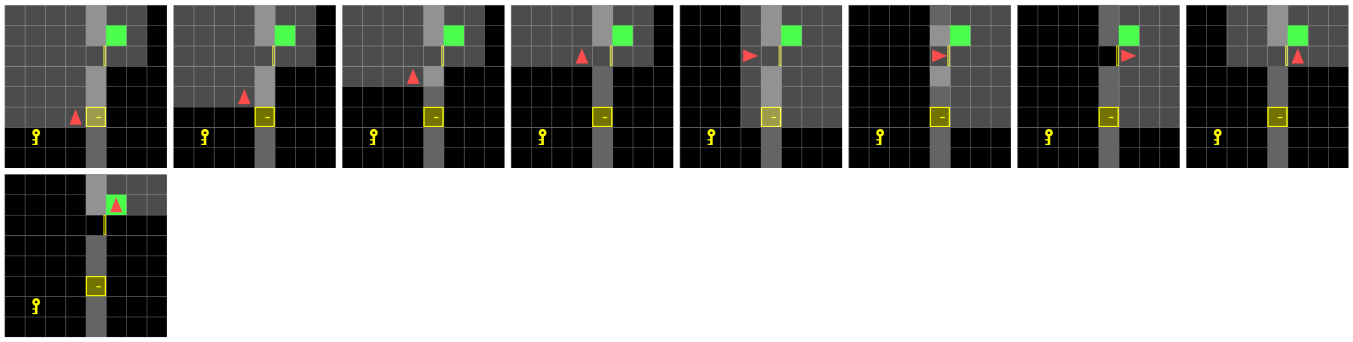
Fig. 8: PartB: Random Envs, 19-23



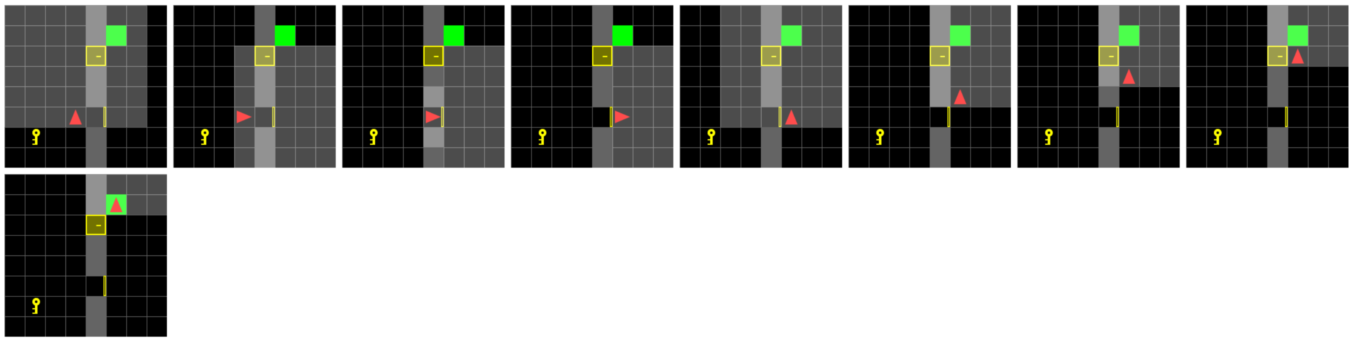
(a) DoorKey-8x8-24



(b) DoorKey-8x8-25

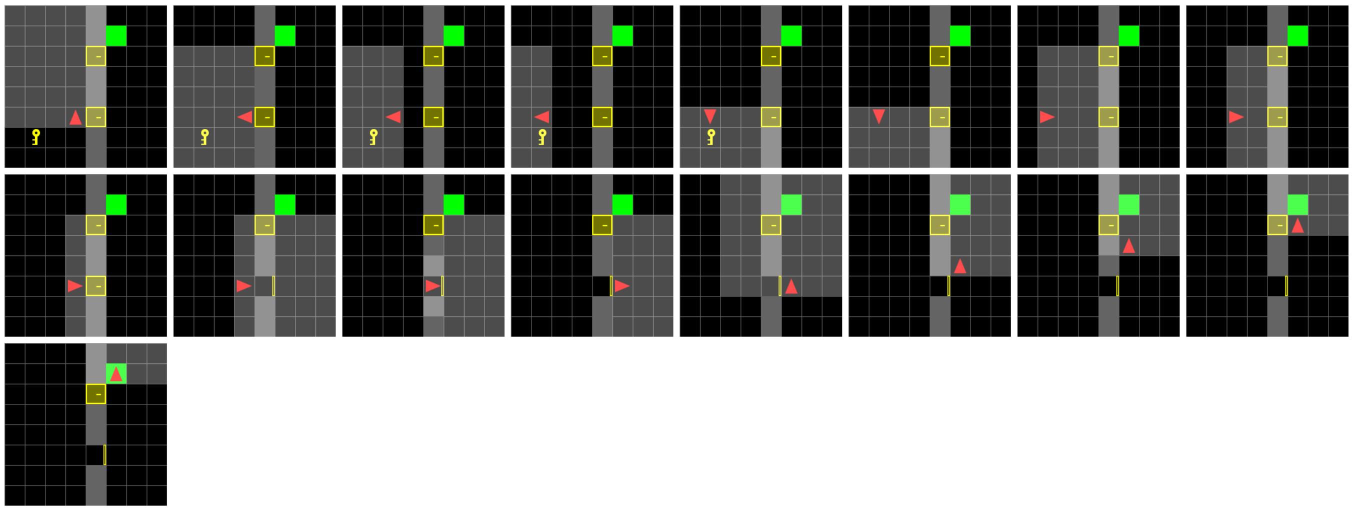


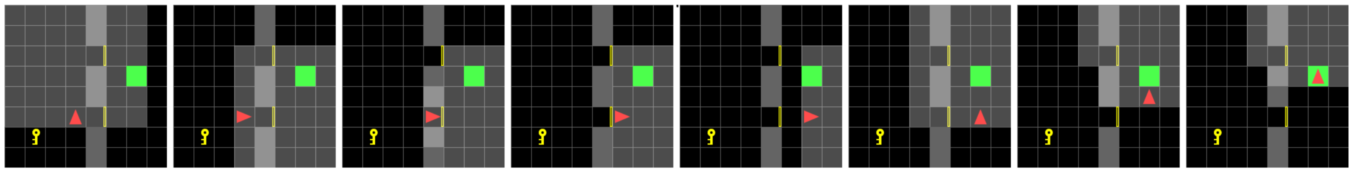
(c) DoorKey-8x8-26

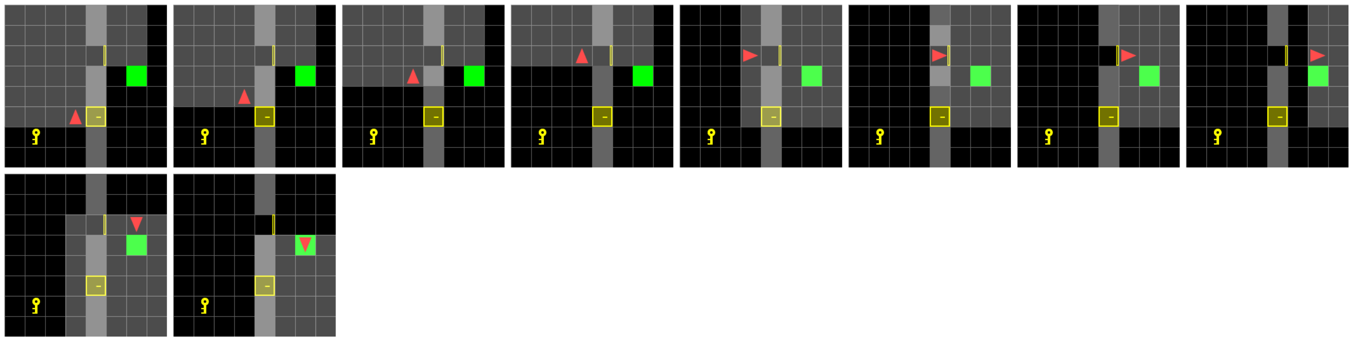


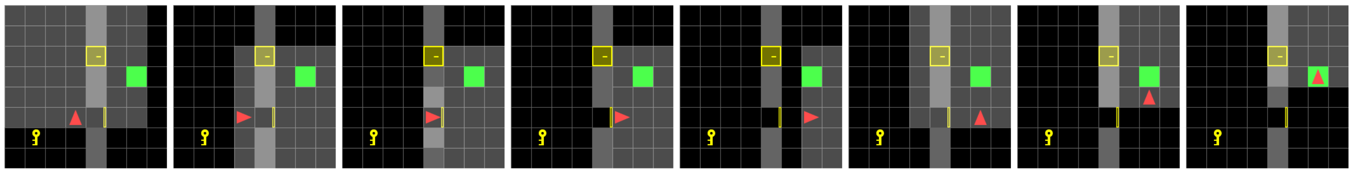
(d) DoorKey-8x8-27

Fig. 9: PartB: Random Envs, 24-27


(a) DoorKey-8x8-28


(b) DoorKey-8x8-29


(c) DoorKey-8x8-30


(d) DoorKey-8x8-31

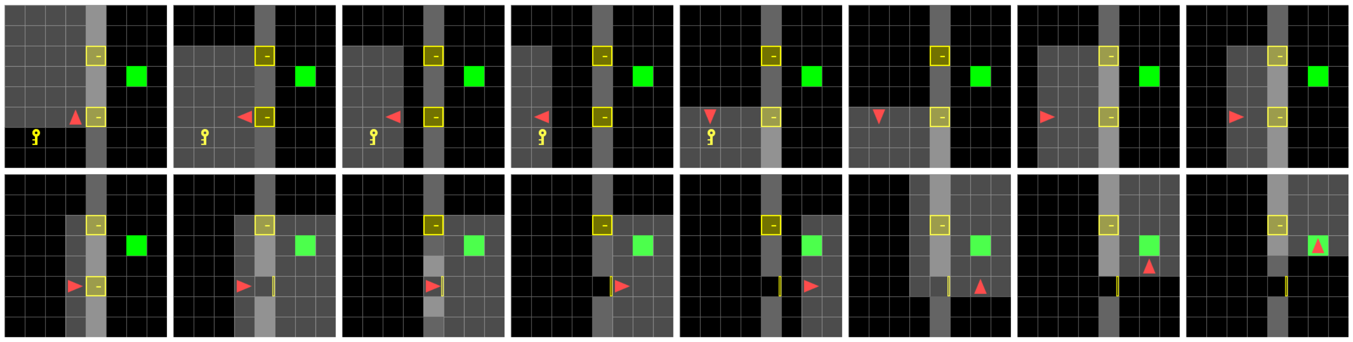
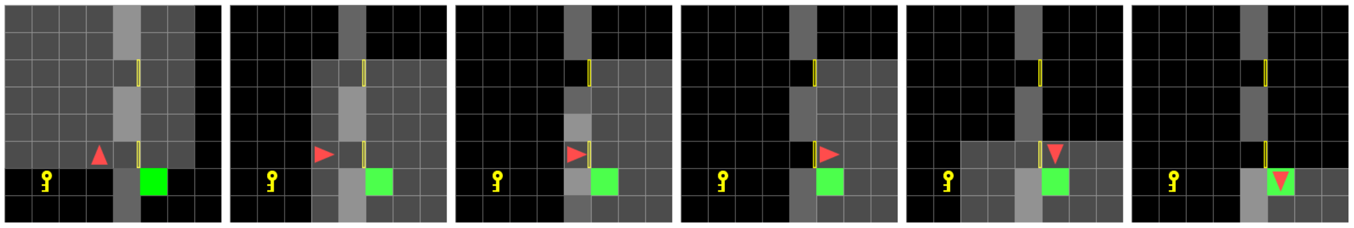
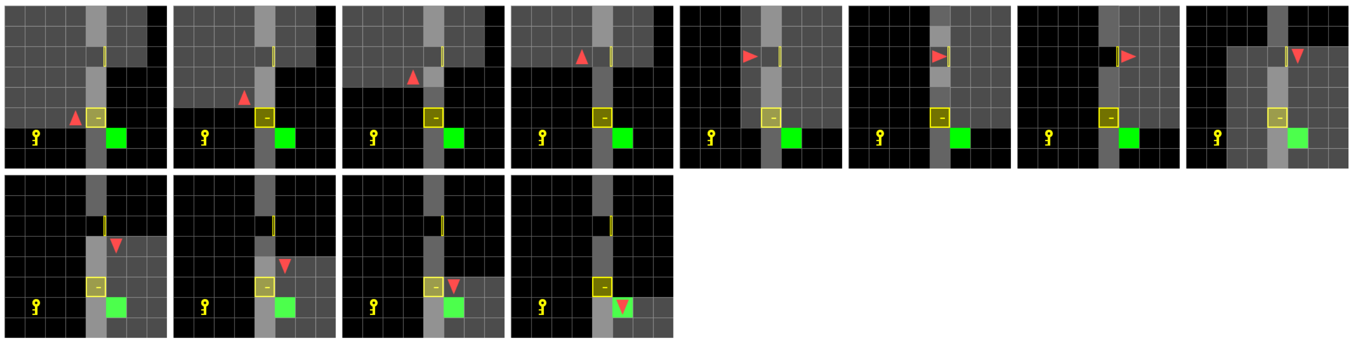
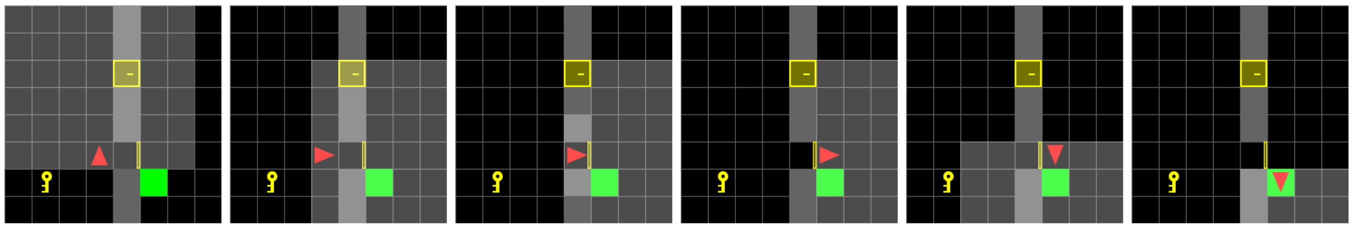

(e) DoorKey-8x8-32

Fig. 10: PartB: Random Envs, 28-32


(a) DoorKey-8x8-33


(b) DoorKey-8x8-34


(c) DoorKey-8x8-35

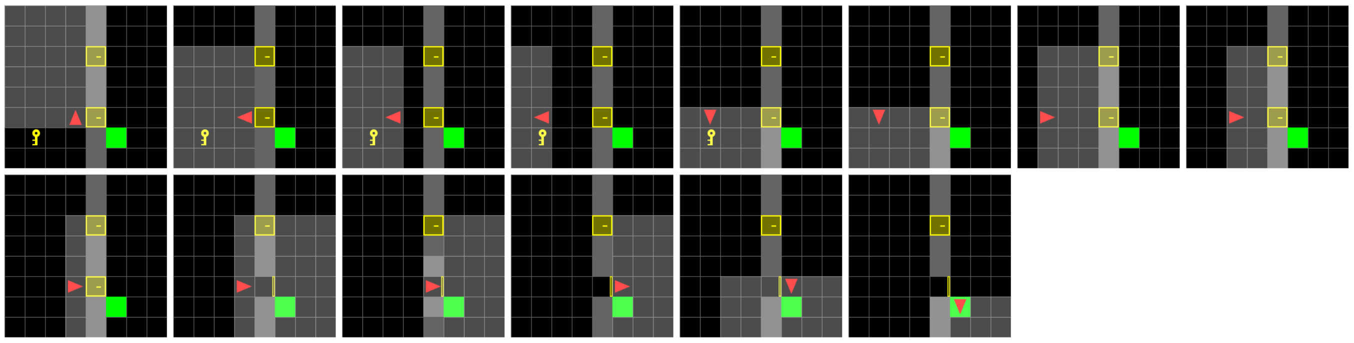

(d) DoorKey-8x8-36

Fig. 11: PartB: Random Envs, 33-36