Dynamic Programming Algorithm for Optimal Path Planning in a 2D Key and Door Problem

Kwok Hung Ho
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
khh019@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
natanasov@eng.ucsd.edu

I. Introduction

As the world drives towards autonomous and intelligent robots, the need for efficiency of robot actions is desired, as this can allow robots to be more fuel efficient and become more effective than humans in performing tasks. Efficient robotic movement is often tied to the field of path planning, which for example can involve finding the most optimal way to traverse a landscape or grasp an object with a robotic arm.

Despite the gravitation towards this field, real world systems for path planning are often complex and involve obstacles, obscure paths and dynamic variables. This paper presents a general approach to solving such optimal path problems by formulating the problem as a Markov Decision Process (MDP) and solving the optimal policy or path using the Dynamic Programming Algorithm (DPA). This paper then demonstrates the implementation in static discretized 2D map environments involving keys, doors, walls and goals, and computes optimal policies for individual known maps respectively and a single optimal policy which works for multiple configurations of known maps.

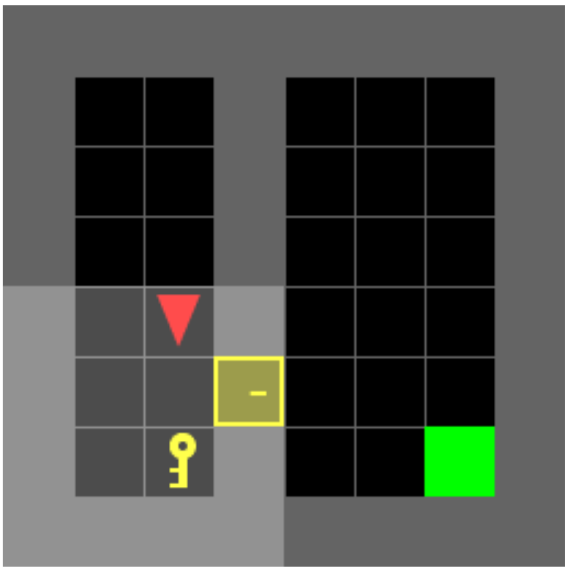

Fig. 1. Example of a map environment

II. Problem Formulation

A. Problem Formulation: Objective and Description

Understanding that there can be a myriad of approaches to solve the problem, only the input, output, and associated models that cannot be changed and are inherent to the problem will be presented here. The technical approach, where solutions to the problems that will be described, will be presented in the next section.

The 2D map environments that this paper will demonstrate DPA on are discretized square grid maps that can contain several objects in them. Grids can be classified as $Walls, Doors, Keys, and None$. There is also the notion of the $Goal$ which can only be in one grid cell and the $Agent$, which can also only be in one grid cell at each time step but can also face four directions: $Up, Down, Left, Right$. Since the map is static, the only thing that moves is the agent, this however does not mean that objects such as keys and doors cannot be picked up or opened respectively, causing changes in the environment. Static is referring to the movement in between grid cells.

Due to the nature of discretized time, and grid cells inherent to the problem as described above, modeling the problem as a Markov Decision Process is possible, allowing multiple methods to be used including DPA. A feature of the MDP model is the Markov Assumption. Derived from Bayesian conditional probability, this allows states to be conditionally independent from states that are not its parent.

B. Problem Formulation: MDP Formulation

The MDP also contains components including initial state x_0, goal state τ state space $\mathcal{X}$, control space $\mathcal{V}$, prior probability distribution $\mathcal{P}_0$, motion model $\mathcal{P}_f$, t horizon T, stage cost ℓ, terminal cost q, and discount factor γ. Due to the deterministic nature of the problem, prior probability will be ignored and so will the discount factor leading to the problem being being classified as a Deterministic Shortest Path (DSP) problem. Due to finite states of the problem, it is also modeled as a Deterministic Finite State (DFS) Optimal Control Problem.

The MDP DSP formulation will be described by the following variables.

$$\mathcal{X}, x_0, \tau, , \mathcal{U}, \mathcal{P}_f, T, \ell, q \tag{1}$$

$\mathcal{X}$ is the state space and for part a, it contains the co-ordinates of the agent, the orientation of the agent, the key status and the door status. Where co-ordinates are 2D indices (column, row), orientation is one of four directions (up, down, left, right), key status is a Boolean indicating whether the key has been picked up, and door status is a Boolean indicating whether the door is locked. (locked and closed are grouped, while unlocked and open are grouped). Important to note is that all walls can be excluded from the state space. As such, for example, a 8x8 grid with no walls will contain $8x8x4x2x2 = 1024$ states. The details of the representation will be examined in the technical approach where the motion model $\mathcal{P}_f$ will play a key role.

For part b, due to having 3 possible locations of the goal, 3 possible locations of the key, and 4 configurations of the door status due to 2 doors, the state space will be expanded to include agent position, agent orientation, key location, goal location, key status and door lock configuration. For a 8 by 8 grid with no walls, this will mean $8x8x3x3x2x4 = 4608$ states.

x_0 is the initial state or states. Although generally not important for algorithms which compute the cost from the goal to all states, the initial state lies in the state space and can be user-defined to check the optimal policy once obtained from a path finding algorithm. In the one-to-one maps to policy case (part a), there will only be one initial state where as the 36 possible maps case (part b), the policy will have 36 initial states.

τ represents the goal state, which due to a sufficient condition can take on multiple states in the state space.

$\mathcal{U}$ is the control input space which is given as follows:

$$\{MF, TL, TR, PK, UD\} \tag{2}$$

Where MF represents move forward, TL represents turn left, TR represents turn right, PK represents pick up key, and UD represents unlock door.

$\mathcal{P}_f$ represents the motion model which takes in a state and control to output a new state. The motion model also describes the motion of the problem, illustrating possible transitions between states.

$$\mathcal{P}_f(x_t, u) = x_{t+1} \tag{3}$$

T is the time or planning horizon, which for the problem is finite as there is a finite number of states.

ℓ is the stage cost indicating the cost to transition from one state to another.

q is the terminal cost indicating the cost for reaching the goal. For DSP problems, it is common to set it as 0.

C. Problem Formulation: Conclusion

With the problem formulated, the technical approach which describes how to solve the problem will be presented. Modifications and implementations of the formulation above to achieve the goal will be presented. The realization of optimal path planning for part a (one-to-one) policy to map, (one-to-many) policy to map will be shown in detail.

III. Technical Approach

A. Technical Approach: Introduction

Although there could be other ways to solve the problem formulated in the above section, the section: technical approach, will attempt to solve the problem formulated in the previous section via the Dynamic Programming Algorithm (DPA), which is the centerpiece of this paper.

B. Technical Approach: Motion Model

Integral to the problem at hand is defining the motion model $mathcalP_f$ as described in the problem formulation. Without a precise and accurate motion model, the state transitions will be incorrect, leading to inaccurate policies, or policies that stay in a single grid cell forever.

Due to having five control inputs, $\{MF, TL, TR, PK, UD\}$, $mathcalP_f(x, u)$ will be implemented as a switch case of 5 values. Using common sense, the possible transitions are deduced.

Algorithm 1 Motion Model for part a (One-to-One map to policy)

Input: vertices V, start $s \in V$, goal $\tau \in V$, and costs cij for $i, j \in V$

```
if u = MF then
    if In front is a wall then
        Return x
    end if
    if In front is a door and door is locked then
        Return x
    end if
    Return x(Move Forward)
end if
if u = TL then
    Return x(Turn Left)
end if
if u = TR then
    Return x(Turn Right)
end if
if u = PK then
    if In front is a key then
        Return x(have key=True)
    end if
    Return x
end if
if u = UD then
    if In front is a door and have key then
        Return x(door locked = False)
    end if
    Return x
end if
```

In general, the motion model is very similar in part a and part b despite different dimensions of the state space. This is because the state is passed into the motion model so despite having the possibility of three goals/keys, there is only 1 goal and key for each state. The only slight differences are due to having 2 door locations and the locking configuration.

C. Technical Approach: Dynamic Programming Applied to DSP

Due to certainty of the movements and the problem being formulated as a deterministic shortest path (DSP) problem, DPA can be applied to DSP, which has the algorithm below.

Algorithm 2 Dynamic Programming Algorithm Applied to DSP

Input: vertices V, start $s \in V$, goal $\tau \in V$, and costs cij for $i, j \in V$

1: $T = |V| - 1$ ▷ Planning Horizon
2: $V_T(\tau) = V_{T-1}(\tau) = ... = V_0(\tau) = 0$
3: $V_T(i) = \infty, \forall i \in V \neq \tau$
4: $V_{T-1}(i) = c_{i,\tau}, \forall i \in V \neq \tau$
5: $\pi_{T-1}(i) = \tau, \forall i \in V \neq \tau$
6: **for** $t = (T-2), \ldots, 0$ **do**
7: $\quad Q_t(i,j) = c_{i,j} + V_{t+1}(j), \forall i \in V \neq \tau, j \in V$
8: $\quad V_t(i) = \min_{j \in V} Q_t(i,j), \forall i \in V\ \tau$
9: $\quad \pi_t(i) = \arg\min_{j \in V} Q_t(i,j), \forall i \in V\ \tau$
10: $\quad$ **if** $V_t(i) = V_{t+1}(i), \forall i \in V \neq \tau$ **then** ▷ Early Stop
11: $\quad\quad$ **Break**
12: $\quad$ **end if**
13: **end for**

Where the input V for vertices is synonymous to states in this problem, $c_{i,j}$ is the stage cost of transitioniong from state i to state j, Q is a function that is used to compute optimality, and the V function is the optimal value function, indicating the most optimal value for a paticular state. To be more specific, the way j is obtained given i is through the motion model and control u.

The algorithm above will explore all paths from the goal to each state and compute the minimal cost policy π and value V. Once the algorithm terminates via early stopping or by the finite time horizon time, π will contain the optimal policies for each time step and each state. To check whether the policy is optimal, one can use any initial position, get u from policy pi at a time step, input x and u into the motion model to get a new x, and repeat. The policy should always bring you to the goal in the most optimal way due to minimizing the cost of the agent.

To be more specific, line 8 and line 9 in (Algorithm 2) are the reason the optimal path is taken. By taking the min and argmin of the Q function, the initialization steps done in lines $1-5$ which set the terminal cost to 0 and 1 step before the goals policy to move forward, allow an iterative process starting from the goal to all states in an optimal way.

A parameter choice that was taken is to make all stage costs $c_{i,j}$ or $l(x,u)$ simply unit 1. This is a feasible and reasonable choice because the algorithm would work for this simple choice. Due to no action having substantial risk out of $\{MF, TL, TR, PK, UD\}$, the cost actions can be equal. The reason uniform cost would work well is because the map is static, there are no random variables or necessary idle time, and there is always something the agent could be doing at each time step to shorten the journey to the goal. As such, the algorithm would always prefer the optimal choice over wasting an action or remaining in the same state.

D. Technical Approach: Code Implementation

For the implementation of the algorithms illustrated in the pseudo-code above (Algorithm 1) and (Algorithm 2), Python will be used and the data structures herefor mentioned will be as ordained.

Firstly, the state space $\mathcal{X}$ will be represented as a list of tuples. This is same for part a and b except part b will have some extra states as described in the MDP formulation. Next, control space $\mathcal{U}$ will trivially be a list of numbers from 0 to 4 indicating the 5 policies. By running a fixed amount of recursive for loops, all states will be contained within X. Due to the nature of the problem being a known map, all walls can be prior added to a hashset or simply set data structure in Python, to omit such states from the state space.

More importantly, are the data structures used to store the Q function, Value function V and policies π. An efficient data structure that allows instant retrieval of a state's parameters and easy referencing is a hash-map or dictionary in Python. Due to individual states being tuples, a hashable type, dictionaries are a good choice. In implementation, V and π are both lists of dictionaries, with the list indexing time t of the time horizon T and the dictionary indexing the state with value for V and policy for π. Q is implemented as a list of dictionaries of lists, where the outer list indexes time, the dictionary hashes state i, and the inner list will always be size 5. The inner list is also temporarily cached as a NumPy array, allowing quick computation of min and argmin.

The motion model $mathcalP_f$ in (Algorithm 1) does not use any fancy algorithms except for the set as mentioned for efficient checking of whether a wall is in front of the agent.

IV. Results

A. Results: Analysis and Images

The results are overall positive and everything mentioned worked as expected. Tests for the separate code for part a and part b were run and the optimal paths were double checked to indicate that the algorithm was successful. GIFS for both cases and are provided along with the code implementation on GitHub.

B. Results: Discussion of Performance and Implementation

The results indicate great performance of the algorithm in terms of finding the optimal path. In terms of complexity of part a and b, part a was pretty much instant and did not even need a stopping criteria while part b took a much longer time due to the much larger state space. Additionally, I faced some

difficulties with the early stopping criteria for part b. Although part a was solved quickly, the large state space of part b made it so that debugging part b was very painstaking, as each run would take about 20 minutes or more, and the stopping criteria would never run. Due to the long time part b took to run, using Python's Pickling function to save variables that took a long time to compute was very helpful for debugging. After computation of the optimal policy π, it was pickled so it can be used for testing and the resulting optimal path starting at the initial state and iterating through the motion model while saving the policy was also pickled. Other than difficulties faced about early stopping, small bugs in the motion model for part b also caused significant time wasted. One error was setting the door to True after unlocking the door, when it should've been set to False. Since the variable is named "door locked", it should be set to False whenever the agent unlocks it with the key.

C. Results: Conclusion

Overall, the implementation of the DPA algorithm was a success. For future improvements, spending more time on fixing the stopping criteria would've saved more time. For speed of convergence and search, using other algorithms to solve the problem could've yielded better results. An example could be A-star search, where a heuristic function such as euclidean distance could've yielded faster search times, especially for part b. Another great approach that would've probably yielded great results is Jump Point Search because the environment is enclosed allowing its main bottleneck of searching along one direction to be mitigated while allowing faster search speed by having less nodes in the open set. Nonetheless DPA is a great baseline algorithm and is quite effective in DSP problems of finite state space.

Following are optimal sequences from the GIFs generated from part a and b. They can be read row to row. All 7 envs provided from part a are represented with pngs automatically generated from convert.io. For part b, due to having 36 inital states. 2 optimal path pngs generated from GIFS are provided.

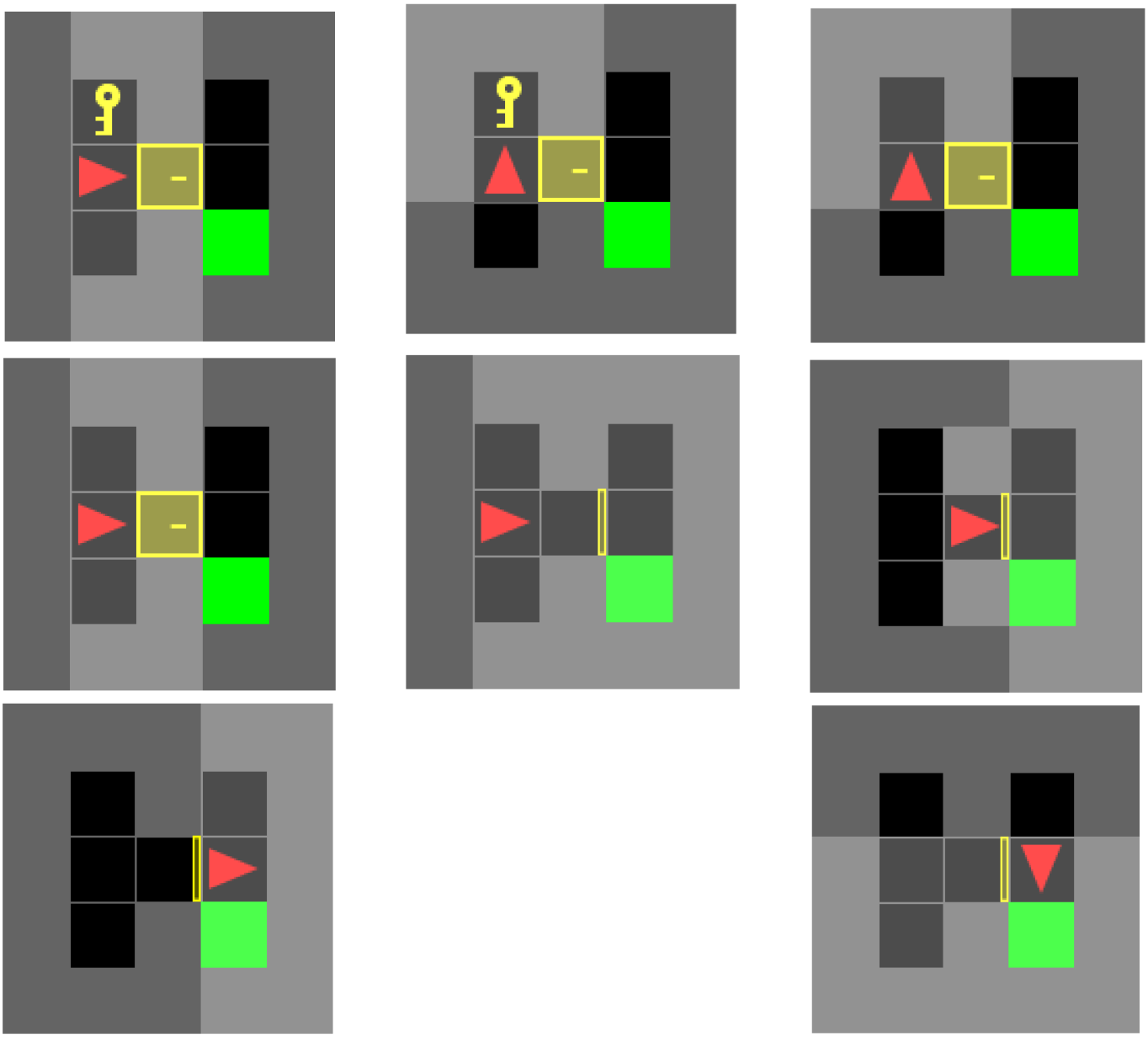

Fig. 2. part a, doorkey-5x5-normal.env.gif

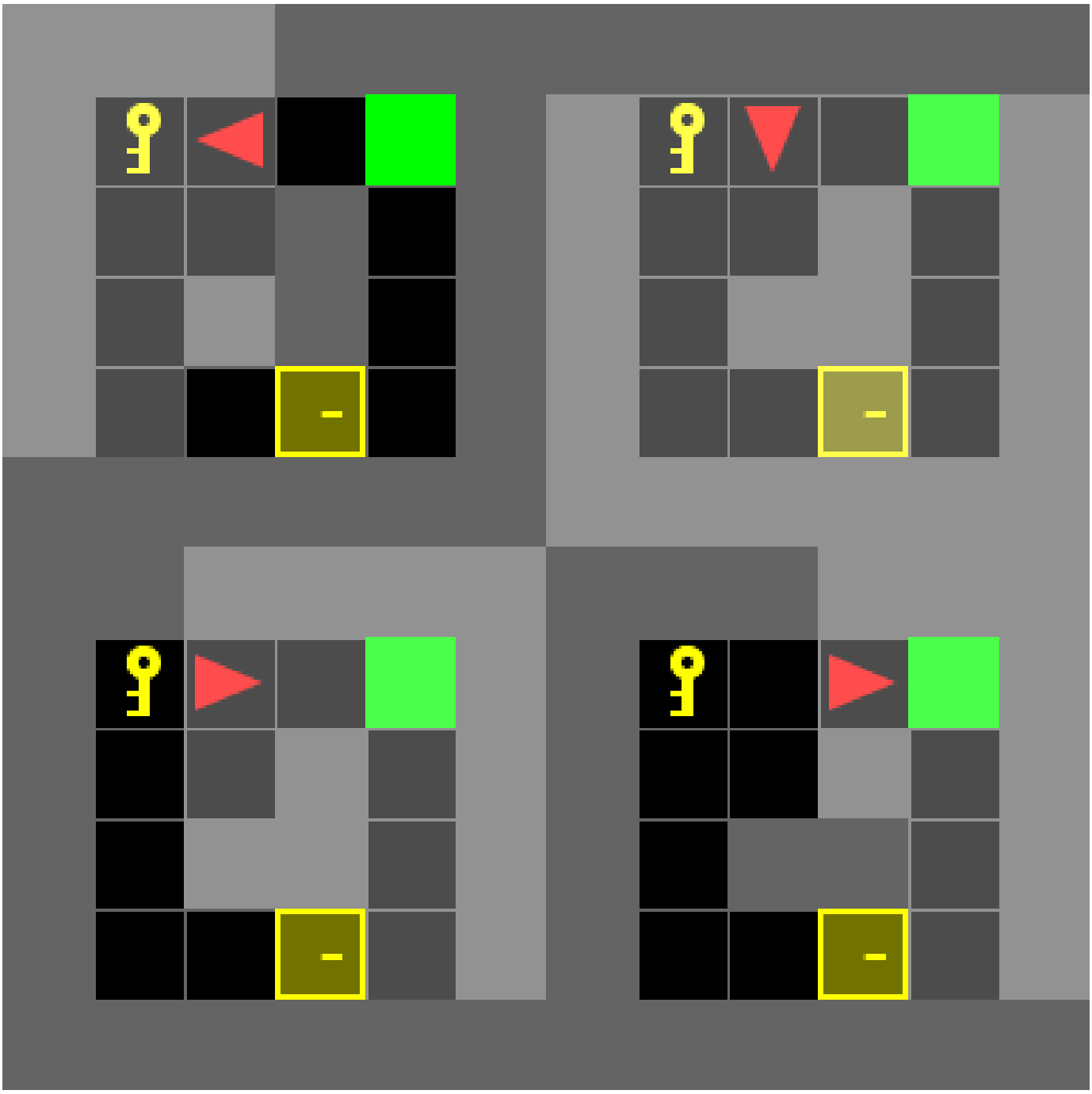

Fig. 3. part a, doorkey-6x6-direct.env.gif

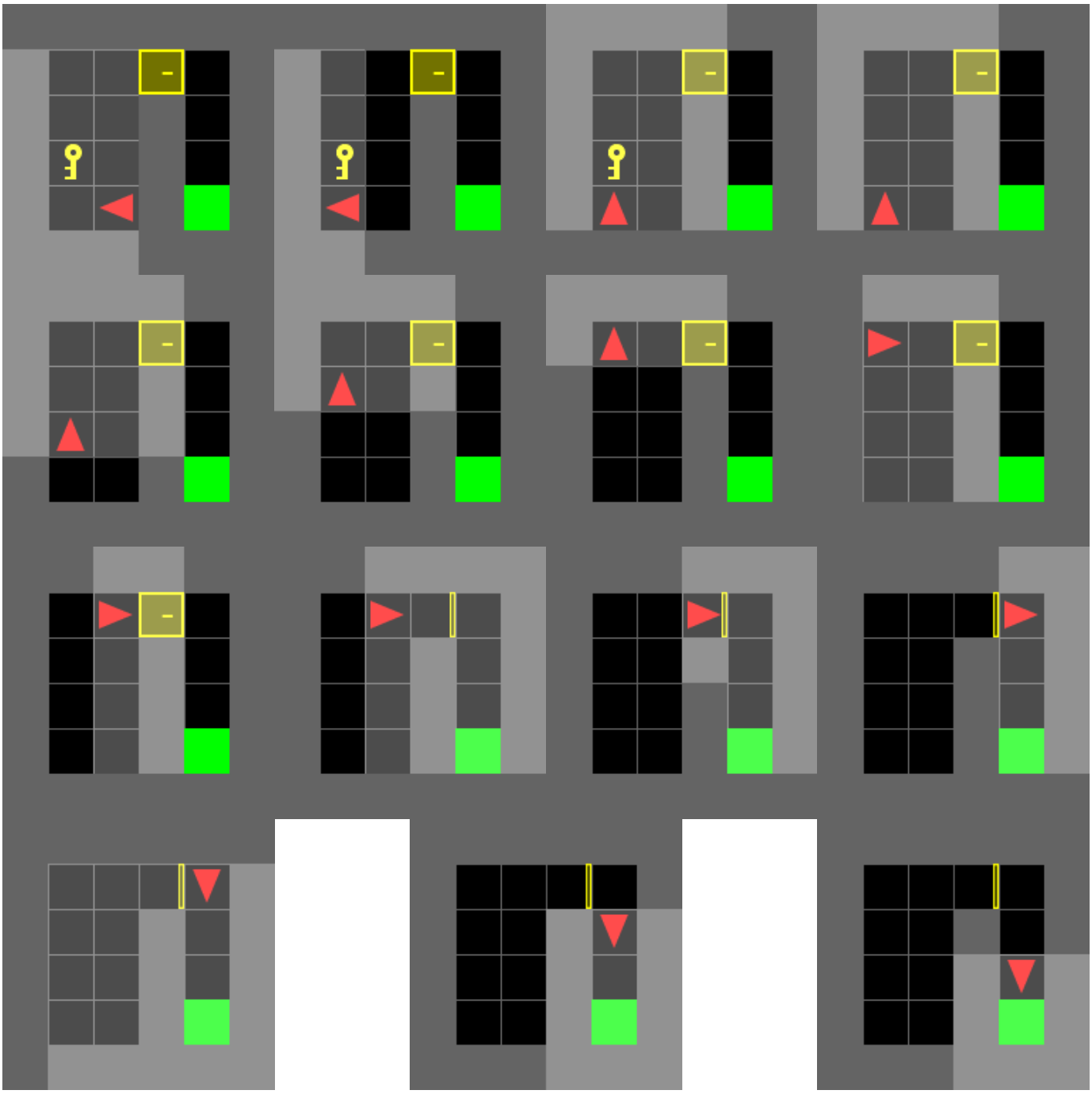

Fig. 4. part a, doorkey-6x6-normal.env.gif

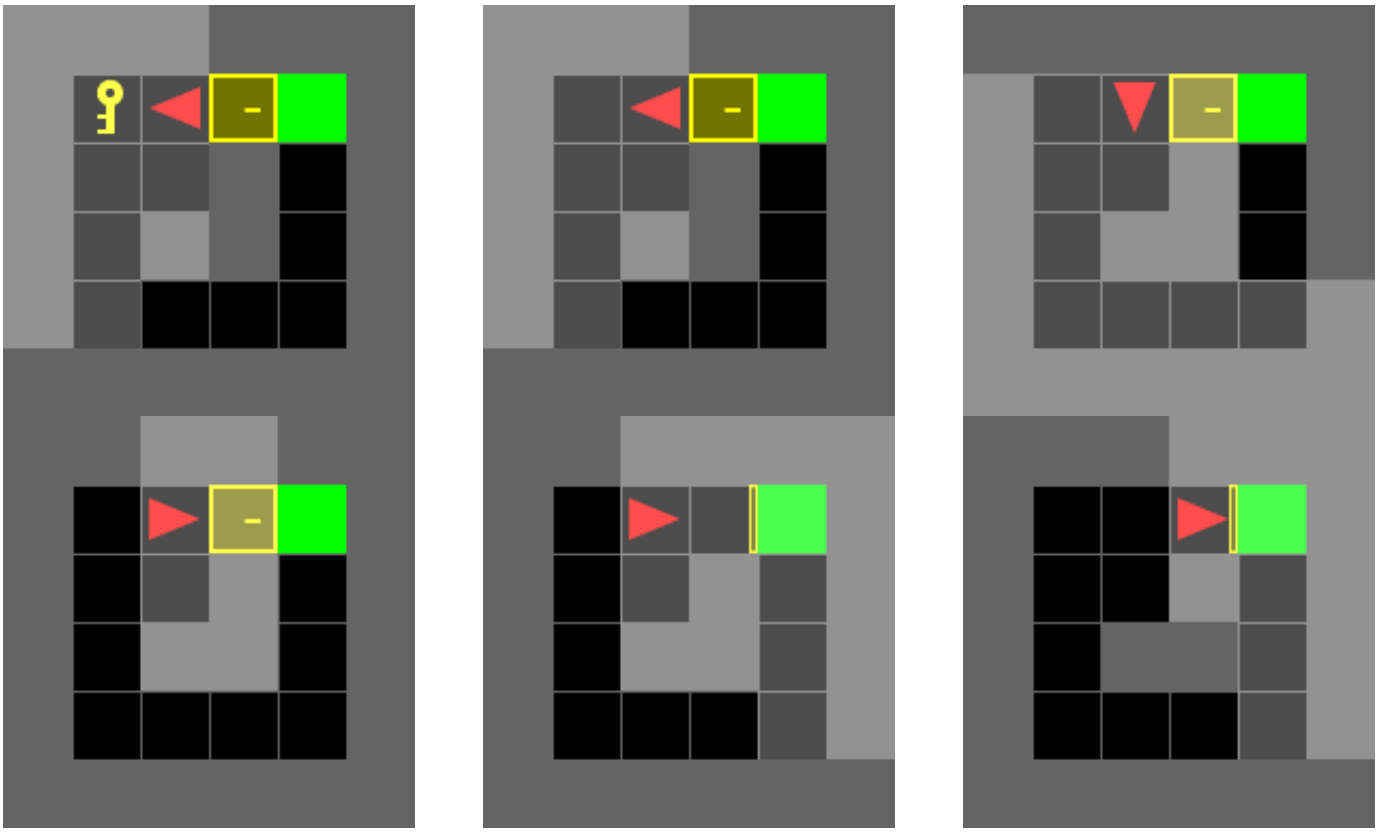

Fig. 5. part a, doorkey-6x6-shortcut.env.gif

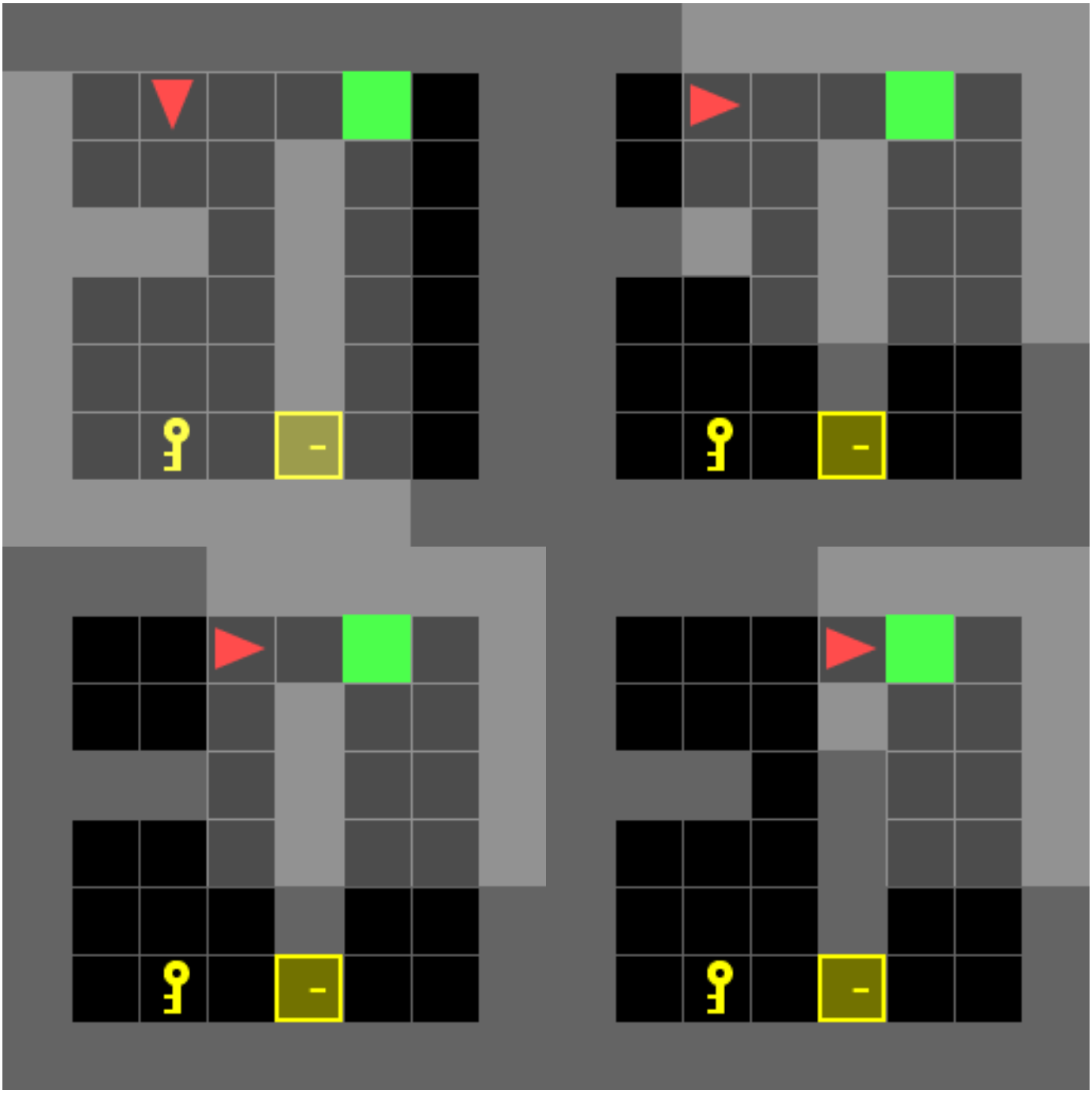

Fig. 6. part a, doorkey-8x8-direct.env.gif

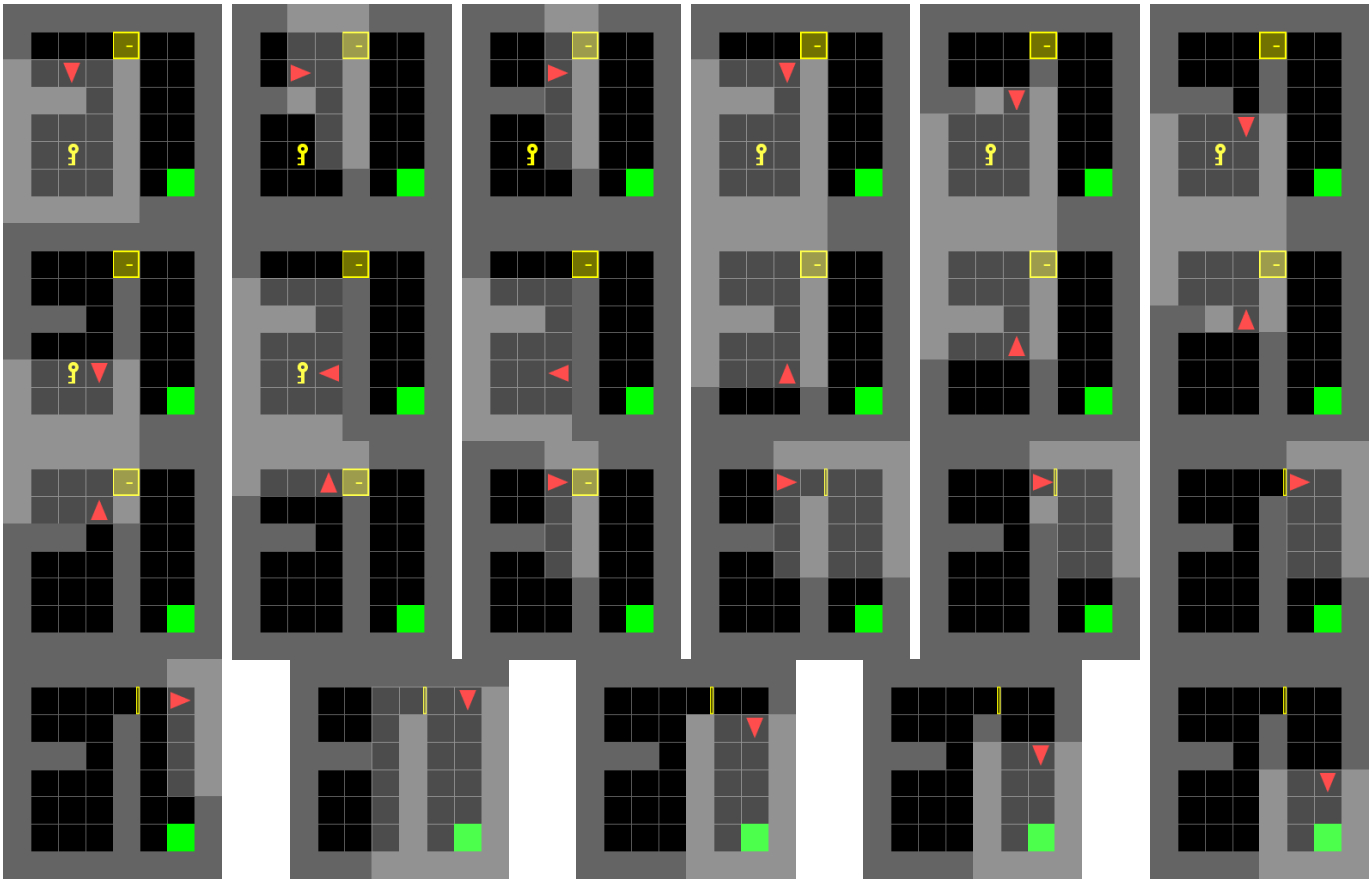

Fig. 7. part a, doorkey-8x8-normal.env.gif

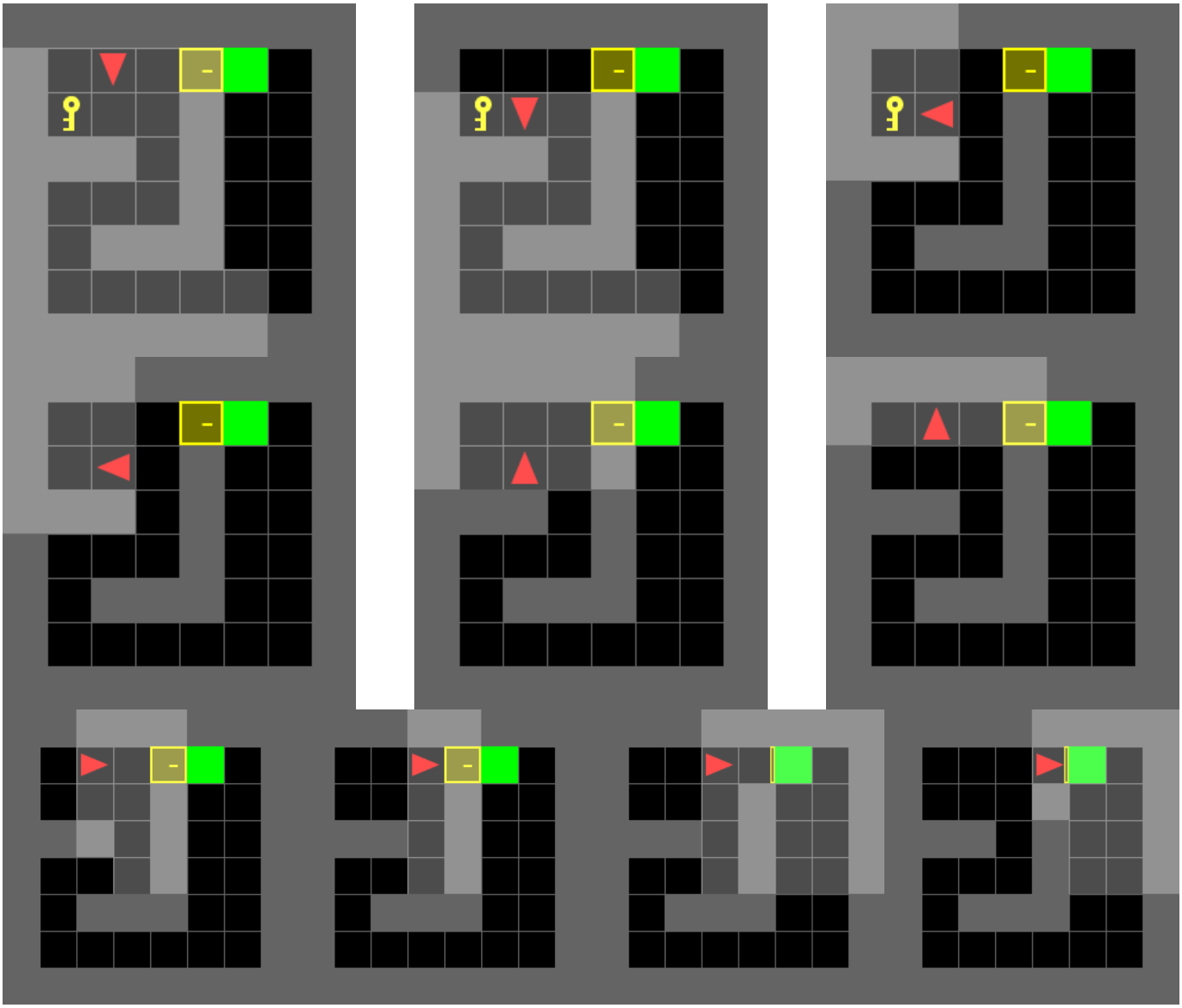

Fig. 8. part a, doorkey-8x8-shortcut.env.gif

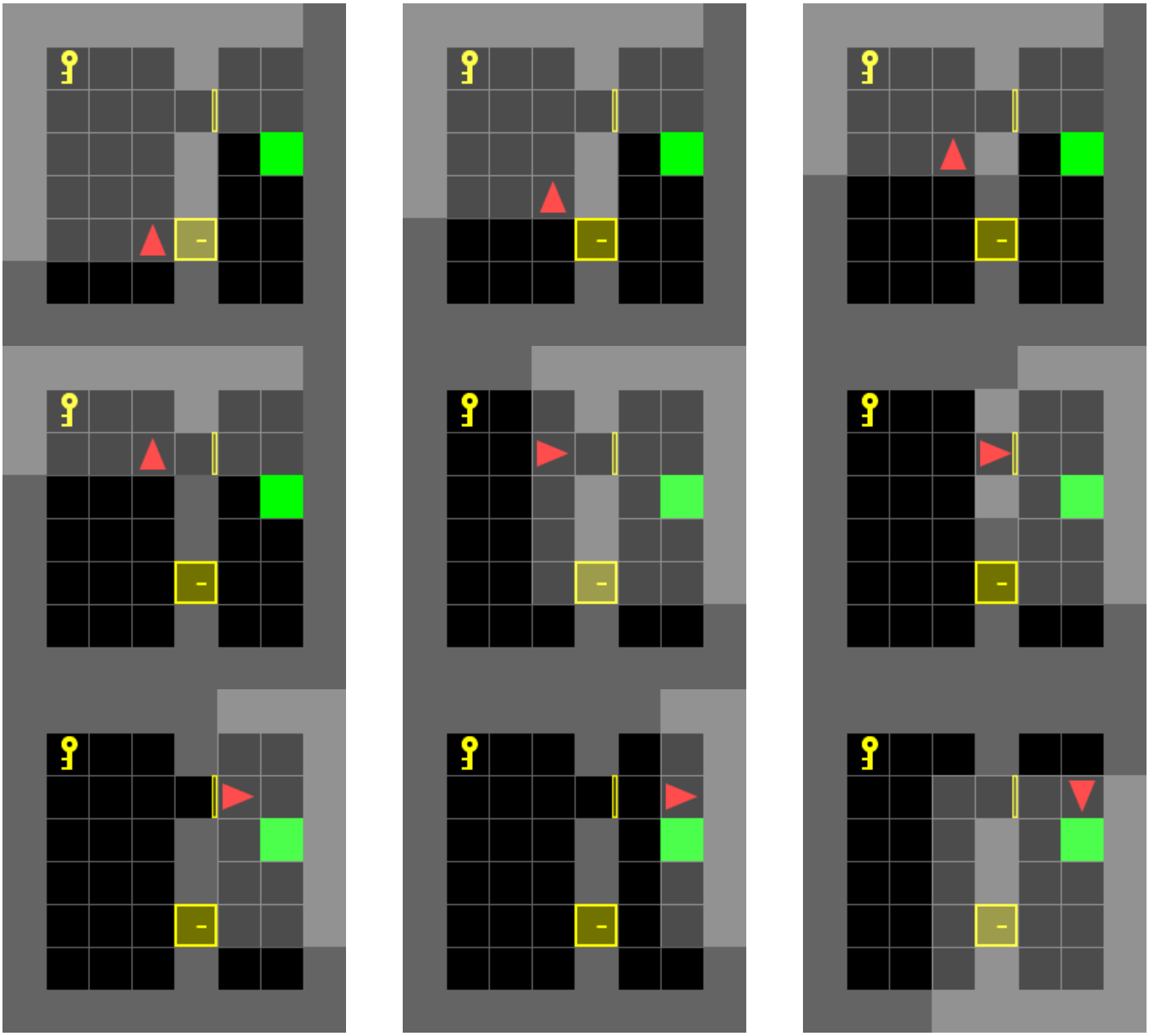

Fig. 9. part b, random1.env.gif

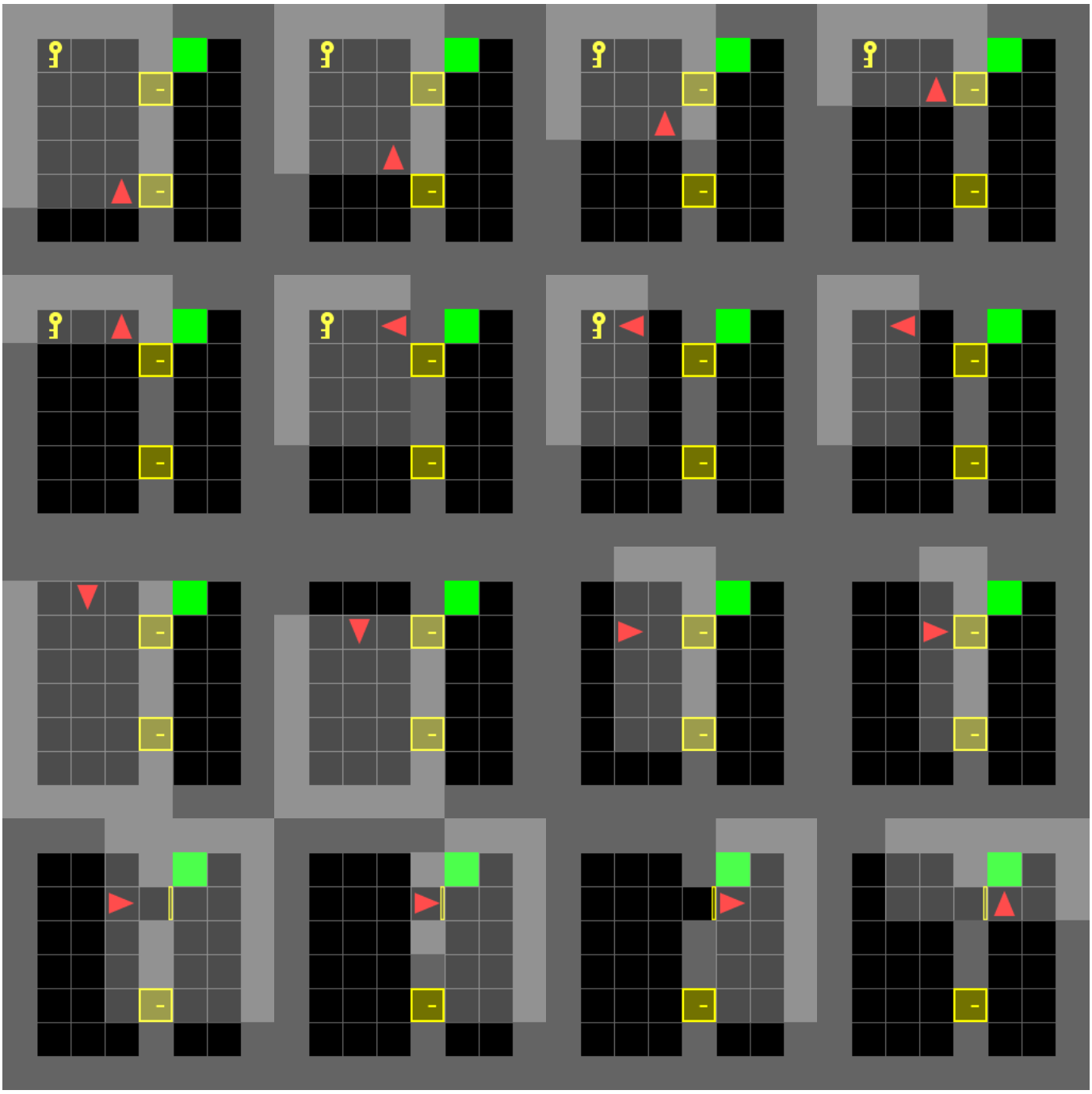

Fig. 10. part b, random2.env.gif