# Dynamic Programming to perform grid search operations

Kunaal Anand Malodhakar
Department of Electrical and Computer Engineering. University of California San Diego
kmalodhakar@ucsd.edu

## I. Introduction

Optimal control has a long history in robotics. It has widespread applications in the minimum-time problem in robotics, exhaustively used for pick-and-place robotic manipulators, motion planning of rovers and in distributed robotics applications.Dynamic programming is one of the seminal methods to compute Optimal control policies due to their guaranteed optimality and simple to understand approach.

Dynamic programming has widespread applications in fields such as Finance, path planning, Genomics and logistics. It uses the value function to structure the search for good policies It is popular due to its innate ability to handle non-convex and non-linear problems, with a guaranteed optimal solution due to the principle of optimality.It is much more efficient than a brute-force approach which evaluates all possible strategies and its computational complexity is polynomial in the number of states and actions.

Dynamic programming algorithms are usually applied backwards in time, thus evaluating an optimal solution at the last stage and then working backwards.At each step, all possible states $X \in \mathcal{X}$ are considered because we do not know a-priori which states will be visited. Dynamic programming is well suited for Deterministic shortest path problems which involve a finite number of states,a fixed planning horizon and a fully-observable map. The most popular of Deterministic shortest path problems is the grid search problems which aim to determine the shortest path for the agent to reach its goal starting from its initial position. In the next section we formally define this problem and aim to design a dynamic programming algorithm to compute the shortest path for the same in subsequent sections.

## II. Problem Formulation

Our goal is to develop an optimal control policy to provide the shortest path for an agent in the grid to start from its initial position to its goal. We first assume that we know the grid map and later consider the special case where map is unknown i.e, The goal cell and key cell may be in any one of the given set of cell positions.The environment also comprises of doors that may be open or closed. These closed doors can be opened with a key that exists at a certain cell in the grid. The agent must be guided to fetch the key if the shortest path passes through a closed door. A typical Grid search includes all the cell positions $(\S, \dagger)$ of the grid as its states. By convention, we choose the x-direction such that it increases from left to right and y-direction such that it increases from top to bottom. We may reduce the number of states by discarding cells which contain walls or obstacles as the agent cannot enter those states. This significantly improves computation time and memory for larger grid spaces. We assume a 4 connected grid such that the agent has only four possible headings: North, East, South and West. We define a state $h \in (0, 1, 2, 3)$ to hold the heading of the agent such that '0' stands for pointing East and we define other directions by an increment of one as we turn clockwise(3 for North) The state transitions in the grid are assumed to obey Markov assumption.

### A. Known map

We first define a cost function $l(x, u)$ which indicates the stage cost to make the transition from state x using action u. This constitutes a Markov decision process. To capture toggle status of the doors we add additional state $d_o$ to the original state space.We further extend the state space by adding a state $k_h$ to check if the agent is holding the key.This concludes our state space $\mathcal{X}$

$$\mathcal{X} = (x, y, h, d_o, k_h)$$

We define the action space as the possible actions that our agent can undertake. We define the agent to have three possible transitions concerning moving across cells and changing its heading: 'Move Forward(MF)', 'Turn Left(TL)' and 'Turn Right(TR)'. Turn Right corresponds to a clockwise rotation within the cell itself and Turn left corresponds to an anti-clockwise rotation. These two actions do not effect transitions across cells but changes the agent's heading. Move Forward will enable the agent to move to the cell it is facing (provided by the heading).Additionally, to unlock the door with the key we define a special action 'Unlock Door(UD)'. We also define an action 'Pickup Key(PK)' for our agent to pick up the key. These actions constitute our action space $\mathcal{U}$

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

With the state space $\mathcal{X}$ and action space $\mathcal{U}$, we now define the motion model $f(\mathcal{X},\mathcal{U}) = f(x, y, h, d_o, k_h, u)$for our agent. Given the current cell position (x,y) and heading h of the agent in $\mathcal{X}$, we find the cell in front of the agent ( for an agent at (1,2) facing east, its front cell will be (2,2) based on the grid cell convention). This is vital for actions such as Moving Forward, Unlock Door and Pickup Key as we will see later.To simplify the motion model we will consider the function f(X,U) for each action u. $\in (MF, TL, TR, UD, PK)$.

*1) u =MF:* Let us first consider the case when the desired action is Moving Forward (u=MF).We first check if the front cell is an obstacle or the wall surrounding the grid.In either of the cases, the agent won't be able to move forward. Thus the state remains unchanged and the agent remains at the same position with a similar heading as before.We define a similar behaviour when the door is closed and the agent does not posses the key. The agent is not allowed to move to the cell with the closed door without unlocking it with the key. Thus it remains at the same position and heading. The grid is designed such that with the agent cannot step on the cell with the key. Thus we emulate this behaviour in the motion model by returning the same position and heading when it is asked to move to the cell with the key. Barring these aforementioned circumstances, the agent can move to its front cell, resulting in the change of the current position (x,y) of the agent without affecting its heading.

*2) u =TL,TR:* Next we consider the case when the we need to define $f(\mathcal{X},\mathcal{U})$ when u = TR or u=TL i.e, the agent is asked to turn right or left.In both the cases, the cell position of the agent does not change. The heading of the agent changes in accordance with the definition of the $h$ state in $\mathcal{X}$.We define the agent facing East as h=0 and define other directions by an increment of one as we turn clockwise(1 for South, 2 for West, 3 for North). for u=TR(clockwise rotaiion, the motion model increments h by 1, every time we turn right, and wraps around at h=3( at h=3, u=TR will result in h=0). Similarly for Turn Left, the motion model decrements h by 1, every time we turn left, and wraps around at h=0( at h=0, u=TL will result in h=3).

*3) u =PK:* Next we consider the case when the agent is asked to pickup key(u=PK).We know however that the agent can pickup the key only when the front cell has the key in it. We thus set $k_h(k_h = 1)$, when the front cell has the key and update the state of the grid $\mathcal{X}$, indicating that the agent now possesses the key. However if the key is absent in the front cell, we do not update the state of the grid and leave $k_h$ unchanged.

*4) u =UD:* Next we consider the case when the agent is asked to unlock the door(u=UD). As defined previously,we need the key to unlock a closed door. The motion model first checks if the agent possesses the key by checking if $k_h$ is set($k_h = 1$). Then we check if the front cell contains a door. If at the current state, the agent satisfies both the conditions, we allow it to unlock the door and set $d_o$(door open). This is to ascertain that the door is now opened in the grid.

Planning Horizon($\tau$) refers to the maximum number of time steps allowed to reach the goal position. Heuristically we might place this number at the total number of cells in the grid $\tau = widthofthegridxlengthofthegrid$. However, this does not guarantee that the shortest path to the goal will be traversed in $\tau$ steps. Thus we choose an infinite Time horizon to reach the goal. Thus it is important to define the cost function $l(x, u)$ to take the minimum number of time-steps and least number of state transitions to reach the goal.

We encourage the agent to reach the goal by providing incentives for state transitions leading to the goal position(motion model is used to ascertain if the state transition leads to the goal. To achieve this, we assign a terminal cost of 0 to the goal position.We place very high costs for all transitions which do not change the state of the environment such as moving forward to the cell occupied by a wall or obstacle or cell holding the key or locked door cell.The cost function works in unison association with the motion model which does not change in the state of the environment in the aforementioned situations. We also discourage the agent to not pick up the key when the front cell does not have any key and not open the door when it does not have any door or does not possess the key through defining the cost functions for such transitions to be very high values(theoretically infinity). A uniform cost is defined for all other state transitions(typically 1 or 10).A well designed cost function reduces the computation time and saves memory as state transitions to cells with infinite cost is highly discouraged by the cost function.

The initial state of the agent represents the cell position of the agent at time t=0, its heading,the status of the door(open/closed) and if the agent has the key at time t=0(denoted by X0).This provides a starting point for the dynamic programming algorithm to trace the shortest path from this state to its goal within the least number of timesteps.With this formulation, we design the dynamic programming algorithm to determine the shortest path for the agent in the grid to reach its goal

### *B. Random map*

We define a random map as a grid where The goal cell and key cell may be in any one of the given set of cell positions.The action space $\mathcal{U}$ for a random map is the same as the one we considered in the known grid case, since the possible actions that the agent can undertake is fixed.The random map is very similar to the known map case except that it comprises of two doors instead of one , each of which may be either closed or open. Thus we extend the state space $\mathcal{X}$ to include two states $d1_o$ and $d2_o$ instead of the previously used $d_o$ to capture the status of first and the second states respectively.The motion model is also modified accordingly to accommodate for both the doors such that either of the doors is only opened when the front cell has a door and the agent possesses the key.To account for the randomness in the map with respect to key positions and goal positions, we include them as additional states in $\mathcal{X}$.We encode the key positions in state $k_pos$ and goal positions as $g_pos$ to render the modified state space for Random maps as

$$\mathcal{X} = (x, y, h, d1_o, d2_o, k_h, k_pos, g_pos)$$

With this formulation, we design the dynamic programming algorithm to determine the shortest path for the agent in the grid to reach its goal.

## III. Technical Approach

### A. Known map

We now present the DP technique of solving the problem of grid search for a known map. Let $X_t = (x_t, y_t, h_t, d_o^t, k_h^t)$ represent the state at time t.The value function $V_t(X)$ provides the total cost required to reach the goal starting at state X and time t. The control policy $\Pi_t(X)$ gives the optimal action to be performed at this state X to reach the goal through the shortest path. We start with assigning the terminal cost at the goal to the value function at the goal $V_\tau(X) = q(X_{goal_pos})$.Heuristically, we assign the time Horizon to the total number of cells in the grid( $\tau$ = 64 for an 8X8 grid). For each of the actions, we compute the Q-value, which is an intermediate to find the best action to be performed at that state and the cost associated with the same.The Q- value for state X,action $u \in \mathcal{U}$ and time t is given by

$$Q_t(X, u) = l(X, u) + V_{t+1}^*(X)$$

where $V_{t+1}^*(X)$ is the cost computed in the later time-step (t+1) and $l(X, U)$ is the stage cost that we defined as the cost function l.From this we compute $V_t(X)$ and $\Pi_t(X)$ as

$$V_t(X) = \min_{u \in \mathcal{U}} Q_t(X, u)$$

$$\Pi_t(X) = arg\min_{u \in \mathcal{U}} Q_t(X, u)$$

The Pseudo code for the same is given in 1.

Once we define the value function and policy table across all possible states, to find the optimal path we trace the shortest path starting from initial state $\mathcal{X}$, and use the greedy technique to find the best action as per the policy table, that renders minimum cost. This is explained in 3

### B. Random map

For the random map, as mentioned in section II-B, we change the state space $\mathcal{X}$ to include two different door states with new inclusion of key and goal positions. The Dynamic programming algorithm remains same as the one in Known map, except the Markov decision process that is provided as input to the algorithm in 2. To find the optimal path we trace the shortest path starting from initial state, we employ a similar strategy used in the previous section for known maps given in 3.

**Algorithm 1** Dynamic programming for known grid search

**Require:** $MDP : (\mathcal{X} \in (x, y, h, d_o, k_h),$
$\mathcal{U} \in (MF, TL, TR, PK, U), X_0, f(x, u), \tau, l(x, u), q(x_\tau))$
$V_\tau(X) \leftarrow q(X_{goal_pos})$ ▷ Value at goal state =0
**for** $t = (\tau - 1)...0$ **do**
  **for** $u = MF, TL, TR, PK, UD$ **do**
    $Q_t(X, u) \leftarrow l(X, u) + V_{t+1}^*(X)$
  $V_t(X) \leftarrow \min_{u \in \mathcal{U}} Q_t(X, u)$
  $\Pi_t(X) \leftarrow arg\min_{u \in \mathcal{U}} Q_t(X, u)$
**return** $V_t(X), \Pi_t(X)$ ▷ minimum cost, optimal policy

**Algorithm 2** Dynamic programming for Random grid search

**Require:** $MDP : (\mathcal{X} \in (x, y, h, d1_o, d2_o, k_h, k_{pos}, g_{pos}),$
$\mathcal{U} \in (MF, TL, TR, PK, U), X_0, f(x, u), \tau, l(x, u), q(x_\tau))$
$V_\tau(X) \leftarrow q(X_{goal_pos})$ ▷ Value at goal state =0
**for** $t = (\tau - 1)...0$ **do**
  **for** $u = MF, TL, TR, PK, UD$ **do**
    $Q_t(X, u) \leftarrow l(X, u) + V_{t+1}^*(X)$
  $V_t(X) \leftarrow \min_{u \in \mathcal{U}} Q_t(X, u)$
  $\Pi_t(X) \leftarrow arg\min_{u \in \mathcal{U}} Q_t(X, u)$
**return** $V_t(X), \Pi_t(X)$ ▷ minimum cost, optimal policy

**Algorithm 3** Tracing optimal policy

**Require:** $V(X)\ \forall X$ ,$\Pi(X)\ \forall X$
opt cost ←$V_0(X = X_0)$ ▷ Value at initial state
t ← 0 ▷ Start at t=0 as agent starts at initial state
current state ←$X_0$
**while** not reached goal position **do**
  action ←$\Pi_t(X)$ ▷ fetch action from policy table
  current state ← $move(currentstate, action)$ ▷ state transition
  optimal policy list ← $action$ ▷ add action to optimal policy
  t ← $t + 1 -$
  **return** optimal policy list

## IV. Results

### A. Known map

We consider 7 different maps of varying grid dimensions ( 5x5, 6x6 and 8x8) to determine if our algorithm can find the optimal path in different grids and different initial positions.for the cost function, we choose the uniform cost as '10' for all non-exceptional cases and infinite cost for cases where the state does not change when the action is performed.We retrieve the initial position of the agent, its initial heading, status of the door,key holding from the grid. The terminal cost for the goal position across all different headings is initialized to '0' irrespective of the door status and the key holding state.Using 1, we obtain the policy table and optimal cost. Following this we use 3 to trace the optimal policy. Table Irepresents the optimal control policy and the optimal cost for each of the 7 maps. For each of these maps, the images representing the optimal path the agent follows to reach the goal is given in the figures 1 with the first figure corresponding to the given original grid and the final grid indicating the agent reaching the goal

### B. Random map

Given a list of random maps with different goal and key positions, the size of the grid is fixed at 8x8 and the perimeter is surrounded by walls. There is a vertical wall at column 4 with two doors at (4, 2) and (4, 5). Each door can either be open or locked (requires a key to open). The key is randomly located in one of three positions (1, 1), (2, 3), (1, 6) and

TABLE I: Optimal control policy and cost for known maps

| Map name | Optimal policy | Optimal cost |
|---|---|---|
| doorkey-5x5-normal | [1, 3, 2, 4, 0, 0, 2, 0] | 80 |
| doorkey-6x6-normal | [0, 2, 3, 0, 0, 0, 2, 0, 4, 0, 0, 2, 0, 0, 0] | 150 |
| doorkey-6x6-direct | [1, 1, 0, 0] | 40 |
| doorkey-6x6-shortcut | [3, 1, 1, 4, 0, 0] | 60 |
| doorkey-8x8-direct | [1, 0, 0, 0] | 40 |
| doorkey-8x8-shortcut | [0, 2, 3, 2, 0, 2, 0, 4, 0, 0] | 100 |
| doorkey-8x8-normal | [1, 0, 2, 0, 0, 0, 2, 3, 2, 0, 0, 0, 0, 2, 4, 0, 0, 0, 2, 0, 0, 0, 0, 0] | 240 |

the goal is randomly located in one of three positions (5, 1), (6, 3), (5, 6). The agent is initially spawned at (3, 5) facing up. The uniform stage cost is fixed at '10' as in the known map case. The figures 7 represent optimal stage transitions for 4 random maps that the agent makes to reach the goal. The fourth random map transitions are split into 2 figures to capture all major state transitions. By extending the state space to include possible positions of key and goal, we develop a single control policy that can work for any given random map. We only need to load the already computed value function and policy table to use 3 to trace the optimal policy.

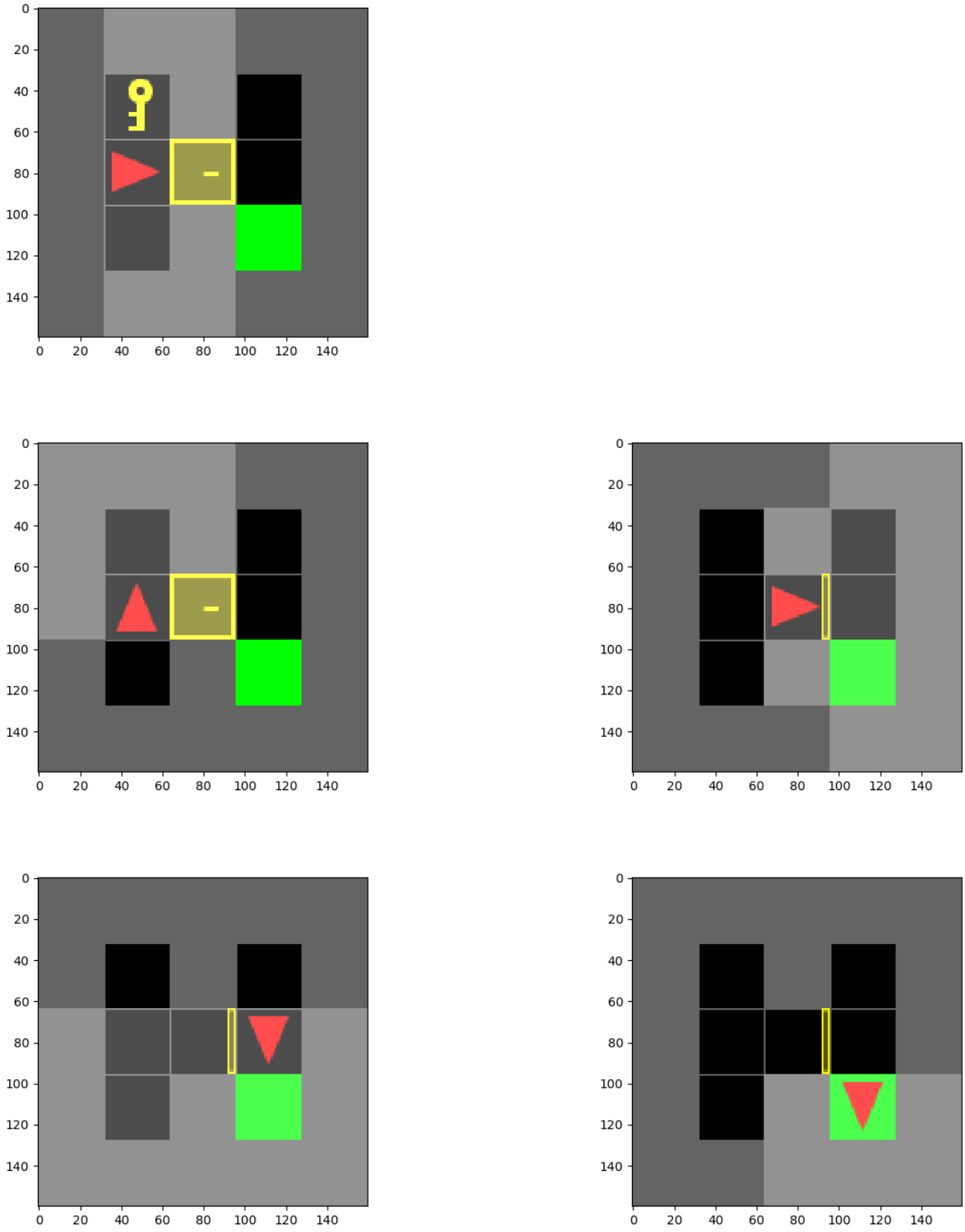


Fig. 1: doorkey-5x5-normal

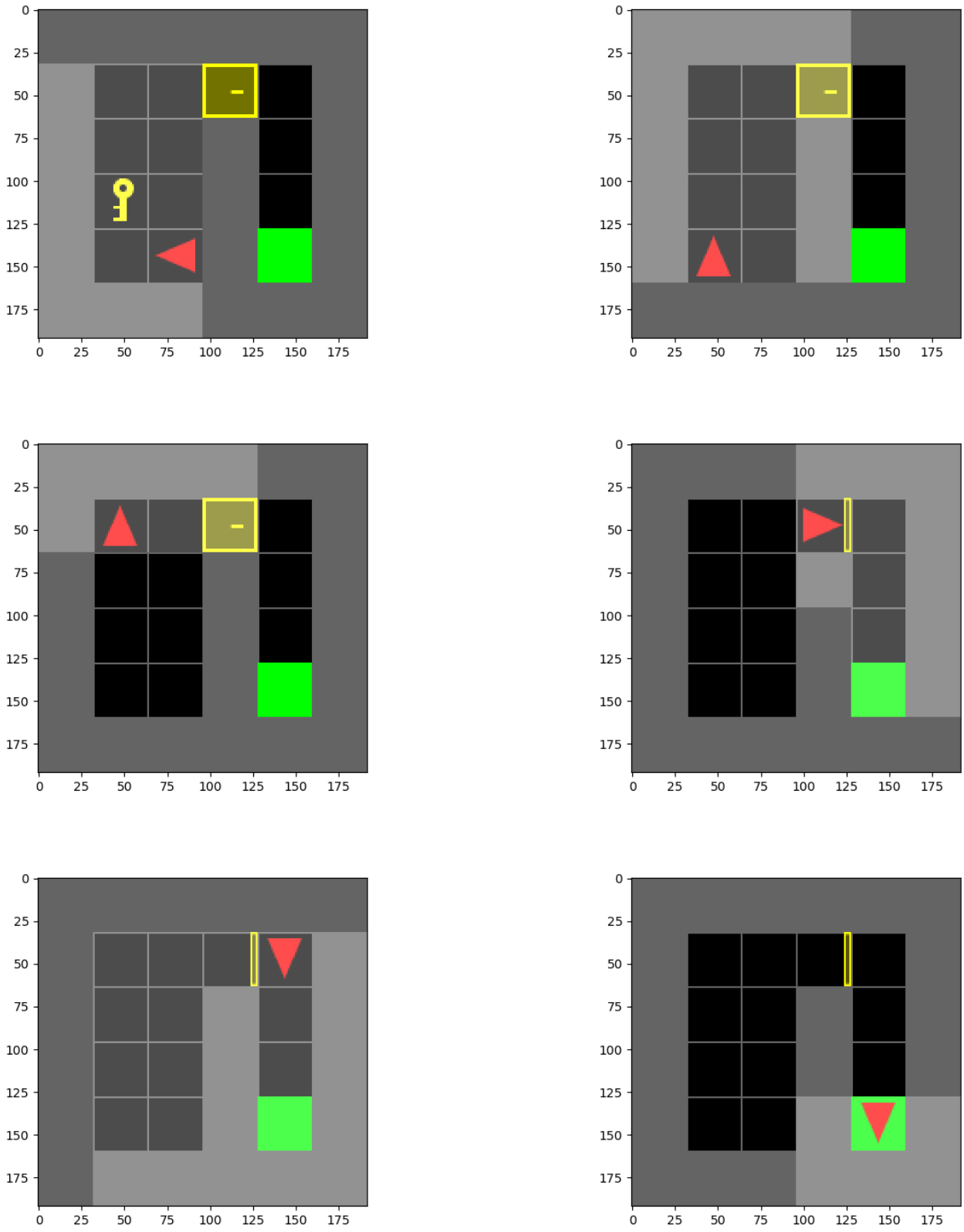

Fig. 2: doorkey-6x6-normal

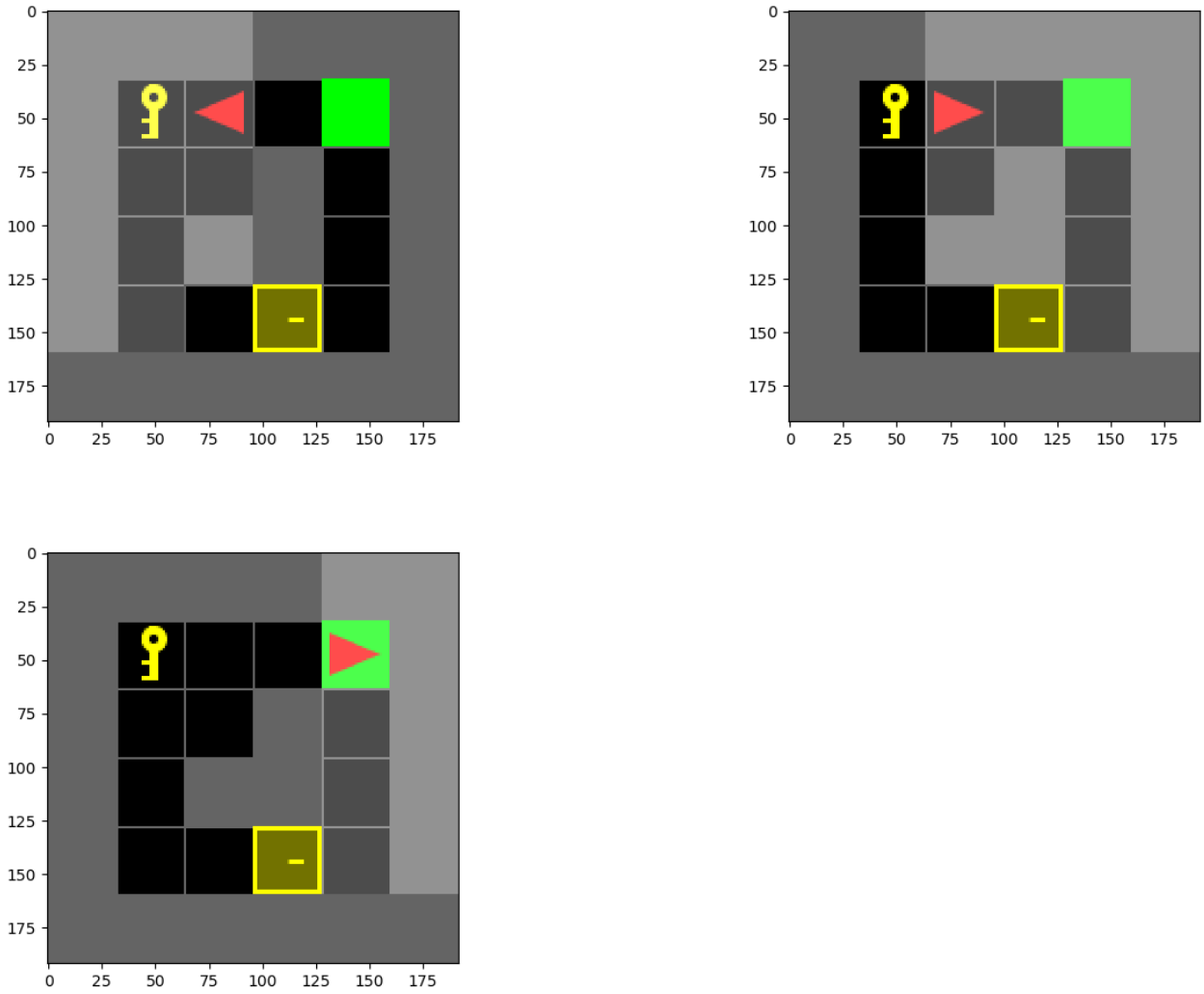


Fig. 3: doorkey-6x6-direct

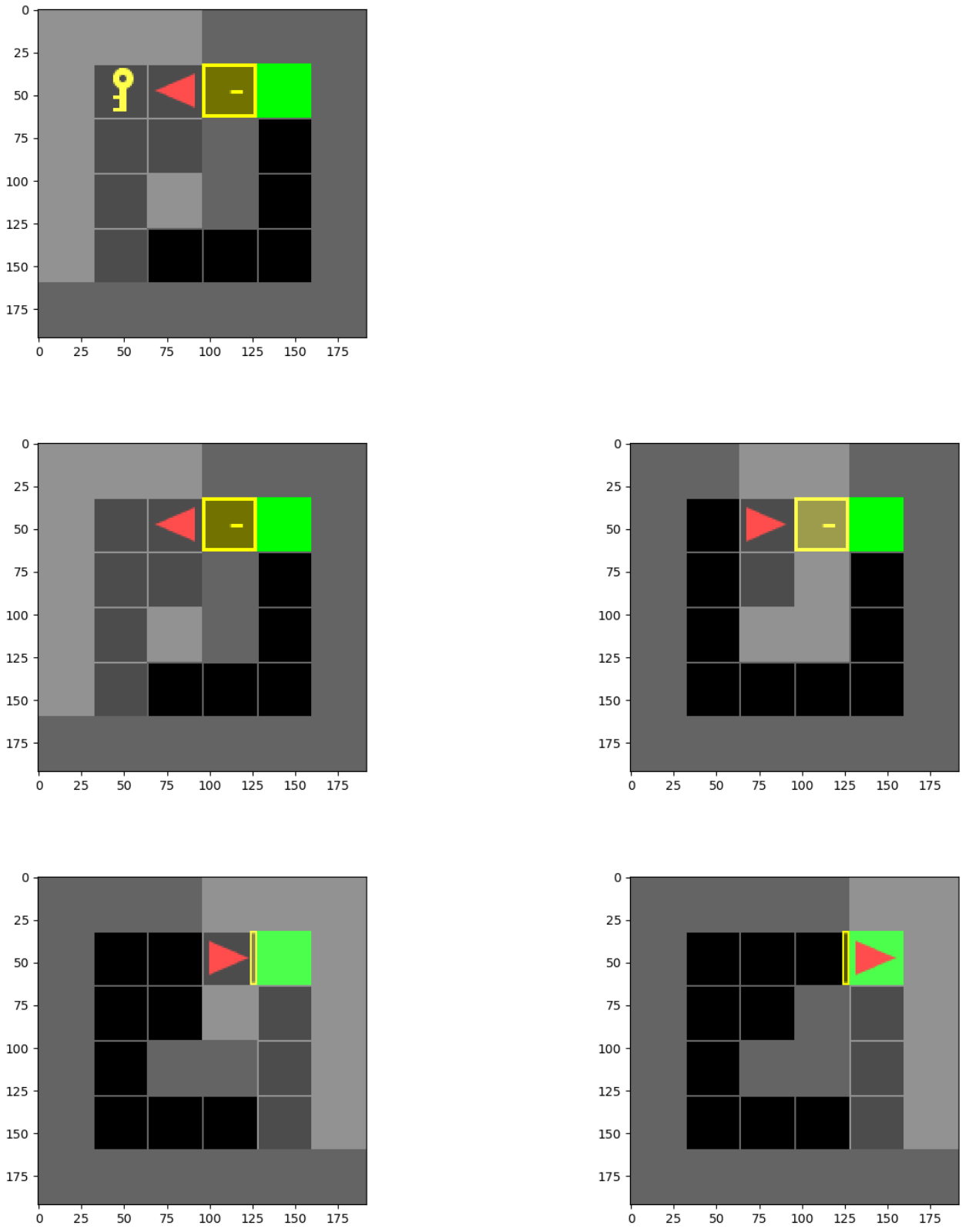


Fig. 4: doorkey-6x6-shortcut

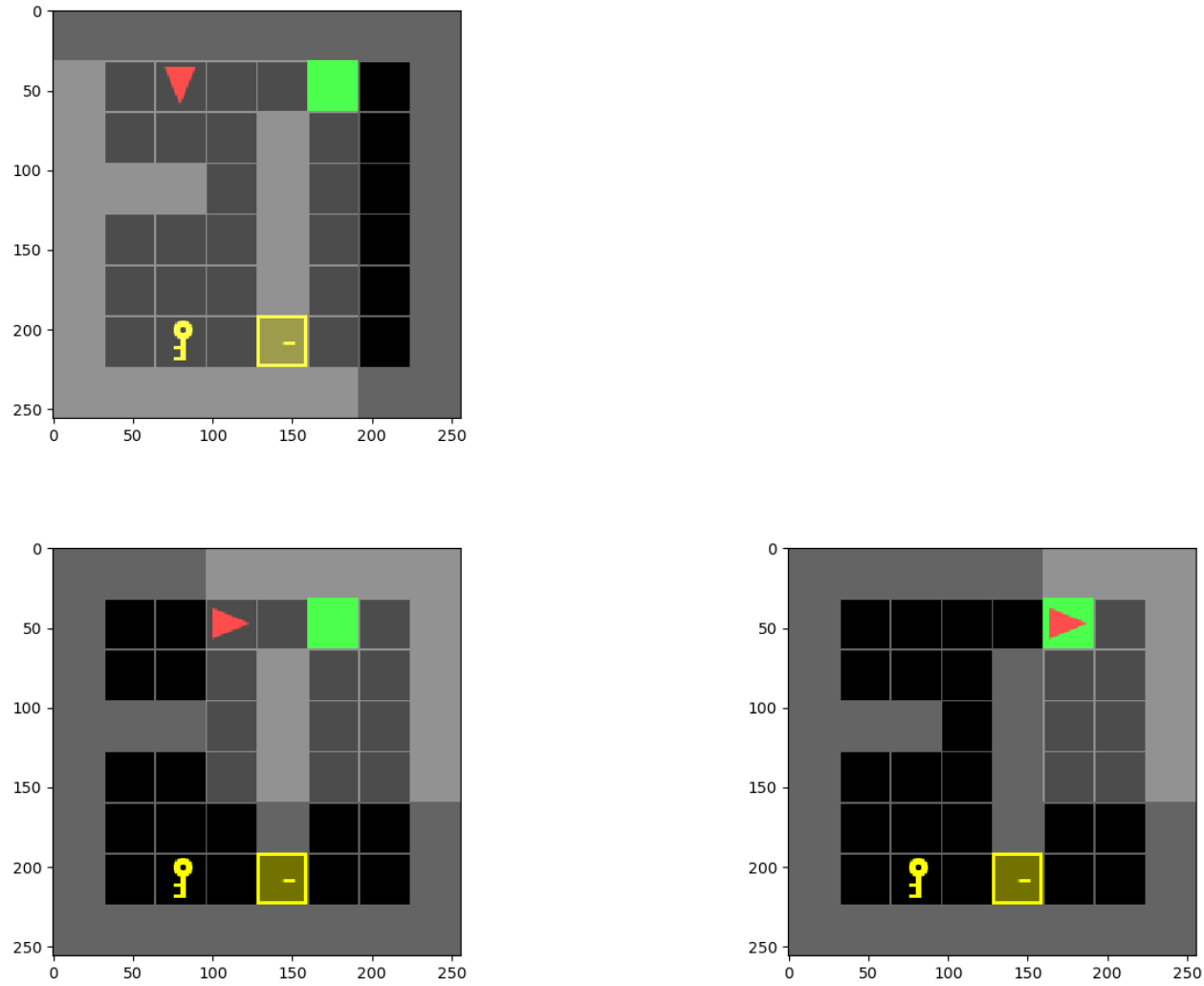


Fig. 5: doorkey-8x8-direct

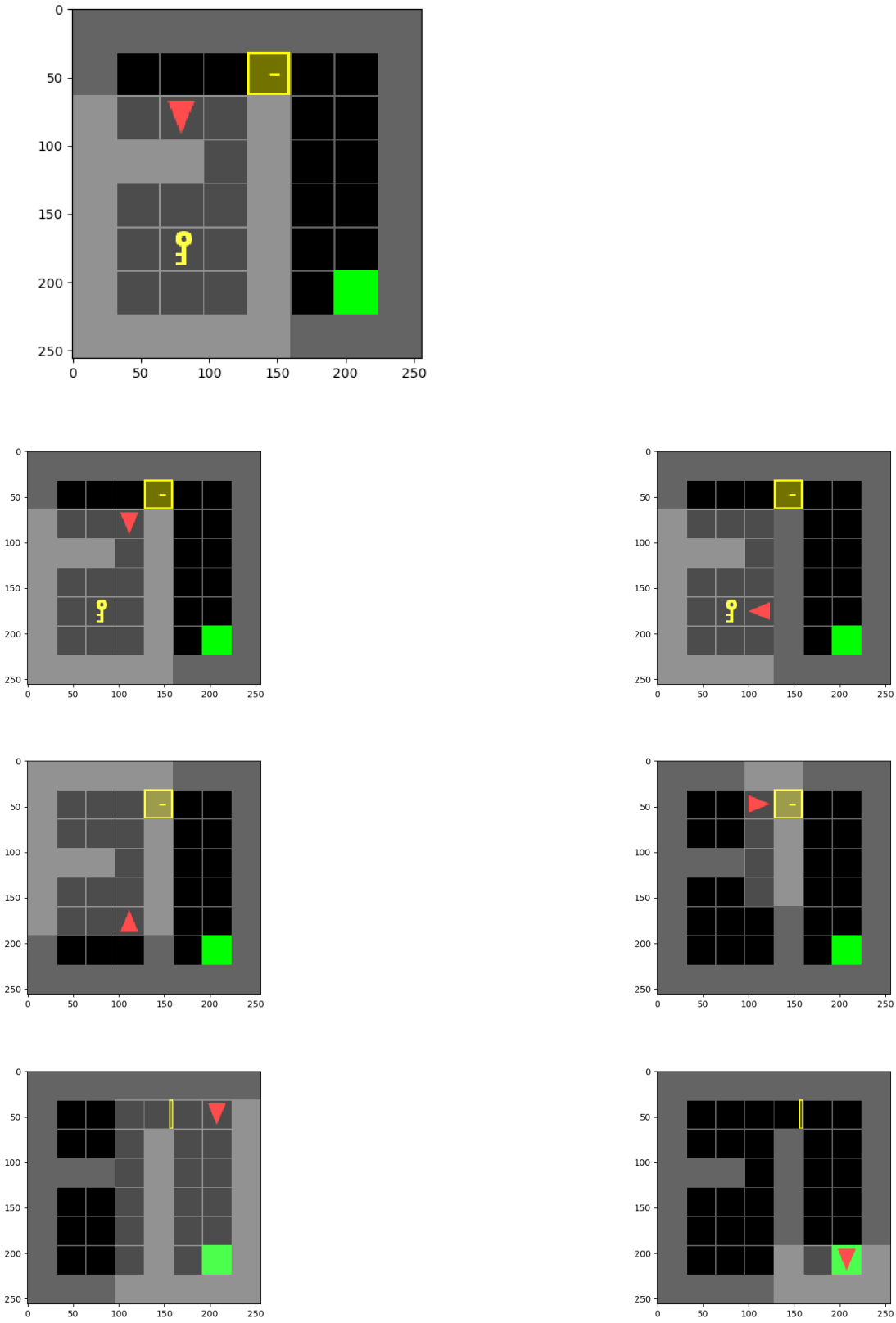

Fig. 6: doorkey-8x8-normal

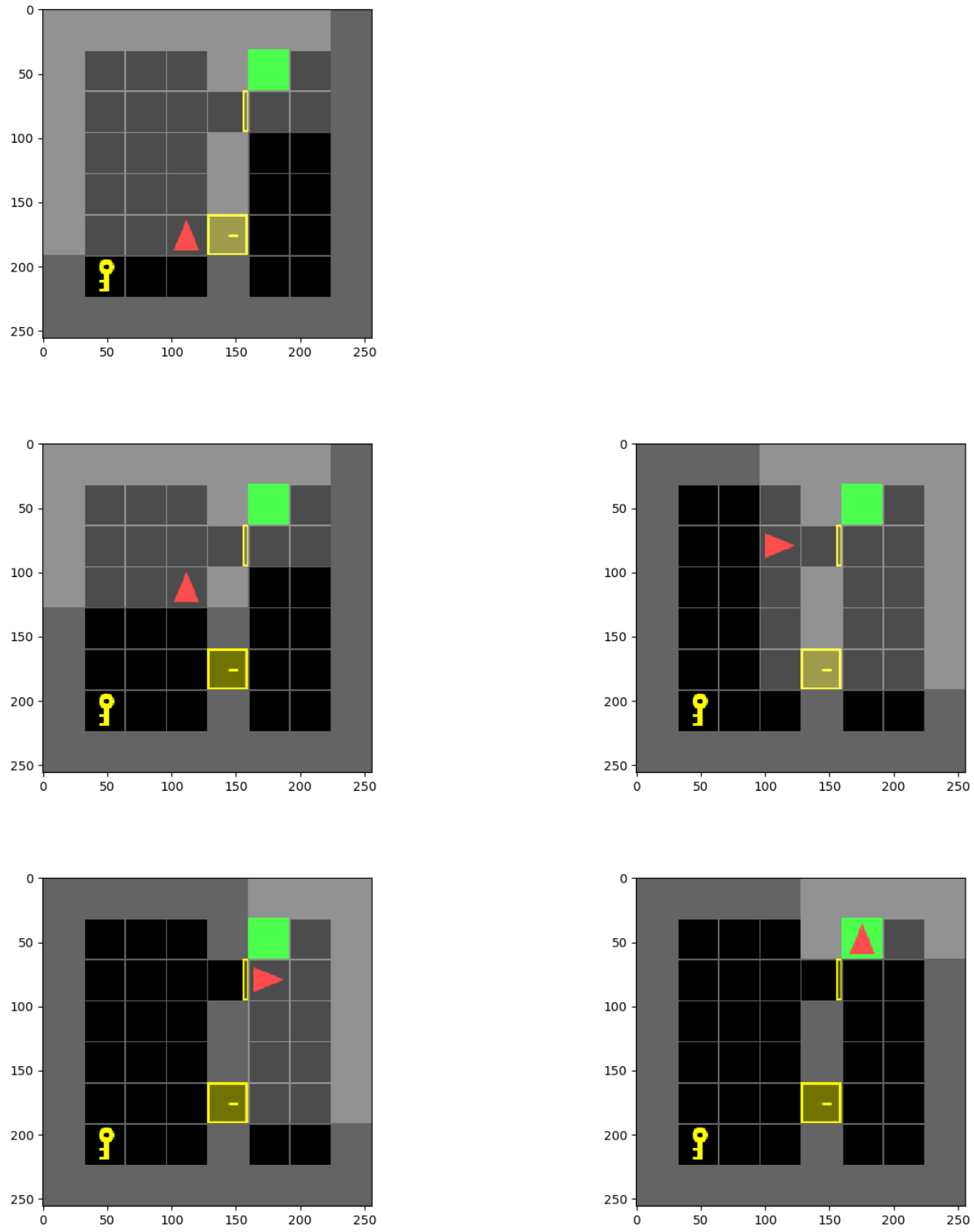


Fig. 7: random map 1

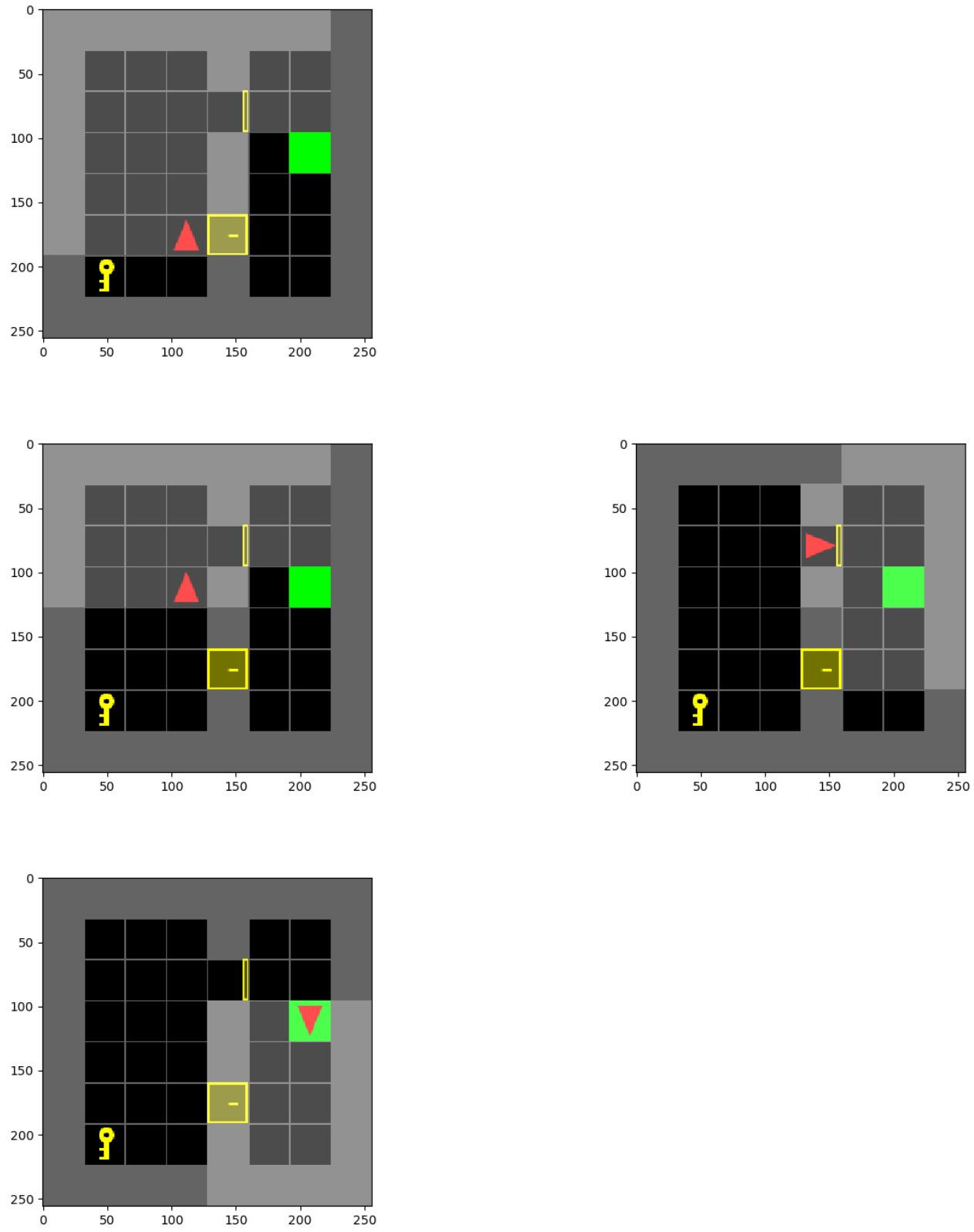


Fig. 8: random map 2

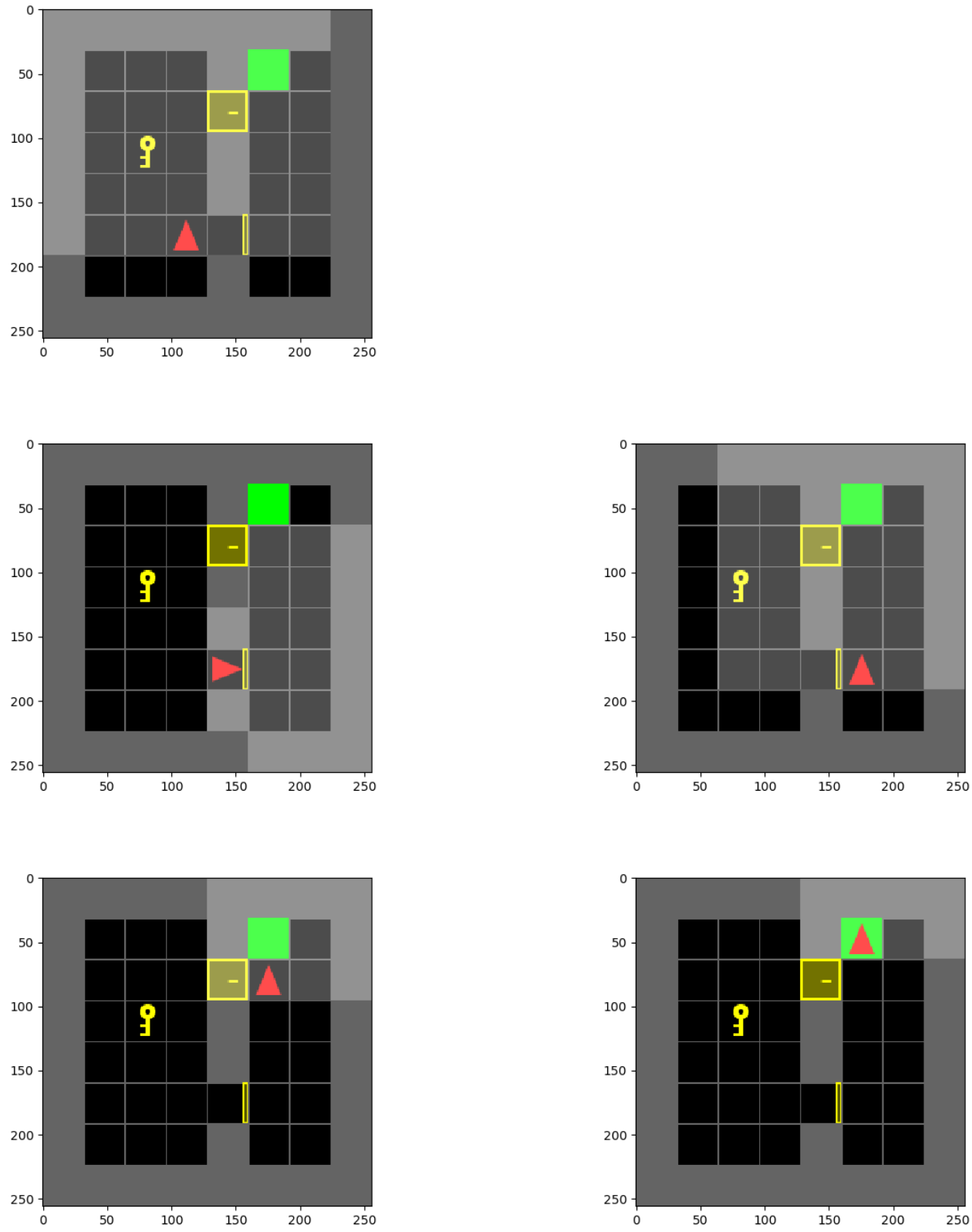


Fig. 9: random map 3

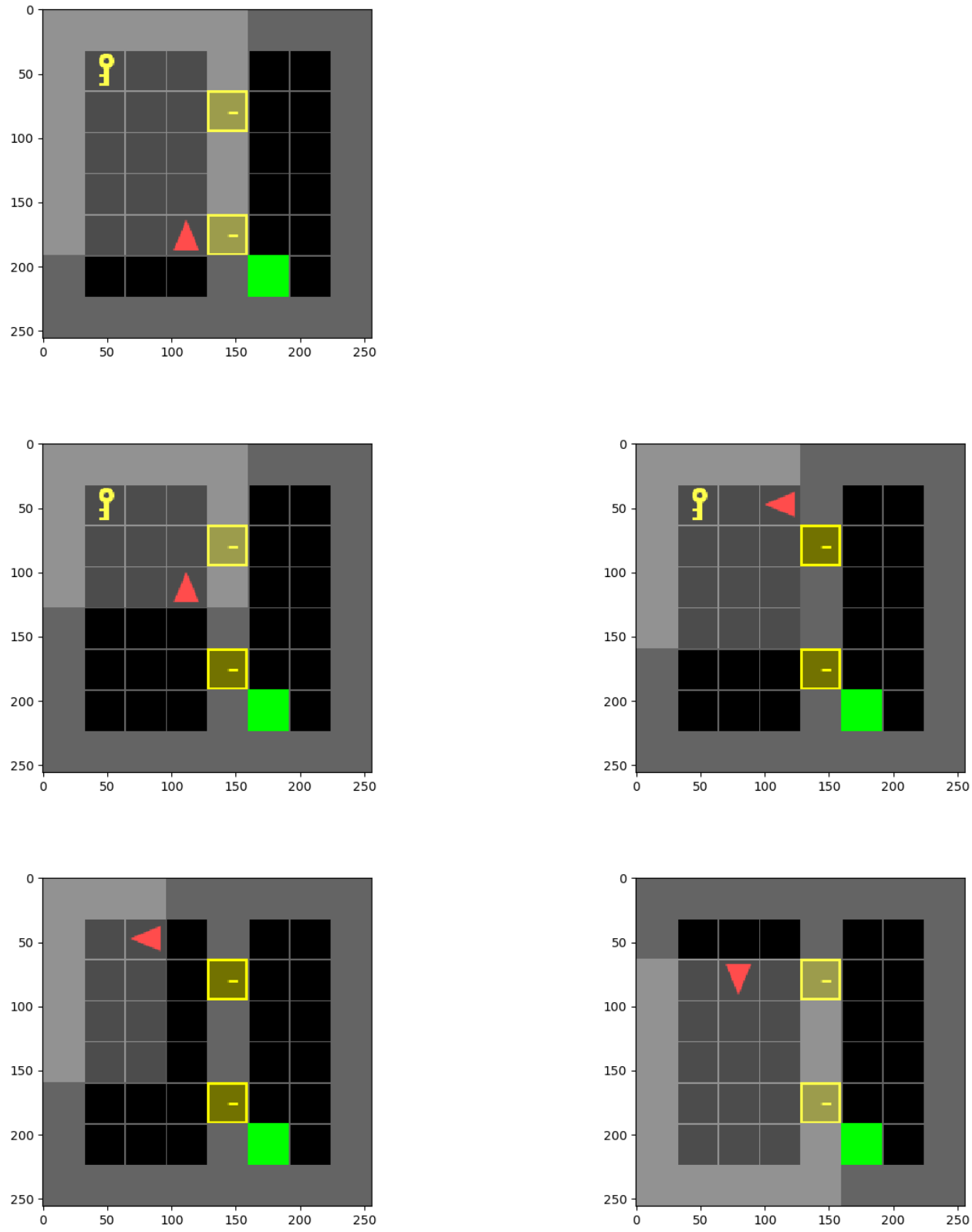


Fig. 10: random map 4-part1

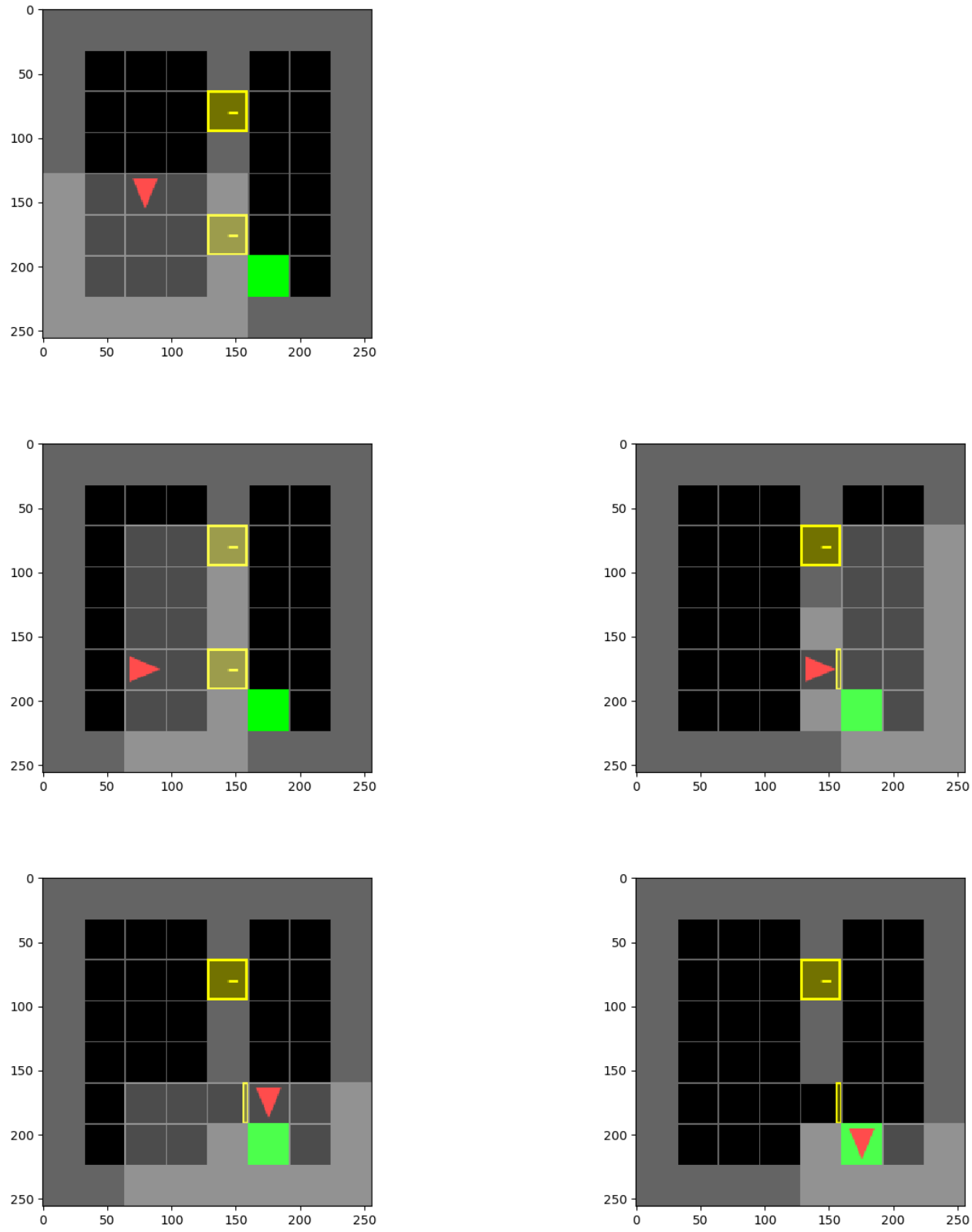

Fig. 11: random map 4-part2