Dynamic Programming

Surya Prakash Thoguluva Kumaran Babu
Department of Mechanical and Aerospace Engineering
University of California San Diego
stk222@ucsd.edu

***Abstract*—The objective of this paper is to implement Dynamic Programming to solve Door and Key environment autonomous navigation problems by modeling it as a Finite Horizon Markov Decision Process. The obtained control policy is then visualized in the environment.**

***Index Terms*—Dynamic Programming, Finite Horizon, Markov Decision Process, Control Policy.**

I. Introduction

Robot planning is an important area of research in the robotics domain. Given sensor data about the environment, the robot needs to know what it should do next to achieve a specific task. This is the exact objective of which planning and learning problems in robotics. Planning in robotics enables robots to operate in complex environments, handle uncertain situations, and accomplish tasks with a high degree of accuracy and precision. A subset of this is the robot path planning problem which can be solved in various ways. There has been a lot of research over the past few decades which tried to solve this problem. However, even today, although the high-level problem formulation is the same, we have different challenges. For example, the current auto-pilot control in some cars is still not 100% reliable and there is a lot of room for improvement.

In this paper, we have discussed the implementation of solving finite horizon optimal control problems by using dynamic programming. The Door and Key environment is modeled as a Markov decision process and an algorithm to obtain a control policy in the general environment is obtained.

II. Problem Formulation For Dynamic Programming

A. Given Environment Description

We are given 2 sets of environments. The objective is to get our agent to the goal location. The environment may contain a door that blocks the way to the goal. If the door is closed, the agent needs to pick up a key to unlock the door. The environments are described as below:

- Known Environment: In this, we are given maps of sizes 5x5, 6x6, and 8x8 where the goal and the key could be in random locations. The placement of them is also given as input in the environment variable. There will only be one door which might or might not block the path to the key.
- Unknown Environment: In this, we are given a random 8x8 map with 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 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. A general policy which can find an optimal path for all the cases is to be determined.

B. Markov Decision Process

The given problem can be formulated as a Markov decision process (MDP) by defining carefully the variables in the MDP.

- State Space χ: For our case, the states are discrete and are integers. For the known environment case, the states are defined as below:

$$\chi \in (X_{agent}, Y_{agent}, \theta_{agent}, K, D)$$

where

  - $X_{agent} \in \mathbb{N}$ denotes the agents X position
  - $Y_{agent} \in \mathbb{N}$ denotes the agents Y position
  - $\theta_{agent} \in \{0 = Left, 1 = Up, 2 = Right, 3 = Down\}$ denotes the agents facing direction
  - $K \in \{0 = Holding, 1 = Notholding\}$ denotes whether the agent has keys or not
  - $D \in \{0 = Closed, 1 = Open\}$ denotes whether the door is open or closed

The state space for random environment case is defined as below:

$$\chi \in (X_{agent}, Y_{agent}, \theta_{agent}, K, D_1, D_2, KP, GP)$$

where

  - $X_{agent} \in \mathbb{N}$ denotes the agents X position
  - $Y_{agent} \in \mathbb{N}$ denotes the agents Y position
  - $\theta_{agent} \in \{0 = Left, 1 = Up, 2 = Right, 3 = Down\}$ denotes the agents facing direction
  - $K \in \{0 = Holding, 1 = Notholding\}$ denotes whether the agent has keys or not
  - $D_1 \in \{0 = Closed, 1 = Open\}$ denotes whether the door at (4,2) is open or closed
  - $D_2 \in \{0 = Closed, 1 = Open\}$ denotes whether the door at (4,5) is open or closed
  - $KP \in \{0, 1, 2\}$ takes 0 if key is at (1,1), 1 if key is at (2,3) and 2 if key is at (1,6)
  - $GP \in \{0, 1, 2\}$ takes 0 if goal is at (5,1), 1 if goal is at (6,3) and 2 if goal is at (5,6)

- Control Input v: We are given 5 sets of inputs that can be performed on the states. v is defined as below:

$$v \in \{0, 1, 2, 3, 4\}$$

where 0 is to Move Forward, 1 is to Turn Left, 2 is to Turn Right, 3 is to Pick Key, and 4 is to Unlock Door.
- Time Horizon t: $t \in \{0, 1, .., T\}$ takes discrete values, where T is the terminal time. T in our case is set to be the total number of all possible states that can be obtained by taking all combinations.
- Prior Pdf $p_0(x)$: This gives the probability density of the inital condition. Since our case is deterministic system, prior pdf $p_0(x) = 1$ if x is in the initial state and $p_0(x) = 0$ everywhere else.
- Motion Model $p_f(.|x_t, u_t)$: This gives the transition from one state to another using the given input. Since our model is deterministic, the motion model can be expressed as:

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

- Stage Cost $l(x, u)$: The cost incurred to apply a control input u to a state x. If the motion model yields a state which is valid, then a cost of 1 is assigned else the stage cost is infinite.
- Terminal Cost $q(x)$: The terminal cost is the cost incurred for reaching the terminal state. The terminal cost is defined to be zero at the goal state and infinite everywhere else.
- Control Policy $\pi_t(x)$: It is a function from state x at time t to control input u.
- Value Function $V_t^{\pi}(x)$: It is the expected cumulative cost of starting at state x at time t and acting according to π.
- Discount Factor $\gamma = 1$

C. Objective

The objective for both parts is to implement a Dynamic Programming algorithm that minimizes the cost of reaching the goal. Mathematically,

$$min\, V_t^{\pi}(x) := q(x_T) + \sum_{\tau=t}^{T-1} l(x_\tau, \pi_t(x_\tau))$$

In the known environment case, dynamic programming can be run for each map to compute different control policies for each map. However, in part b, the dynamic programming should be run only once, and a general control policy that takes into account all the random goals, key, and door positions is computed. Later, depending on the given map, an optimal path sequence should be returned by using the computed control policy.

III. Technical Approach For Dynamic Programming

The implementation of Dynamic Programming algorithm on a high level is mentioned below:

- Create a set of all possible states that can occur for the given problem by considering all possible permutations of each elements of the state.
- Create value function matrix, control policy matrix for all the states for all the time horizon from $0, ..., T$ and initialize it to be infinity
- Create q matrix for all the states for all the control inputs from $0, 1, 2, 3, 4$ and initialize it to be infinity
- Define the motion model for the control input and the states. Consider all edge cases and assign infinite stage cost to a state if it is not feasible and assign a stage cost of 1 for all the feasible states
- Set value function at the terminal time T to be equal to the terminal cost
- For each time from $t-1, ...0$ compute the Q matrix entries for each states and for each control input by adding the stage cost for that control input and the value function entry of the output from the motion model at time $t+1$
- Take minimum of the Q matrix for all the stages and assign it to the value function, and the argument (control input) that produced it to the control policy for that particular time step.
- Terminate the algorithm when the value function of the state corresponding to the initial robot configuration has a cost which is not infinite for a particular timestep.
- Generate the motion sequence from the initial state to the target from the obtained control policy matrix

A. Initializing the Value Function, Control Policy, Q matrix

First a set of all possible states is created by creating a nested for loop for each of the state elements and assigning values. For example, in the known environment case of 5x5 environment, the first element X_{agent} can take values between $0, 1, 2, 3, 4$, the second element Y_{agent} can take values between $0, 1, 2, 3, 4$, the third element θ_{agent} can take values between $0, 1, 2, 3$, the fourth element K can take values between $0, 1$, the fifth element D can take values between $0, 1$. Hence there will be $5 \times 5 \times 4 \times 2 \times 2 = 400$ possible states. The time horizon T is set equal to the number of all possible states. Two empty matrix of size TxT is created and initialized to a large positive integer which will be treated as infinity and assigned to V_t^{π} and $\pi_t(x)$. Q matrix of size $T \times 5$ is created and initialized to infinite where 5 is the number of control inputs possible.

B. Motion Model Definition

A separate motion model function is created in python which takes in the current state, and the control input as the arguments and returns a new state and a flag.
All the feasible and not feasible states the known environment case are listed below:

- If a state X, Y corresponds to $\{None, Key, Goal\}$ in an environment then the state is feasible
- If a state X, Y corresponds to $\{Door\}$ and if in a state $D = 1$, that is the door is open, then the state is feasible
- All other states are not feasible

All the feasible and not feasible states the random environment case are listed below:

- If a state X, Y corresponds to $\{None\}$ in an environment then the state is feasible
- If a state X, Y corresponds to $\{Key\}$ at position (1,1) and if in a state $KP = 0$ then the state is feasible
- If a state X, Y corresponds to $\{Key\}$ at position (2,3) and if in a state $KP = 1$ then the state is feasible
- If a state X, Y corresponds to $\{Key\}$ at position (1,6) and if in a state $KP = 2$ then the state is feasible
- If a state X, Y corresponds to $\{Goal\}$ at position (5,1) and if in a state $GP = 0$ then the state is feasible
- If a state X, Y corresponds to $\{Goal\}$ at position (6,3) and if in a state $GP = 1$ then the state is feasible
- If a state X, Y corresponds to $\{Goal\}$ at position (5,6) and if in a state $GP = 2$ then the state is feasible
- If a state X, Y corresponds to $\{Door\}$ at position 1 and if in a state $D_1 = 1$, that is the first door is open, then the state is feasible
- If a state X, Y corresponds to $\{Door\}$ at position 2 and if in a state $D_2 = 1$, that is the second door is open, then the state is feasible
- All other states are not feasible

The pesudo-code for the motion model function for the known environment is as follows:

- Input: State, Control Input
- Output: New State, Flag
- If State is not feasible then return State, False
- If State is feasible then
- If Control Input = 0
  - Apply Move forward by adding the current position and the direction vector and assign to New State
  - Check if the New State X, Y is within bounds. If not return State, False
  - If New State is feasible, return New State, True
  - Else return State, False
- If Control Input = 1
  - Check the current direction and rotate the direction vector by 90 degree in the anticlockwise direction. This can be done manually as well.
  - Set the State's Θ to be the new direction and assign it to New State
  - Return New State, True
- If Control Input = 2
  - Check the current direction and rotate the direction vector by 90 degree in the clockwise direction.
  - Set the State's Θ to be the new direction and assign it to New State
  - Return New State, True
- If Control Input = 3
  - Check if the key position is in front of the current position by applying move forward input.
  - If yes, then check the State's $K == 0$ to see if we do not have the key with us.
  - If yes, then set $K = 1$ in the State and assign it to New State and return New State, True
  - Else return State, False
- If Control Input = 4
  - Check if the door position is in front of the current position by applying move forward input.
  - If yes, then check the State's $D == 0$ to see if the door is closed.
  - If yes, then set $D = 1$ in the State and assign it to New State and return New State, True
  - Else return State, False

The pesudo-code for the motion model function for the random environment case is as follows:

- Input: State, Control Input
- Output: New State, Flag
- If State is not feasible then return State, False
- If State is feasible then
- If Control Input = 0
  - Apply Move forward by adding the current position and the direction vector and assign to New State
  - Check if the New State X, Y is within bounds. If not return State, False
  - If New State is feasible, return New State, True
  - Else return State, False
- If Control Input = 1
  - Check the current direction and rotate the direction vector by 90 degree in the anticlockwise direction. This can be done manually as well.
  - Set the State's Θ to be the new direction and assign it to New State
  - Return New State, True
- If Control Input = 2
  - Check the current direction and rotate the direction vector by 90 degree in the clockwise direction.
  - Set the State's Θ to be the new direction and assign it to New State
  - Return New State, True
- If Control Input = 3
  - Check if the key position is in front of the current position by applying move forward input.
  - If yes, then check state's X, Y correspond to (1,1) and the State's $KP == 0$ and the State's $K == 0$ to see if we have the key
  - If yes, then set $K = 1$ in the State and assign it to New State and return New State, True
  - Else check state's X, Y correspond to (2,3) and the State's $KP == 1$ and the State's $K == 0$ to see if we have the key
  - If yes, then set $K = 1$ in the State and assign it to New State and return New State, True
  - Else check state's X, Y correspond to (1,6) and the State's $KP == 2$ and the State's $K == 0$ to see if we have the key
  - If yes, then set $K = 1$ in the State and assign it to New State and return New State, True

  - Else return State, False
- If Control Input = 4
  - Check if there is a door in front of the current position by applying move forward input.
  - If yes, then check state's X, Y correspond to (4,2) and the State's $D_1 == 0$ to see if the first door is closed.
  - If yes, then set $D_1 = 1$ in the State and assign it to New State and return New State, True
  - Else check if state's X, Y correspond to (4,5) and the State's $D_1 == 0$ to see if the second door is closed.
  - If yes, then set $D_2 = 1$ in the State and assign it to New State and return New State, True
  - Else return State, False

C. Dynamic Programming

The value function at terminal time T is set to zero if the value function corresponding to a particular state has it's X, Y corresponding to the goal position.
The algorithm is as mentioned below:

- For all the timestep i from $T-1, ...0$ do
  - For all the states x_j from the list of all states χ
  - For all the control input u_k from the control space υ do:
    - Compute Next State, Flag = Motion Model (x_j,υ_k)
    - If Flag == False, set Q(x_j,υ_k) = infinite
    - Else the below update rule is followed:

$$Q\left(x_j, \upsilon_k\right) = 1 + V_{i+1}\left(f\left(x_i, u_i\right)\right)$$

- The value function at i is then computed as:

$$V_i\left(x\right) = min\, Q\left(x, \upsilon\right)$$

- The control policy is the obtained as:

$$\pi_i\left(x\right) = argmin\, Q\left(x, \upsilon\right)$$

- If the Value function at the initial state configuration is less that infinite, then terminate
- For random environment case, terminate if all the states that had X, Y, θ corresponding to $3, 5, 0$, irrespective of other elements in that state, had value function less than infinity.

D. Sequence from Control Policy

Once the dynamic programming is executed, the control input sequence can be obtained as follows:

- Find the column in the value function matrix which has the maximum cost (excluding infinite) in the row which corresponds to the initial configuration. Set this column index to be t
- Set the start state = initial configuration state
- Create a empty list called sequence and append $\pi_t\left(x_t\right)$
- Iterate i over the time horizon from $t+1, ...T$:
- Get the next state x_i by applying motion model with x_{i-1}, $\pi_{i-1}\left(x_{i-1}\right)$
- Append sequence with $\pi_i\left(x_i\right)$
- Return sequence

IV. Discussion

Given below is a brief list of observations obtained while implementing the dynamic programming algorithm and the results:

- The state space of the MDP grows exponentially with adding new elements to the state definition. For example, moving from known environment to random environment case made the state space to be large of magnitude greater than 5 times the known environment case
- All the edge cases needed to be explicitly handled for the algorithm to give accurate results
- Although the total number of state space is very large, only a few states are actually valid. Efficiently handling then led to a significant drop in the computation time.
- Tried to vectorise the dynamic programming algorithm, however, it did not work as the value function can be computed only when the next value function is available.
- Although we could run the dynamic programming for all the timestep, the early termination condition was very significantly useful to reduce the computation time drastically.
- The termination condition for the random environment condition had to be properly handled to obtain one control policy for all the possible random maps. The termination condition for the random environment case as if all the states that had X, Y, θ corresponding to $3, 5, 0$, irrespective of other elements in that state, had a value function less than infinity.

V. Results

This section shows the starting and ending position of the agent in all the given 7 known environments in the first 14 images and then the starting and ending position of the agent in some of the given random environments. Below is the link to a google drive that has the gif generated for all the known and random environments. Link to Gif
The control sequences for the known environment are mentioned below

- 5x5 Normal : TL, TL, PK, TR, UD, MF, MF, TR, MF
- 6x6 Normal : TL, MF, PK, TL, MF, TL, MF, TR, UD, MF, MF, TR, MF
- 8x8 Normal : TR, MF, TL, MF, TR, MF, MF, MF, PK, TL, TL, MF, MF, MF, TR, UD, MF, MF, MF, TR, MF, MF, MF
- 6x6 Direct : MF, MF, TR, MF, MF
- 8x8 Direct : MF, TL, MF, MF, MF, TL, MF
- 6x6 Shortcut : PK, TL, TL, UD, MF, MF
- 8x8 Shortcut : TR, MF, TR, PK, TL, UD, MF, MF

The control sequences for the known environment are mentioned below

- Map 1 : MF, MF, MF, TR, MF, MF, TL, MF

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

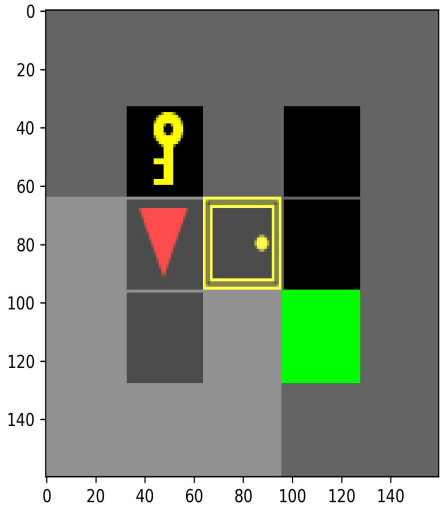

Fig. 1. Doorkey 5x5 Normal Start

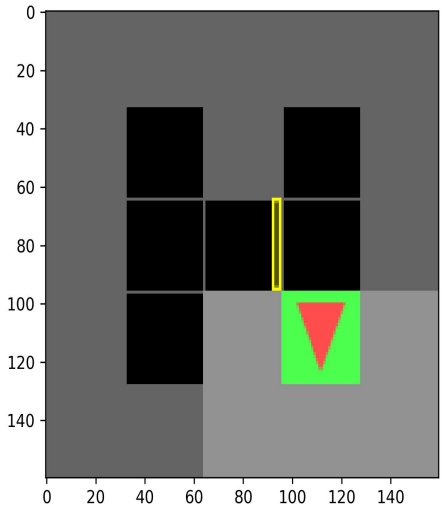

Fig. 2. Doorkey 5x5 Normal End

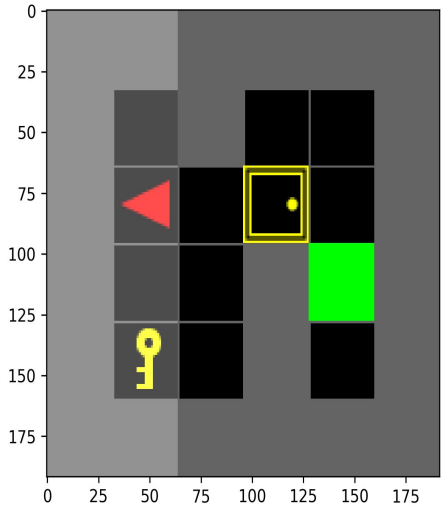

Fig. 3. Doorkey 6x6 Normal Start

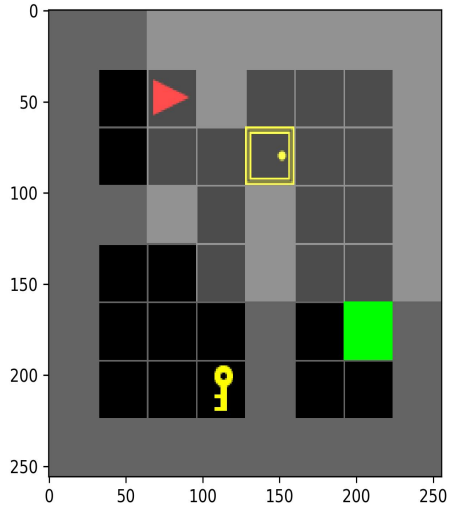

Fig. 5. Doorkey 8x8 Normal Start

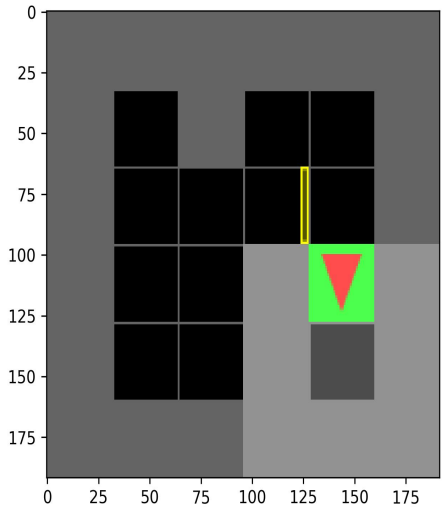

Fig. 4. Doorkey 6x6 Normal End

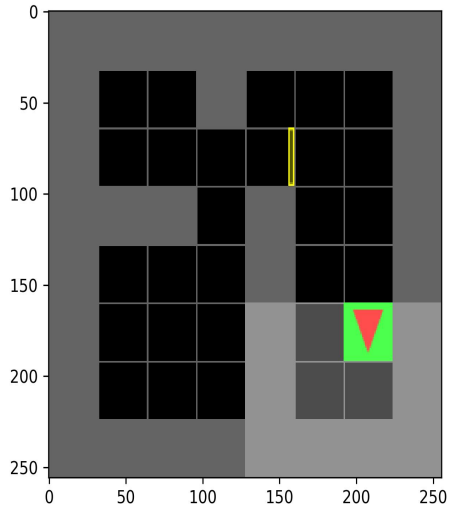

Fig. 6. Doorkey 8x8 Normal End

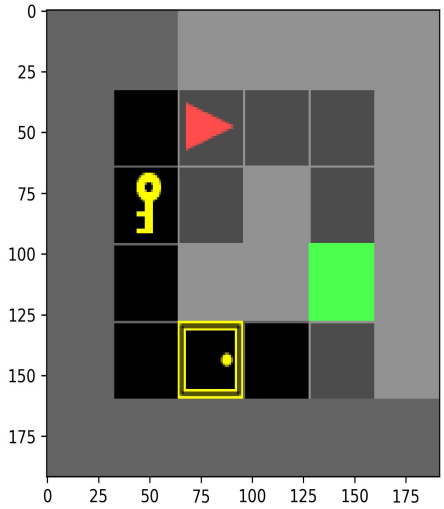

Fig. 7. Doorkey 6x6 Direct Start

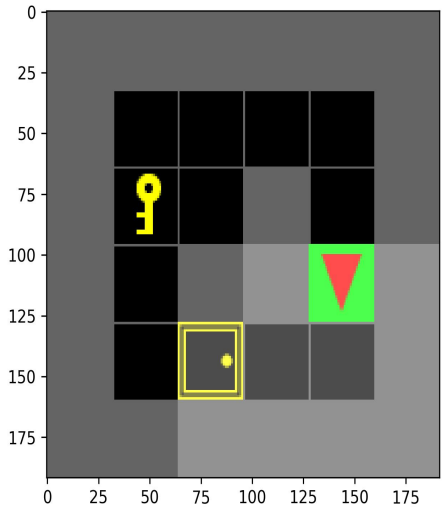

Fig. 8. Doorkey 6x6 Direct End

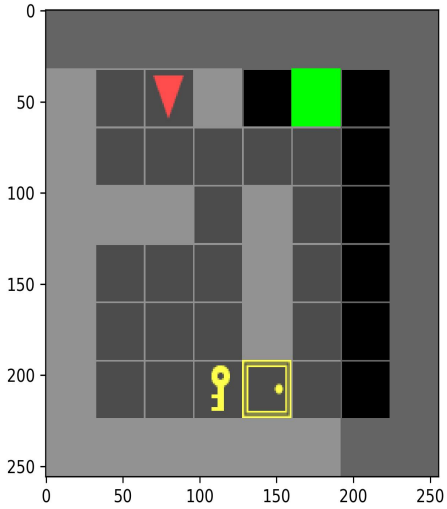

Fig. 9. Doorkey 8x8 Direct Start

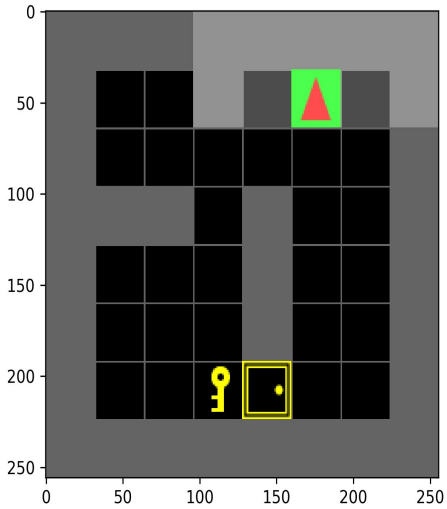

Fig. 10. Doorkey 8x8 Direct End

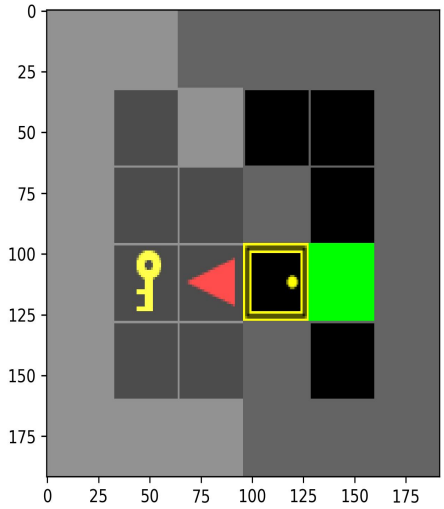

Fig. 11. Doorkey 6x6 Shortcut Start

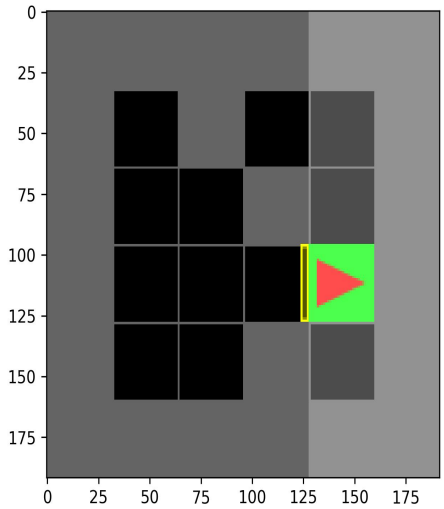

Fig. 12. Doorkey 6x6 Shortcut End

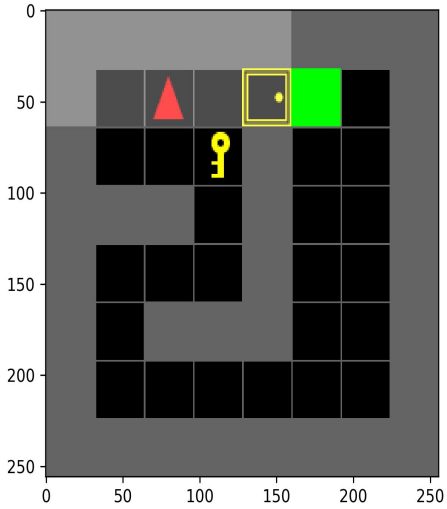

Fig. 13. Doorkey 8x8 Shortcut Start

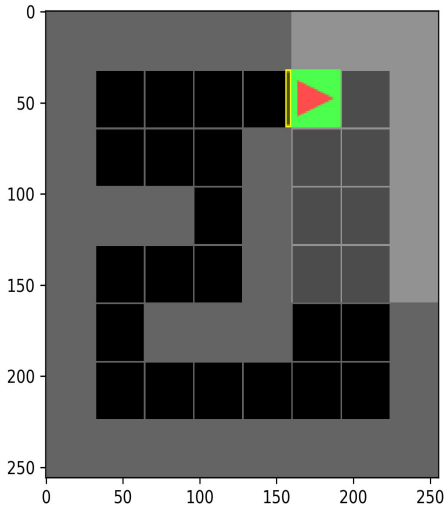

Fig. 14. Doorkey 8x8 Shortcut End

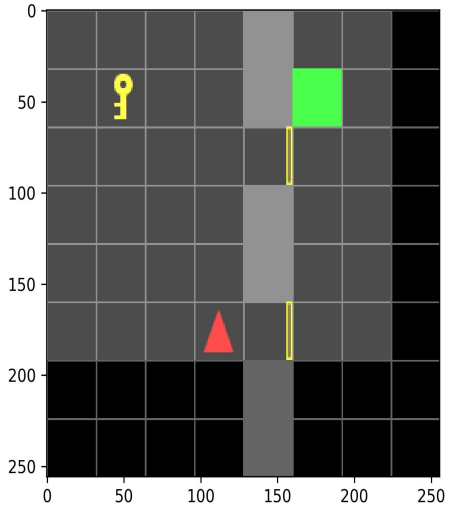

Fig. 15. Random Doorkey 8x8 1 Start

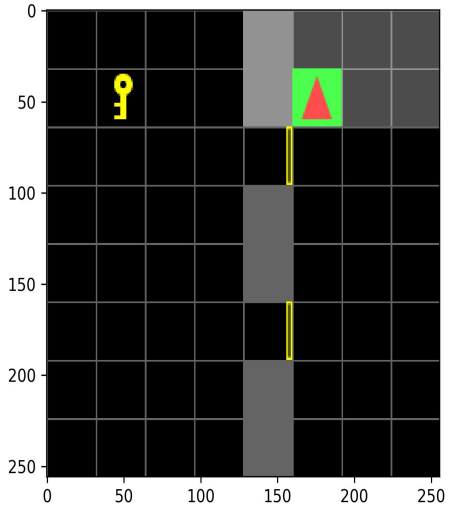

Fig. 16. Random Doorkey 8x8 1 End

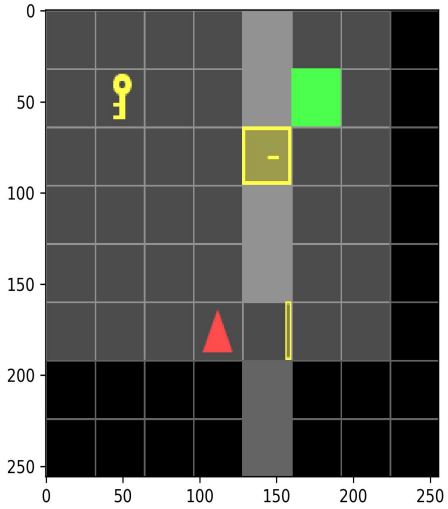

Fig. 17. Random Doorkey 8x8 3 Start

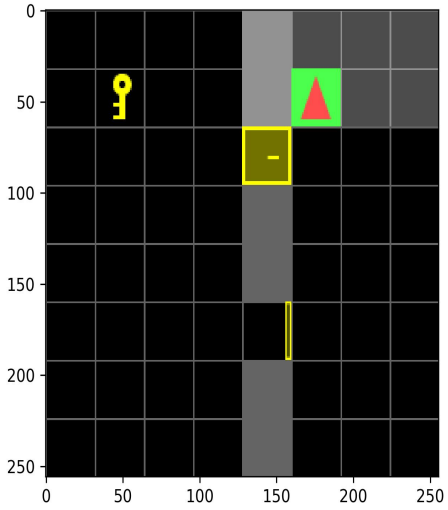

Fig. 18. Random Doorkey 8x8 3 End

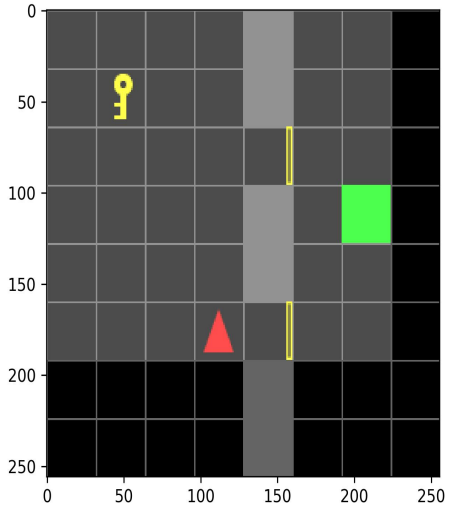

Fig. 19. Random Doorkey 8x8 5 Start

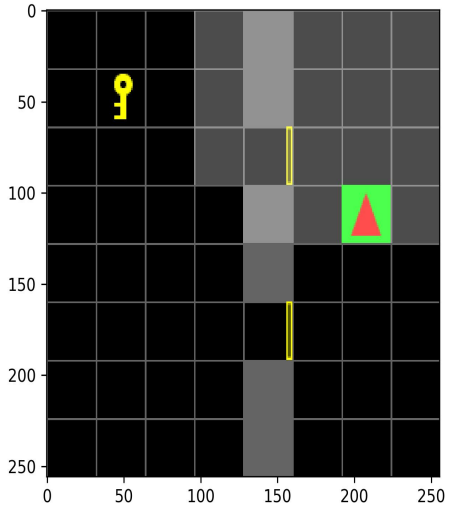

Fig. 20. Random Doorkey 8x8 5 End

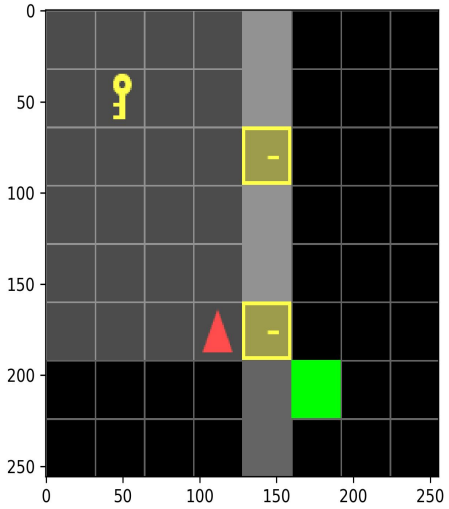

Fig. 21. Random Doorkey 8x8 12 Start

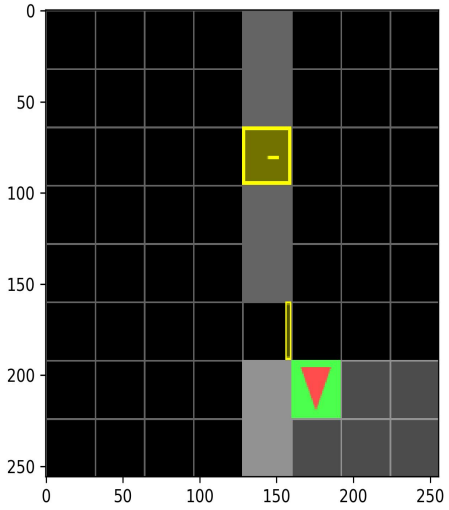

Fig. 22. Random Doorkey 8x8 12 End

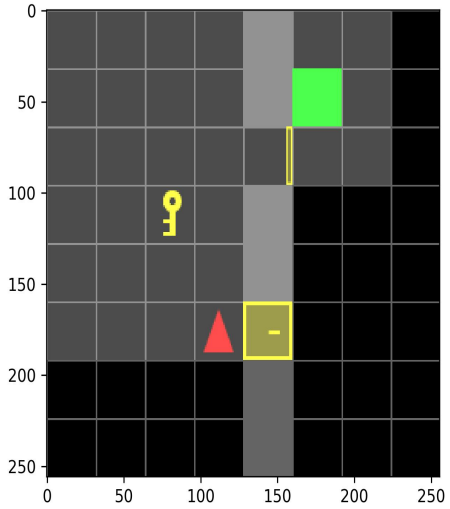

Fig. 23. Random Doorkey 8x8 14 Start

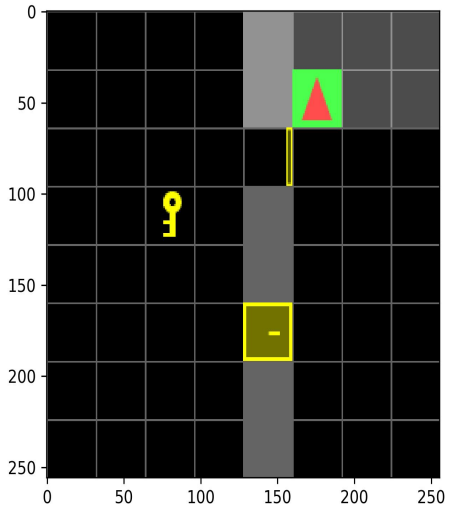

Fig. 24. Random Doorkey 8x8 14 End

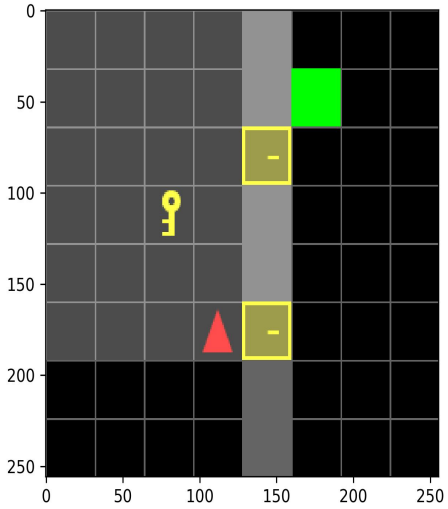

Fig. 25. Random Doorkey 8x8 16 Start

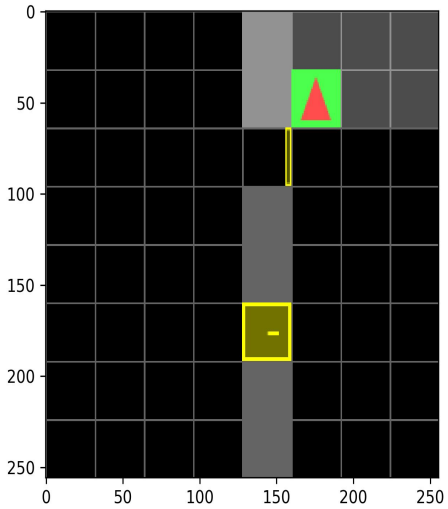

Fig. 26. Random Doorkey 8x8 16 End

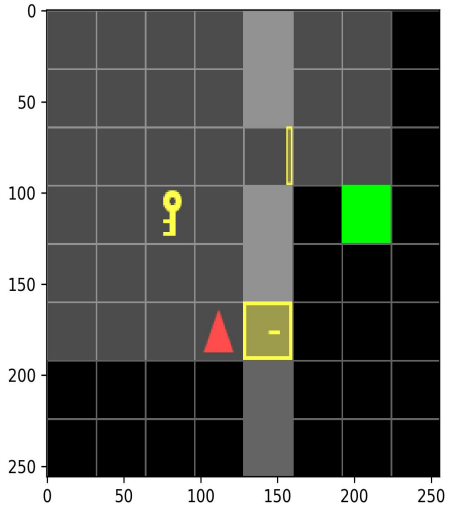

Fig. 27. Random Doorkey 8x8 18 Start

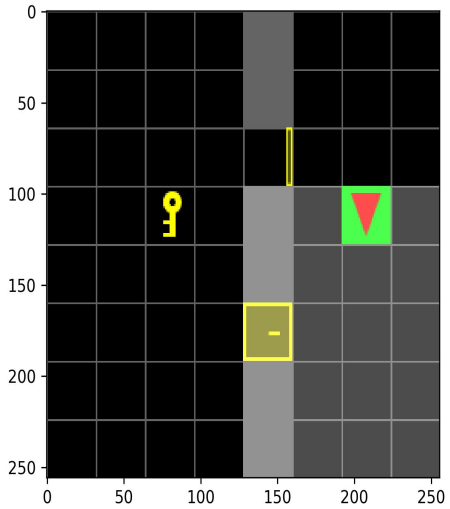

Fig. 28. Random Doorkey 8x8 18 End

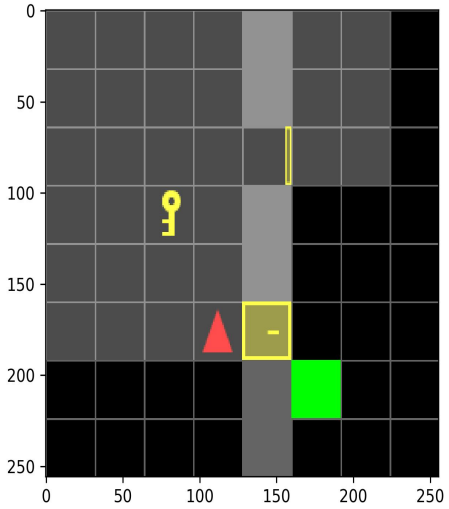

Fig. 29. Random Doorkey 8x8 22 Start

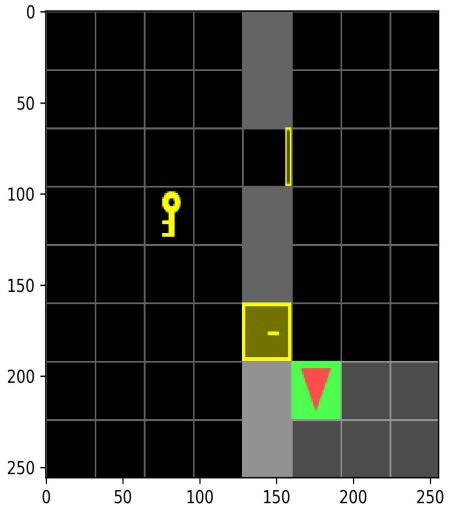

Fig. 30. Random Doorkey 8x8 22 End

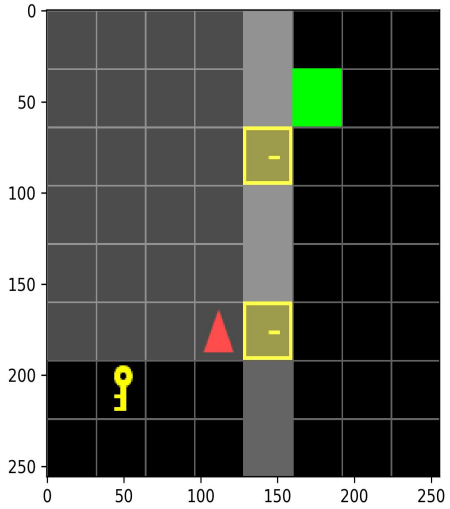

Fig. 31. Random Doorkey 8x8 28 Start

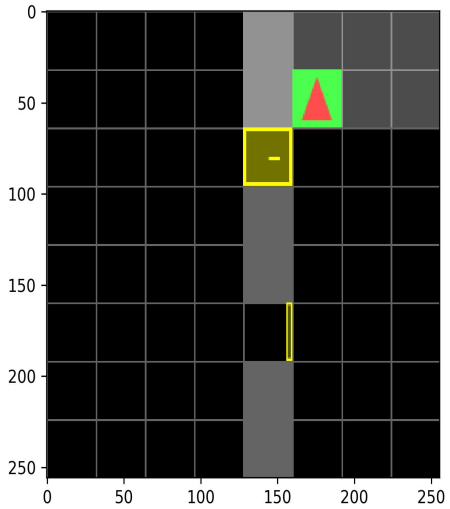

Fig. 32. Random Doorkey 8x8 28 End

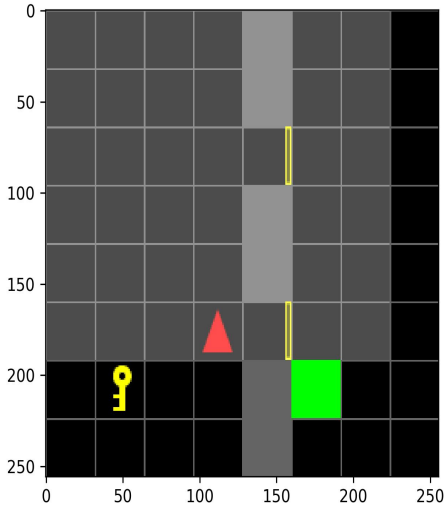

Fig. 33. Random Doorkey 8x8 33 Start

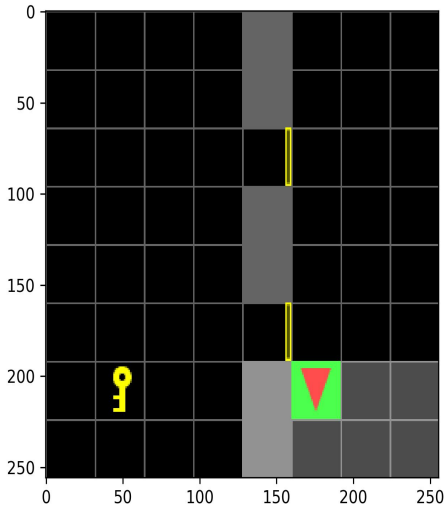

Fig. 34. Random Doorkey 8x8 33 End

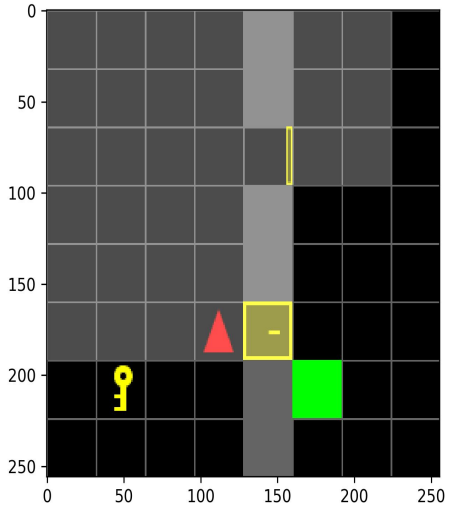

Fig. 35. Random Doorkey 8x8 34 Start

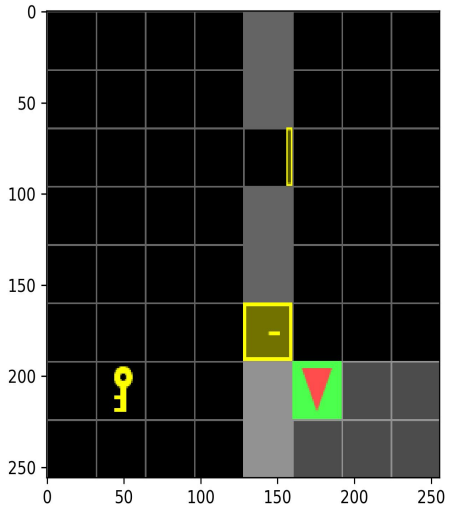

Fig. 36. Random Doorkey 8x8 34 End

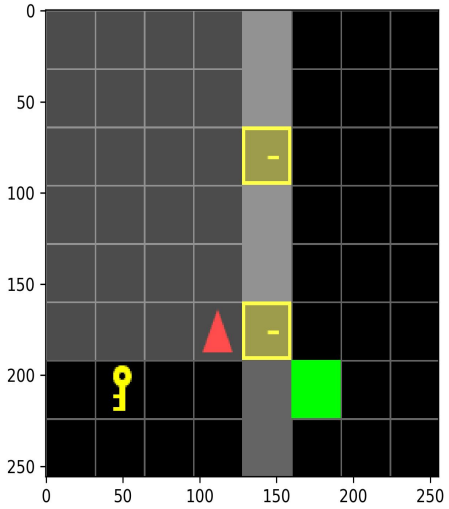

Fig. 37. Random Doorkey 8x8 36 Start

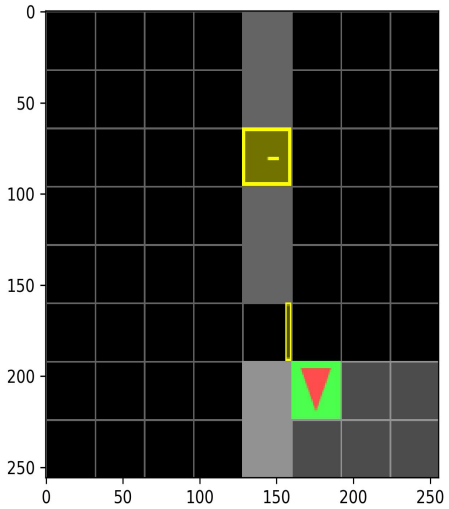

Fig. 38. Random Doorkey 8x8 36 End