Markov Decision Process

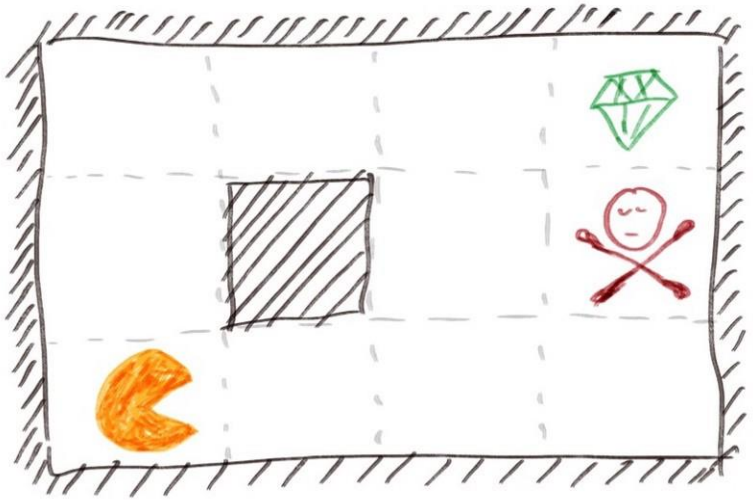
Dr. Venkataramana Veeramsetty

Definition

- *A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov decision process, or MDP*
- *It consists*
  - *Set of states (with an initial state s_0)*
  - *Set ACTIONS(s) of actions in each state*
  - *Transition model P(s' | s, a)*
  - *Reward function R(s).*

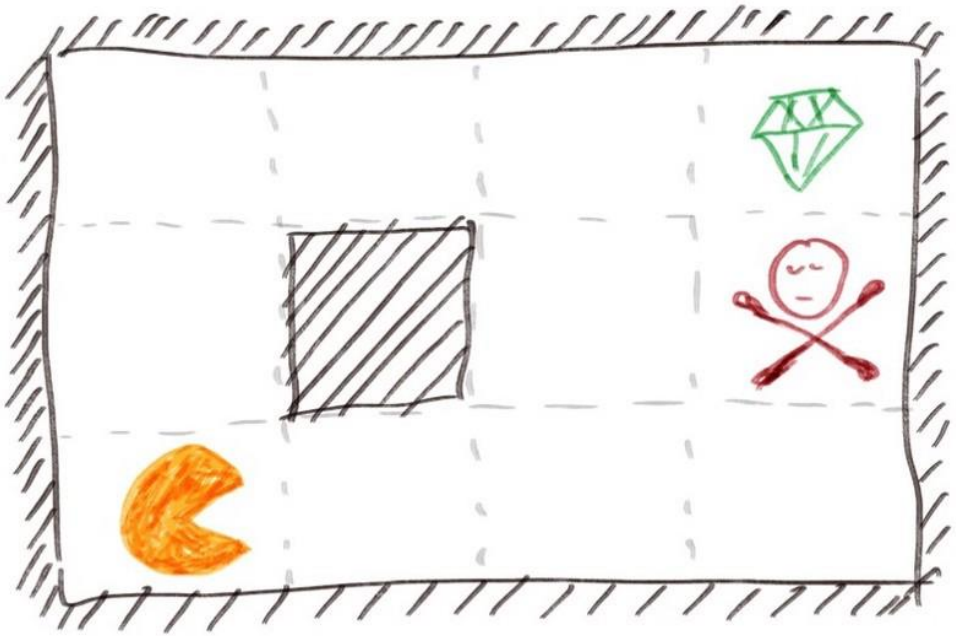
Grid World Problem

- Our robot can move in four directions: up, down, left, and right
- There are also non-passable walls inside the boundary represented by the black square
- The green diamond in the square at the upper right corner represents a finish line. If we reach this square, we win this game and earn a lot of points (+1 in this example).
- A square with a red poison. If we step into this square, we lose the game and incur a lot of penalty (-1 in this example)
- All the other white squares are passable squares. Each time we step into one of them, we lose a small amount of points (-0.04 in this example).



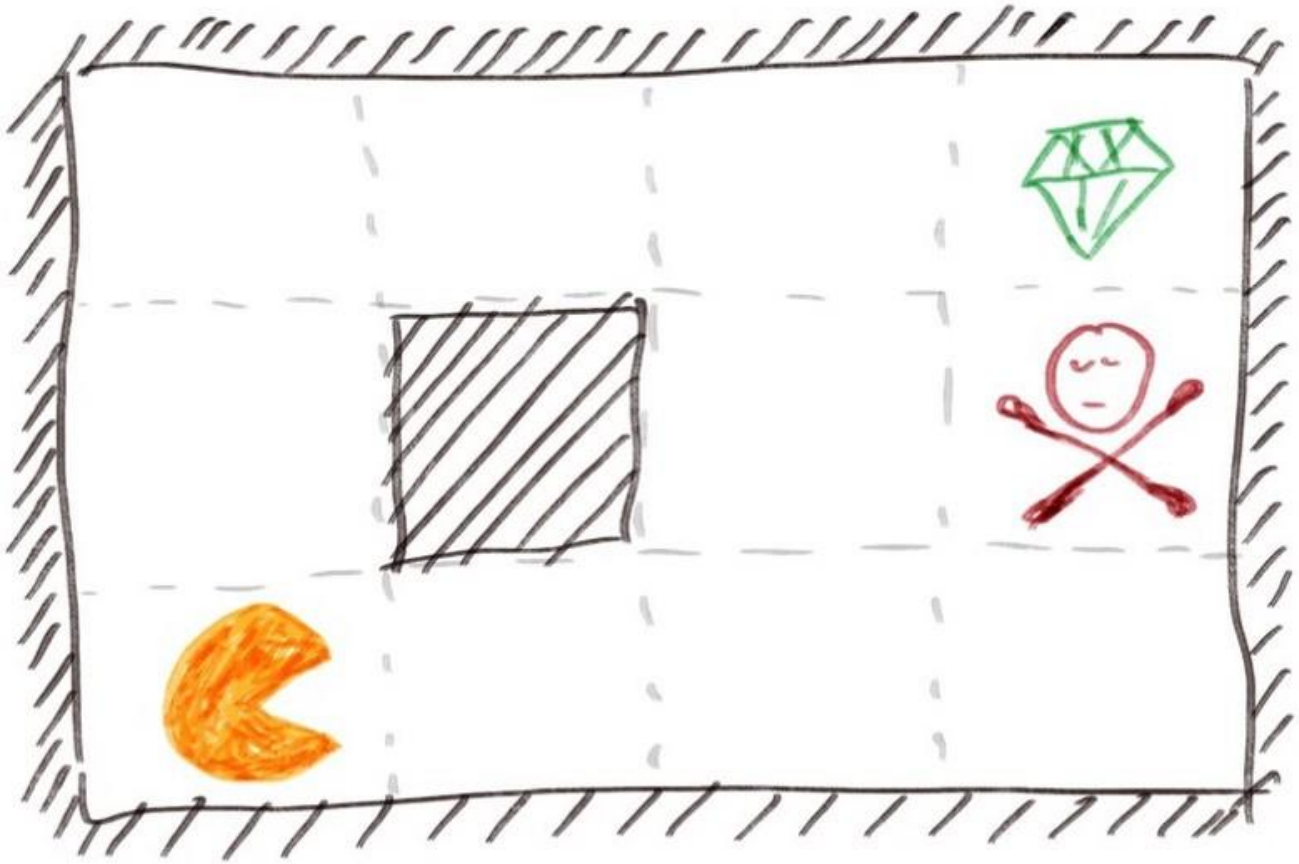
Agent

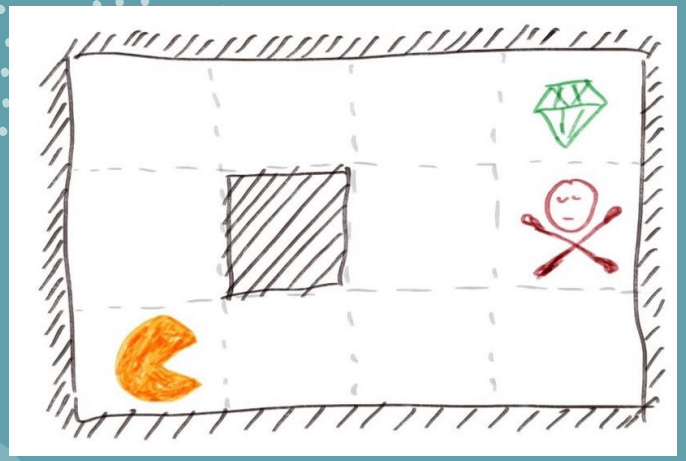
- ***Agent*** refers to the robot whatever we put inside this world that looks around for possible moves, thinks about the pros and cons of possible moves, makes a decision of the next move, and executes the move.



Environment

- ***Environment*** refers to what the world looks like and the rules of the world.
- In our example, environment includes how big our world is, where the walls are placed, what happens if we bump into a wall, where is the diamond and if we reach there how much points we get, where is the poison and if we reach there how much points we lose, and so on.





State

State in MDP refers to the state of our agent, not state of our environment.

In MDP the environment is given and does not change.

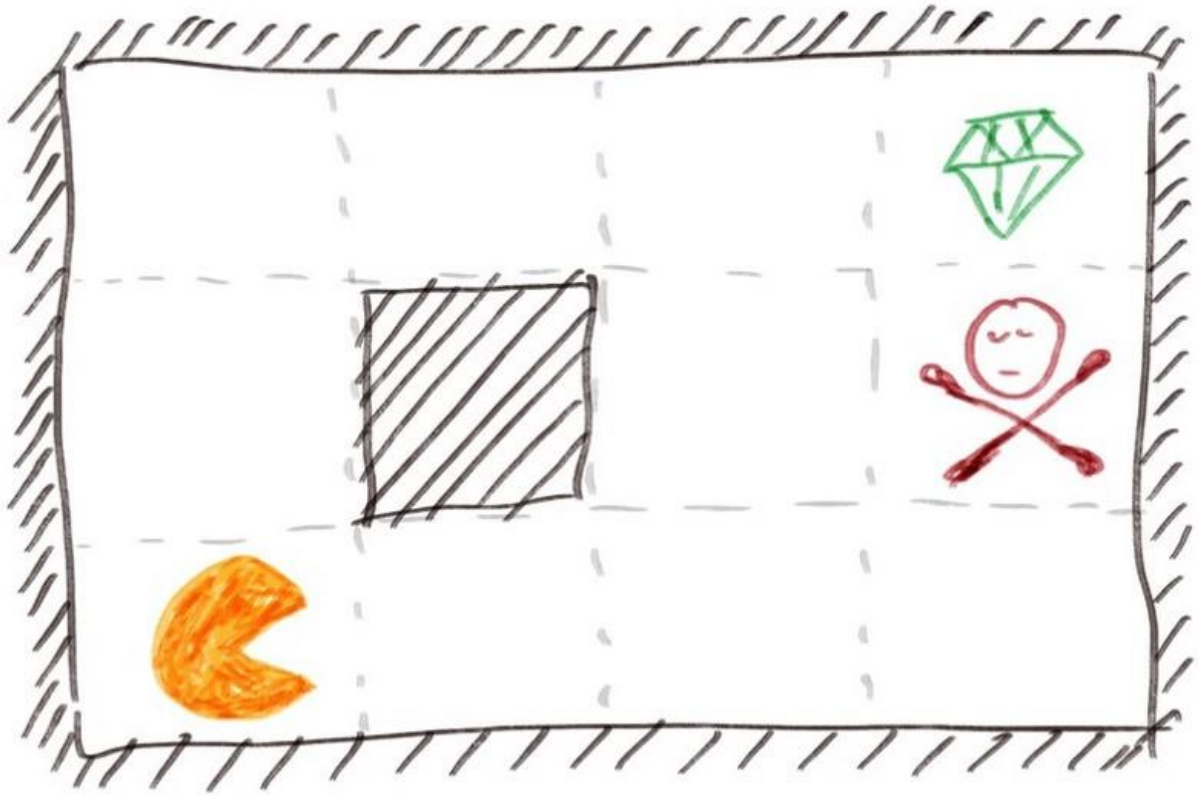
In contrast, our agent can change state from previous move to current move.

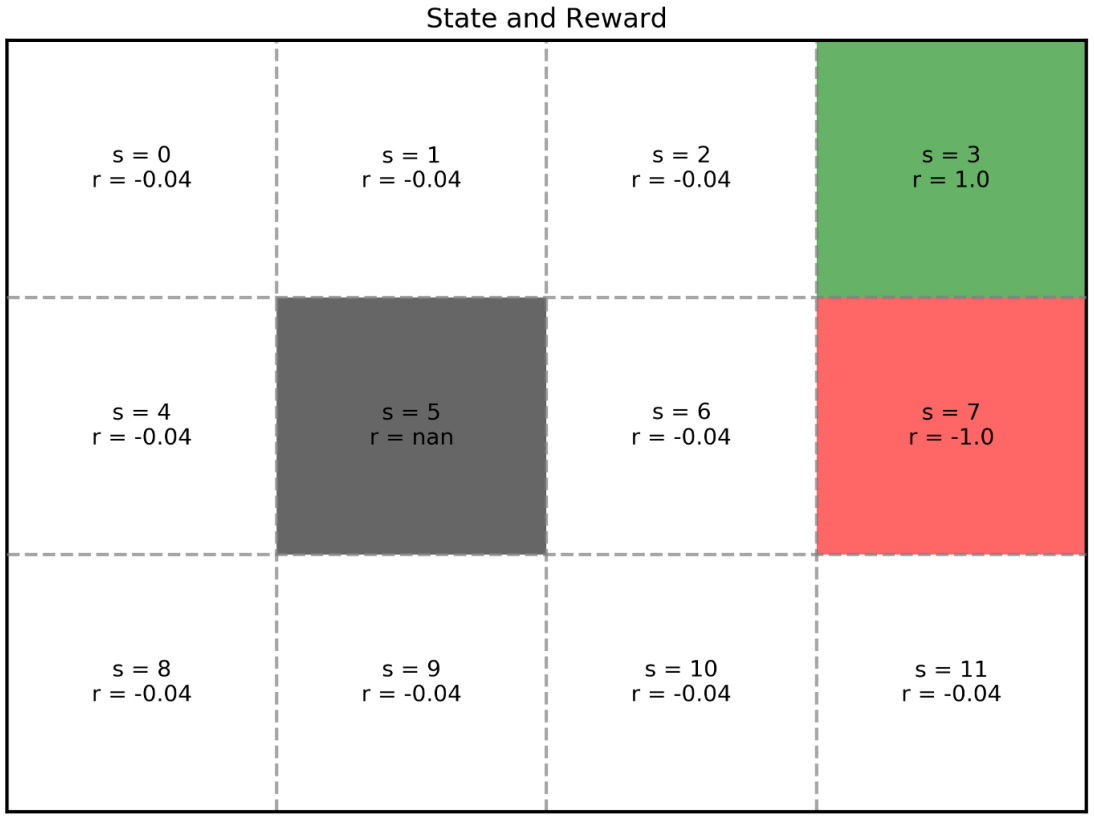
In our example, our agent has 12 states which represent the 12 squares that our agent can possibly be in.

In MDP we use *s* to denote state and in our example we call them *s* = 0, 1, 2, …, 11.

Action

- ***Action*** refers to the moves can be taken by our agent in each state.
- The set of possible actions is not necessarily the same for each state.
- In our example, we have four possible actions $A(s)$ = {up, down, left, and right} for every state s.
- We use a to denote actions.





Reward

- ***Reward*** refers to the points agent receive when agent move into a state.
- In our example, that is +1, -1, or -0.04 for different states.
- Reward is dependent on state only and this function from state to reward is denoted as $R(s)$.
- We receive the same reward when entering a certain state regardless of our previous state.
- For instance, when $s = 1$, $r = R(s) = R(1) = -0.04$. No matter we move right into it from $s = 0$ or we move left into it from $s = 2$, we receive the same -0.04 reward.

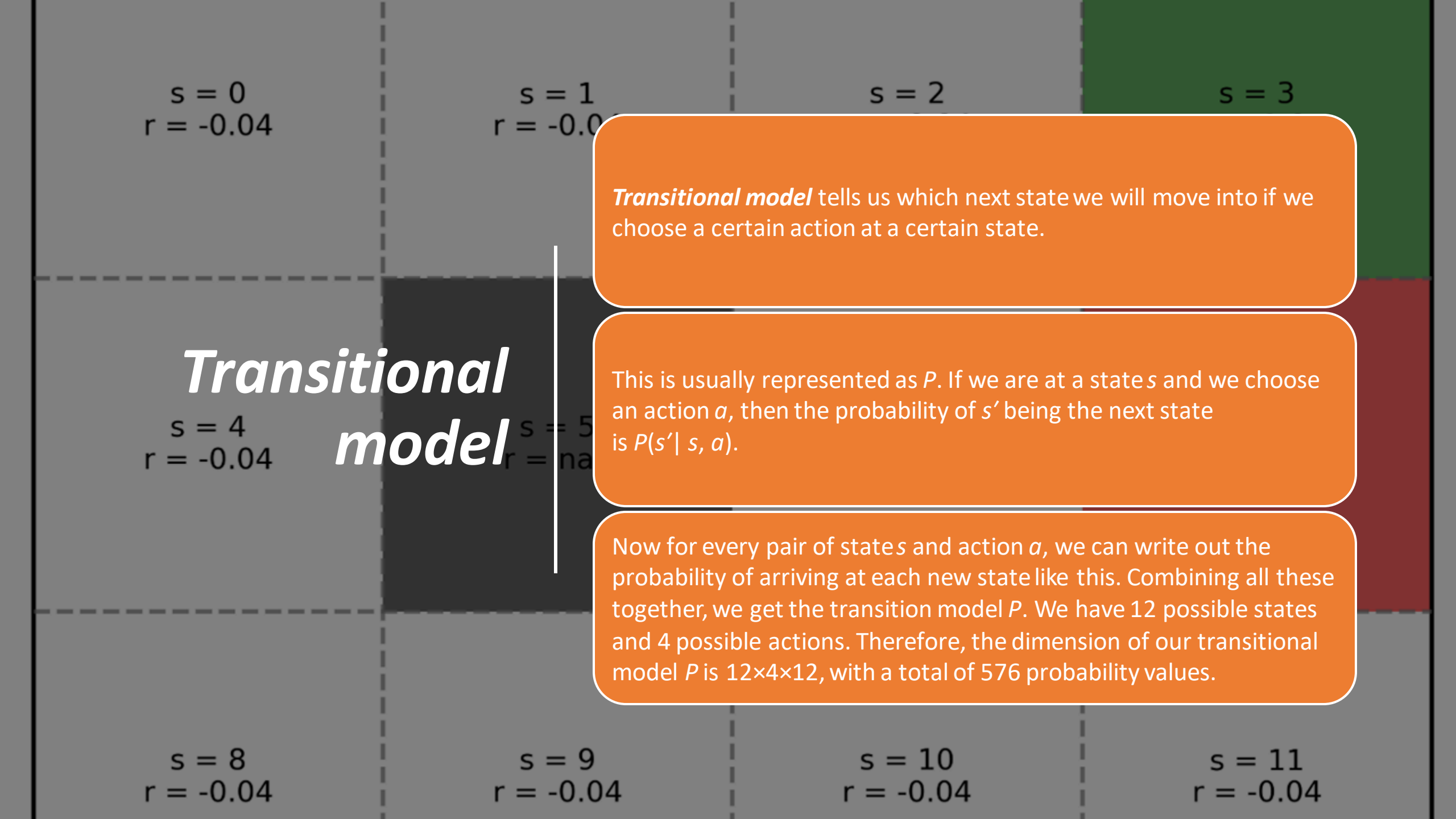
Additive rewards

Additive rewards mean the rewards from different moves are cumulative.

In our example, our final points (total rewards) are the points received from the diamond or the poison and the points received from passable squares along the way added together.

Not only the reward received in the final square but also all the small rewards received along the way matter.

In MDP, we assume we can *add* these rewards together



Transitional model

Transitional model tells us which next state we will move into if we choose a certain action at a certain state.

This is usually represented as *P*. If we are at a state *s* and we choose an action *a*, then the probability of *s'* being the next state is $P(s' \mid s, a)$.

Now for every pair of state *s* and action *a*, we can write out the probability of arriving at each new state like this. Combining all these together, we get the transition model *P*. We have 12 possible states and 4 possible actions. Therefore, the dimension of our transitional model *P* is 12×4×12, with a total of 576 probability values.

Fully observable

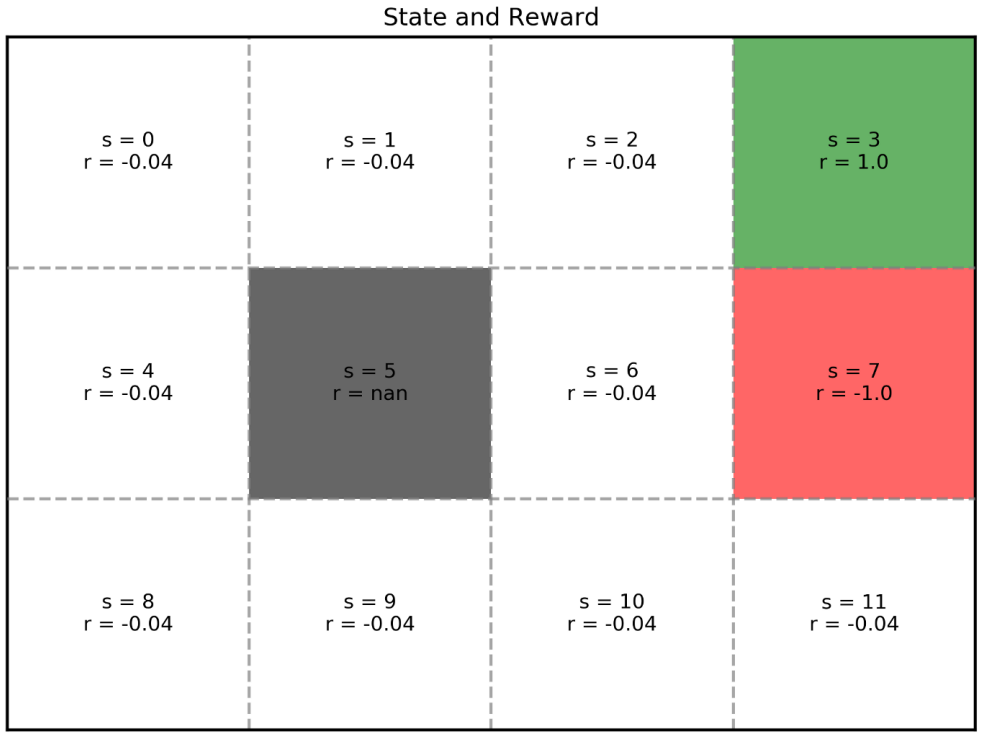
Fully observable means our agent somehow knows what the world looks like and where itself currently is in that world.

It has access to all the knowledge needed to find the optimal decision during each move.

The knowledge here refers to our reward function $R(s)$ and transitional model $P(s' | s, a)$.

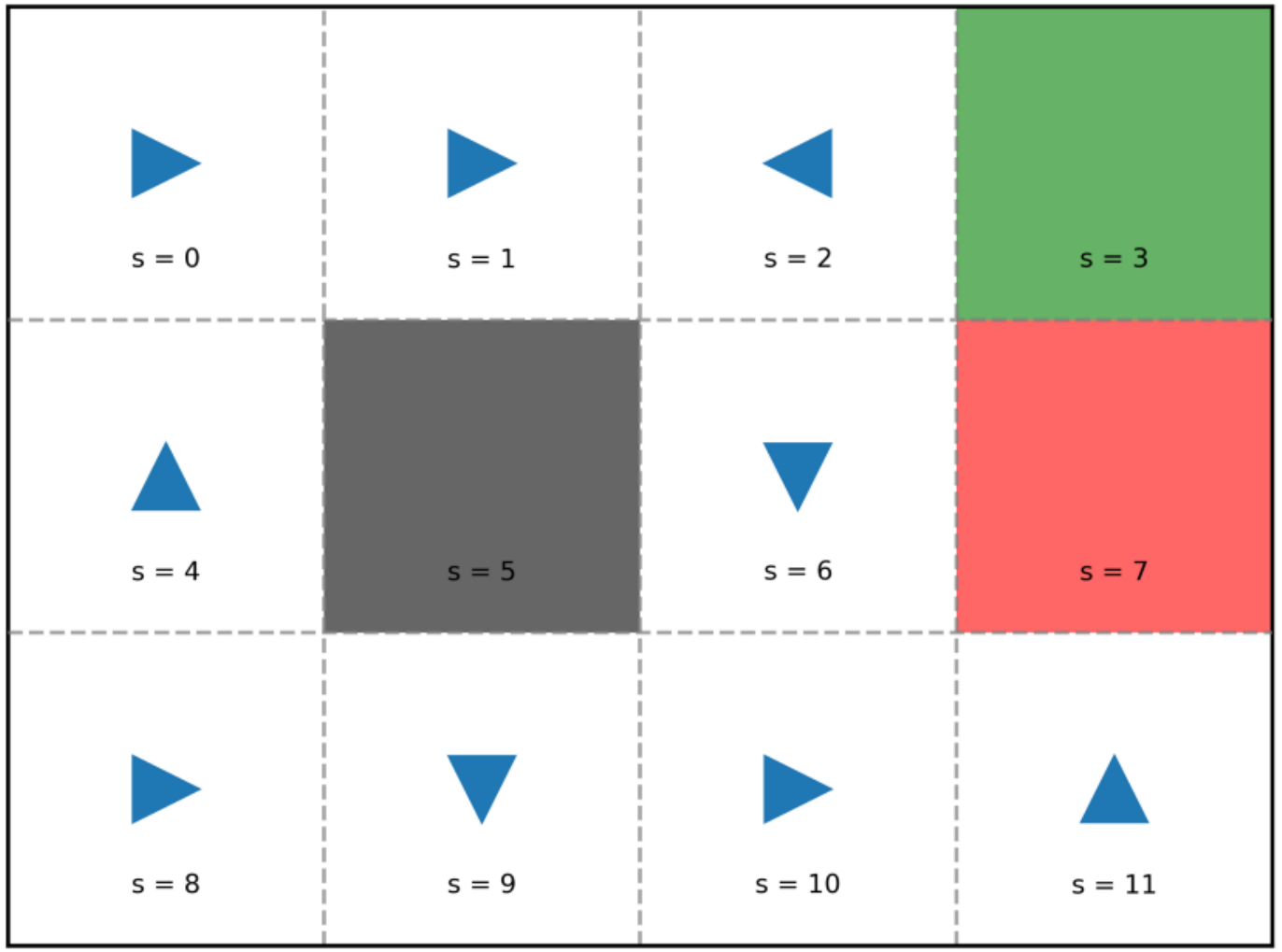
Sequential

- ***Sequential*** means our current situation is affected by previous decisions.
- By the same token, all future situations are affected by our current decision. .
- For instance, if we start from $s = 8$ and decide to move up as our first move, then we arrive at $s = 4$. There is no way we can reach $s = 10$ in the next move.
- However, if we choose move right as our first move from $s = 8$, then we arrive at $s = 9$. Now we can reach $s = 10$ by moving right.
- In other words, whether we can reach $s = 10$ in the second move depend on what we choose in the first move.



Policy

- Policy refers to the instructions of what action to be taken in each state. It is usually denoted as π and it is a function of *s*.
- For instance, if *s* = 2 and π(*s*) = π(2) = RIGHT, then whenever out agent is in state *s* = 2, the policy says it should choose the action to go right.
- Whether it can actually go to the square on the right is another story which depends on the probability in transition model.
- π(0) = RIGHT, π(1) = RIGHT, π(2) = LEFT, … π(11) = UP.



For Deterministic World

- As a probability distribution, the sum of $P(s' | s = 10, a = \text{UP})$ of all 12 s' always equals to 1.

$$P(s' = 0|s = 10, a = \text{UP}) = 0$$
$$P(s' = 1|s = 10, a = \text{UP}) = 0$$
$$P(s' = 2|s = 10, a = \text{UP}) = 0$$
$$\dots$$
$$P(s' = 6|s = 10, a = \text{UP}) = 1$$
$$P(s' = 7|s = 10, a = \text{UP}) = 0$$
$$P(s' = 8|s = 10, a = \text{UP}) = 0$$
$$\dots$$
$$P(s' = 9|s = 10, a = \text{UP}) = 0$$
$$P(s' = 10|s = 10, a = \text{UP}) = 0$$
$$P(s' = 11|s = 10, a = \text{UP}) = 0$$

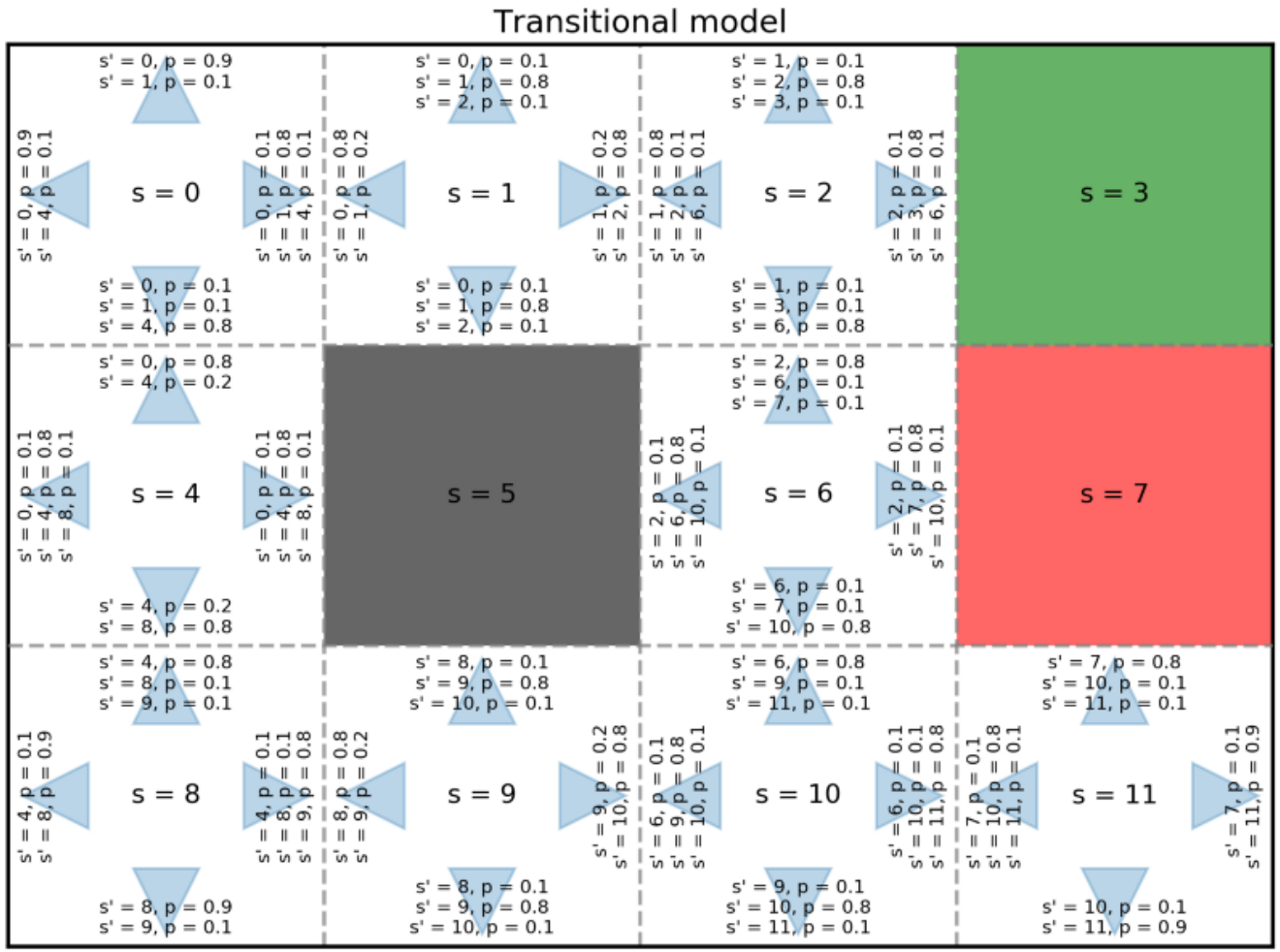
For Stochastic World

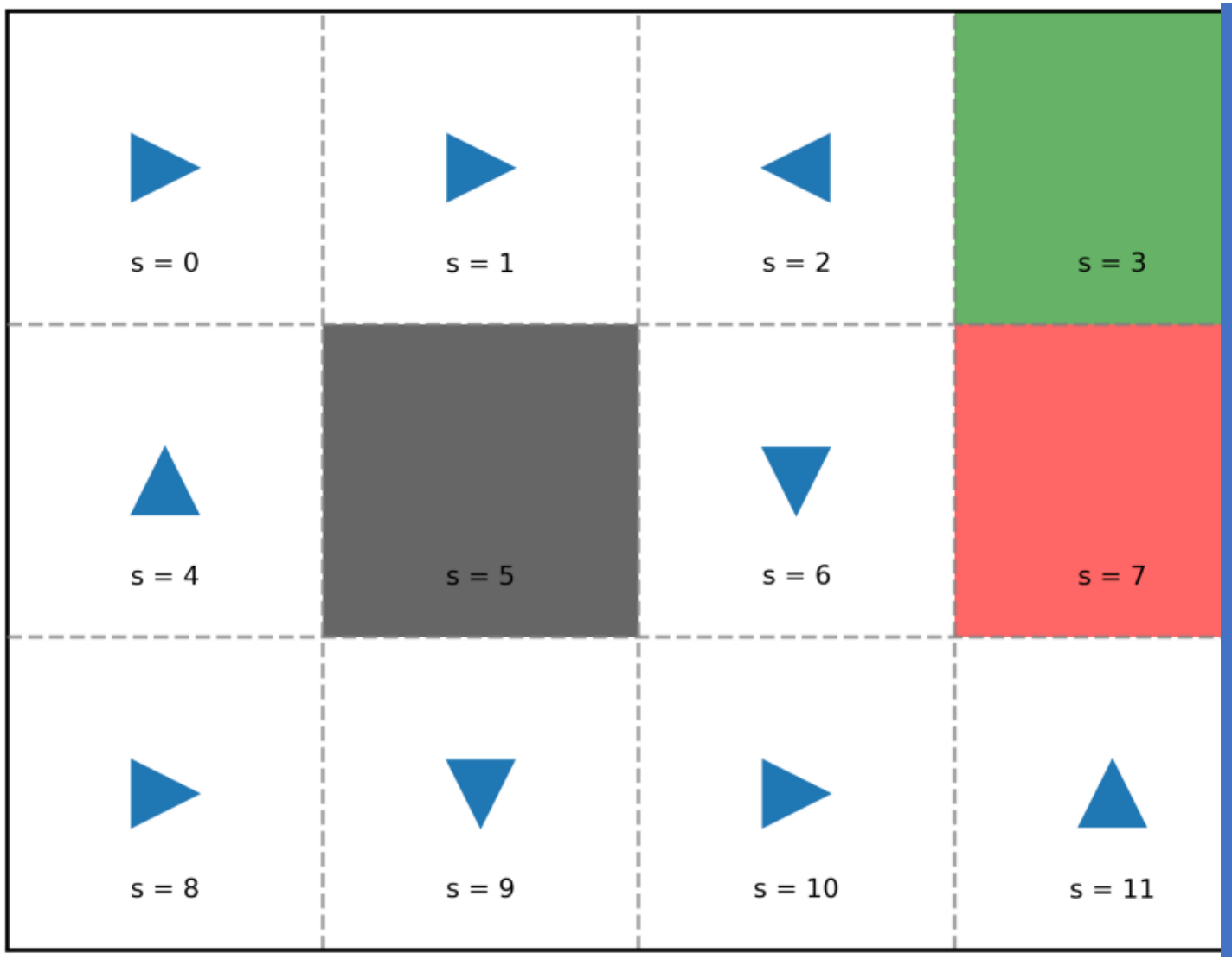
- From a certain state, if we choose the same action, we are not guaranteed to move into the same next state.
- In our game, we can think of this as our robot somehow has some probability of malfunctioning. For instance, if it decides to go left, it has a high possibility to actually go left. Nevertheless, there is a small possibility, no matter how tiny it may be, that it goes wild and moves into directions other than left.

$$P(s' = 0|s = 10, a = \text{UP}) = 0$$
$$P(s' = 1|s = 10, a = \text{UP}) = 0$$
$$P(s' = 2|s = 10, a = \text{UP}) = 0$$
$$\dots$$
$$P(s' = 6|s = 10, a = \text{UP}) = 0.8$$
$$P(s' = 7|s = 10, a = \text{UP}) = 0$$
$$P(s' = 8|s = 10, a = \text{UP}) = 0$$
$$\dots$$
$$P(s' = 9|s = 10, a = \text{UP}) = 0.1$$
$$P(s' = 10|s = 10, a = \text{UP}) = 0$$
$$P(s' = 11|s = 10, a = \text{UP}) = 0.1$$

Transitional model

- If we bump into a wall, we are bounced back and stay where we were.
- If we want to move up or down, we have 0.8 probability to actually go up or down, 0.1 probability to go left, and 0.1 probability to go right.
- If we want to move left or right, we have 0.8 probability to actually go left or right, 0.1 probability to go up, and 0.1 probability to go down.



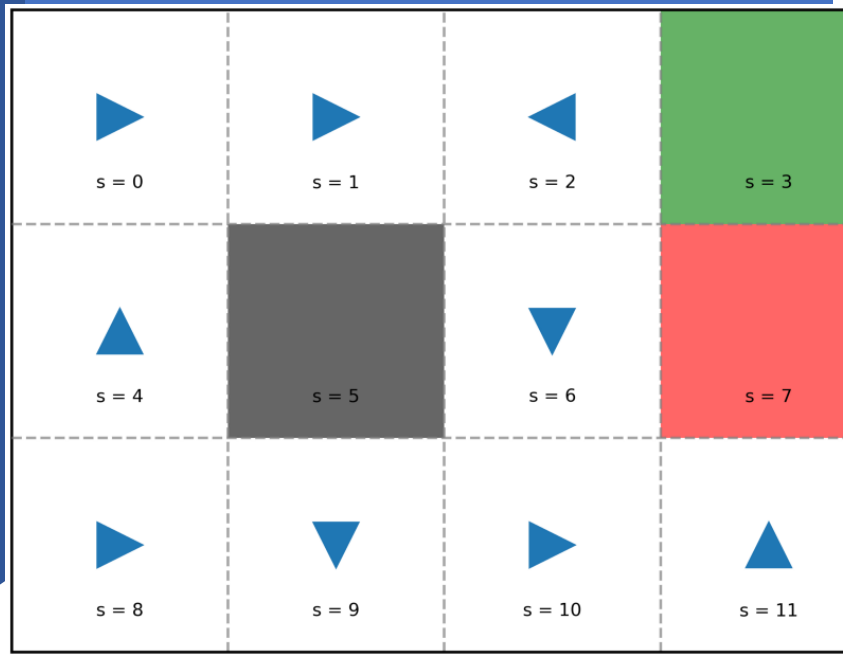


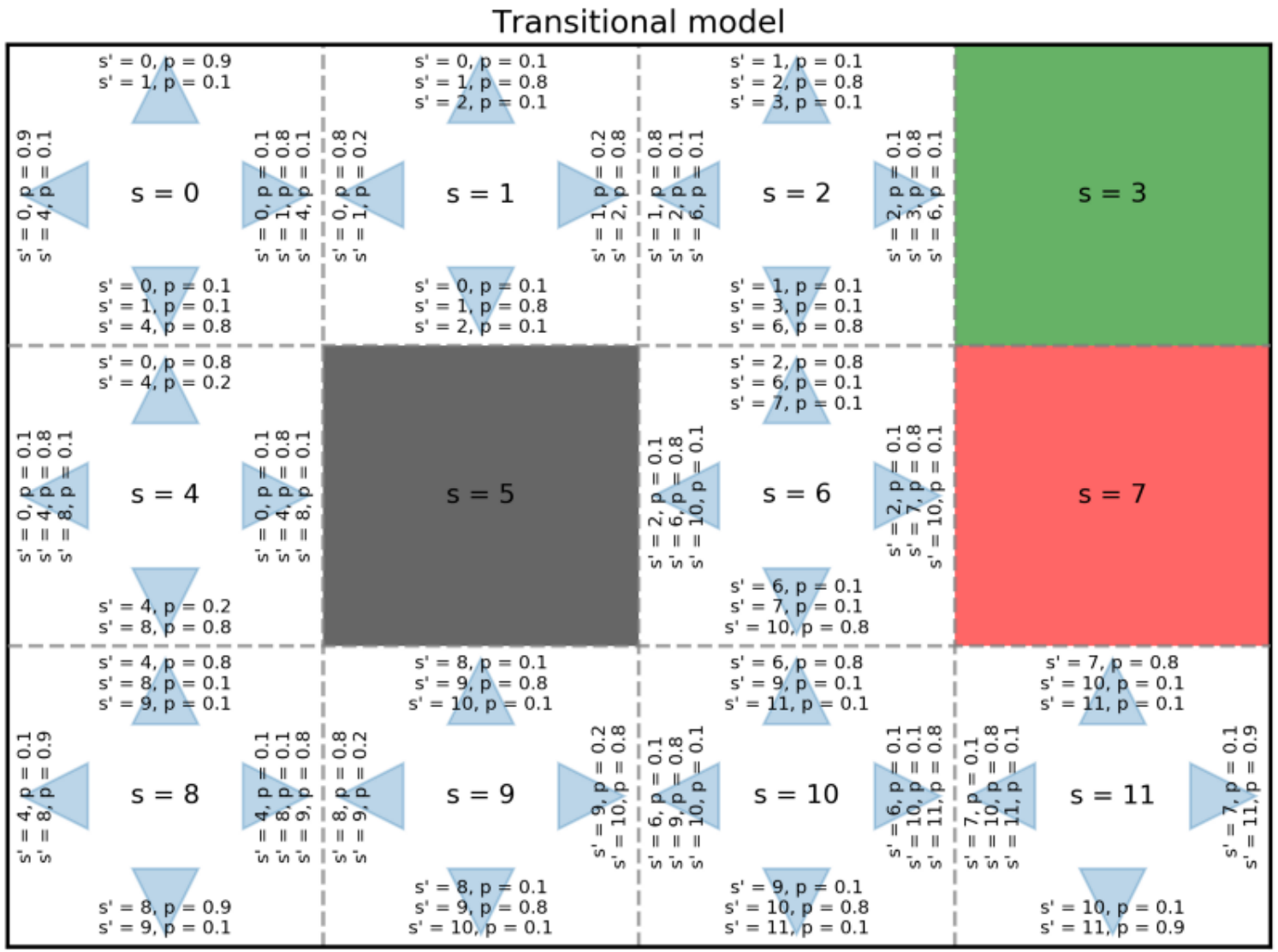
Case Study – Random Policy (In Deterministic World) #1

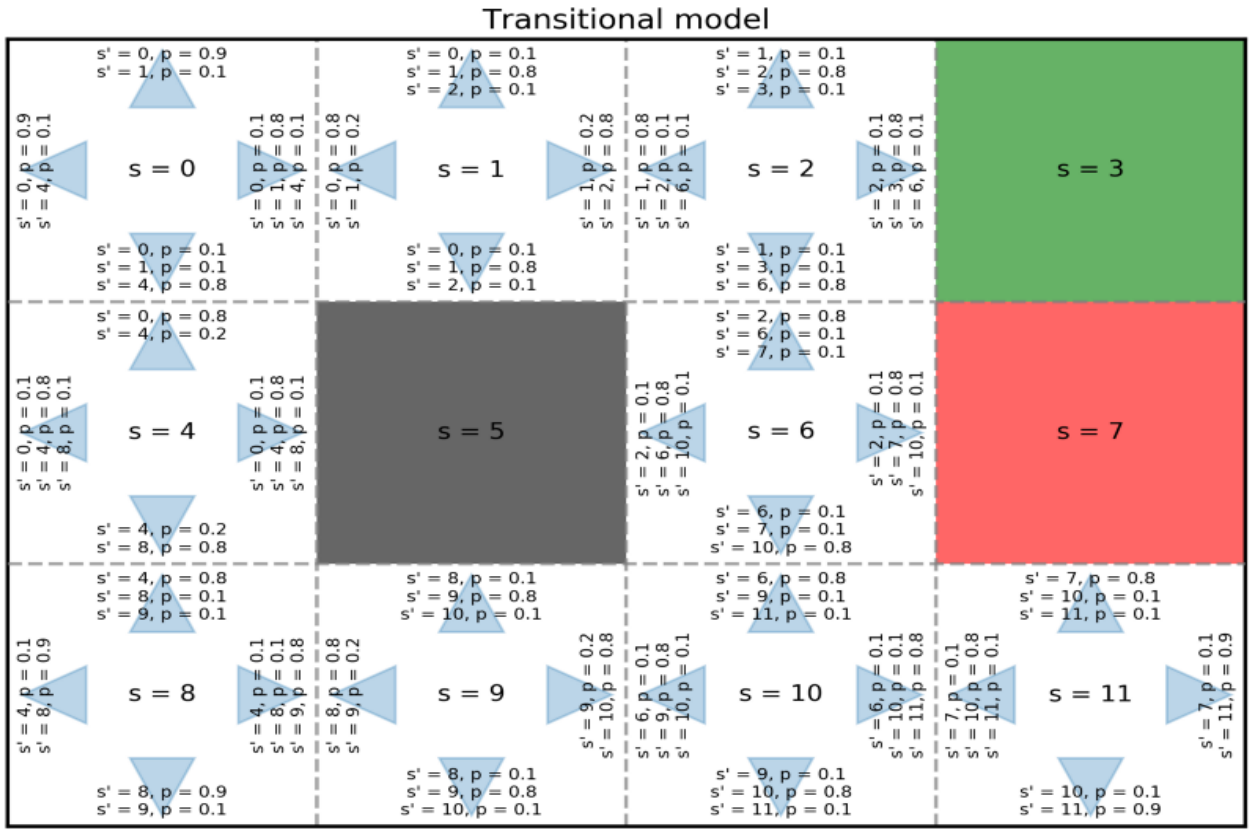
- For instance, we start at $s = 8$, policy says to go right; we move right into $s = 9$, then policy says go down. However, the bottom of $s = 9$ is a wall. We just hit that wall and bounce back to $s = 9$. It seems we stuck in $s = 9$.
- The reason we won't be trapped in s = 9 forever is that we are in a *stochastic* world. Even the policy tells us to go down and we follow that policy, there is still a probability of 0.1 that we end up in $s = 8$ and a probability of 0.1 that we end up in $s = 10$.

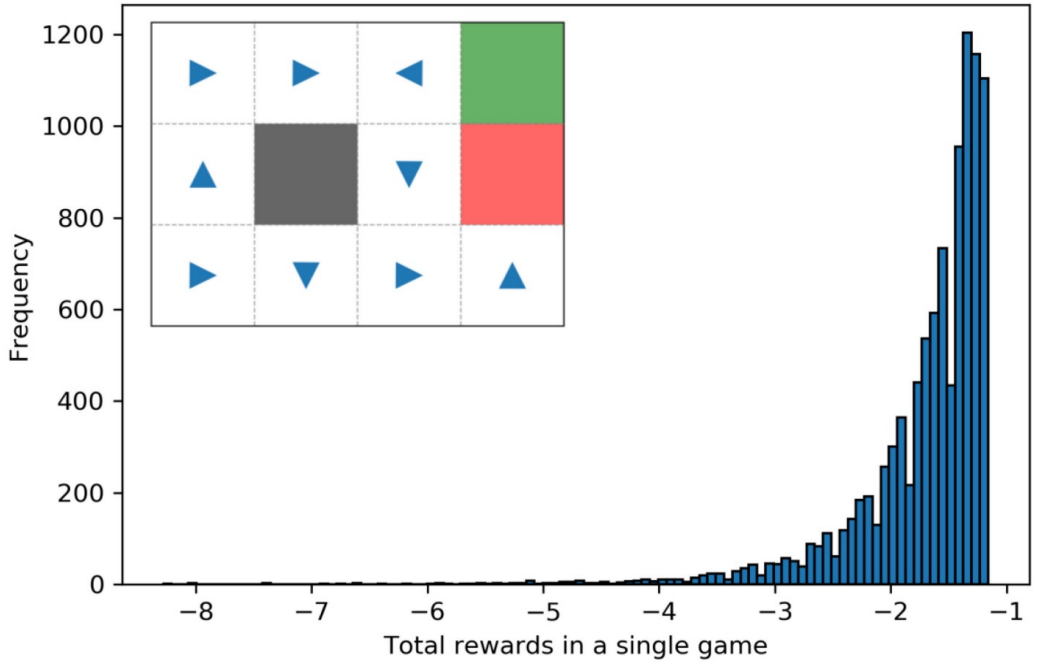
Execute a policy In Stochastic Environment

The history of states of this run is [8, 9, 9, 9, 9, 10, 10, 6, 10, 11, 7].



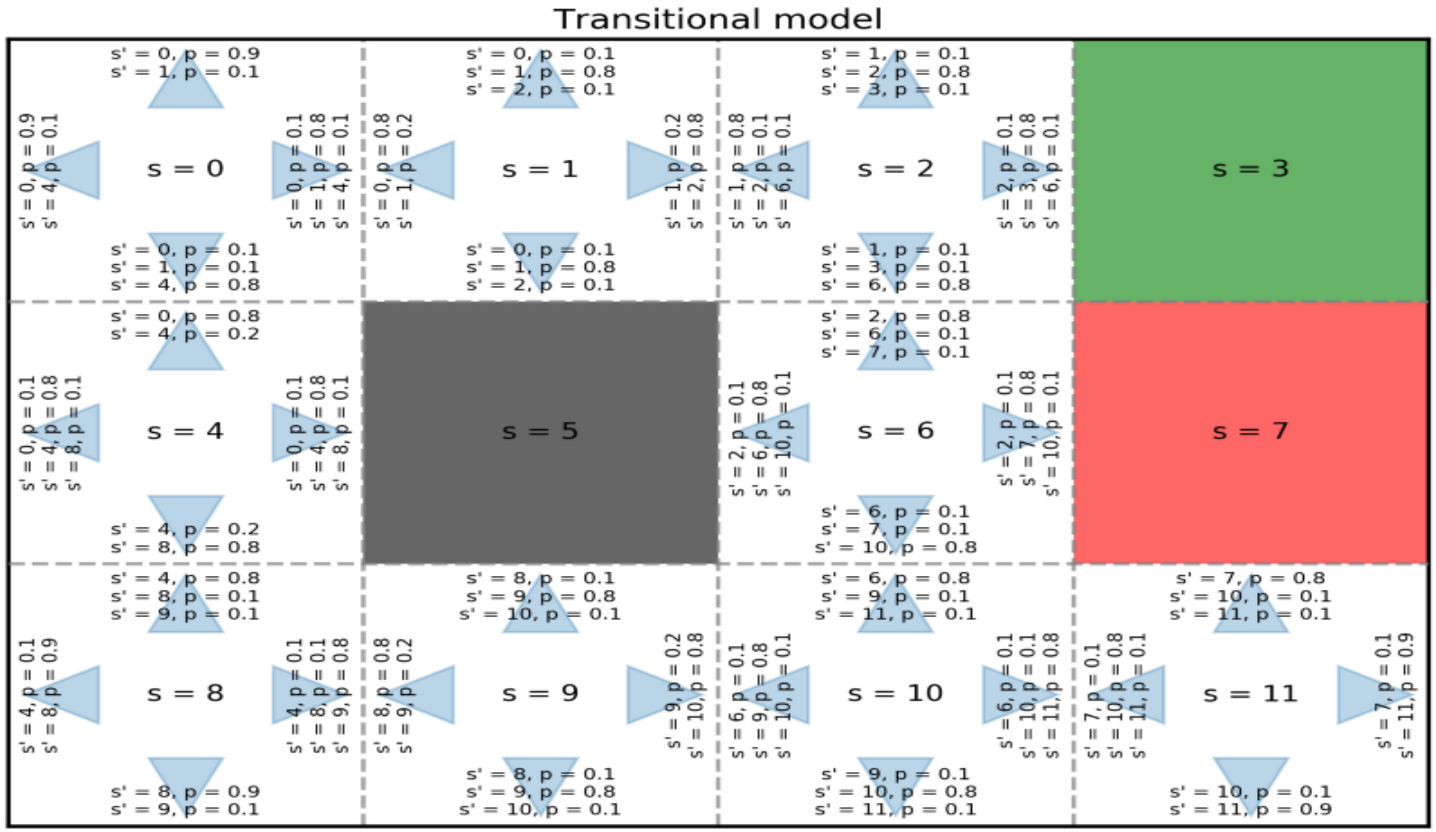


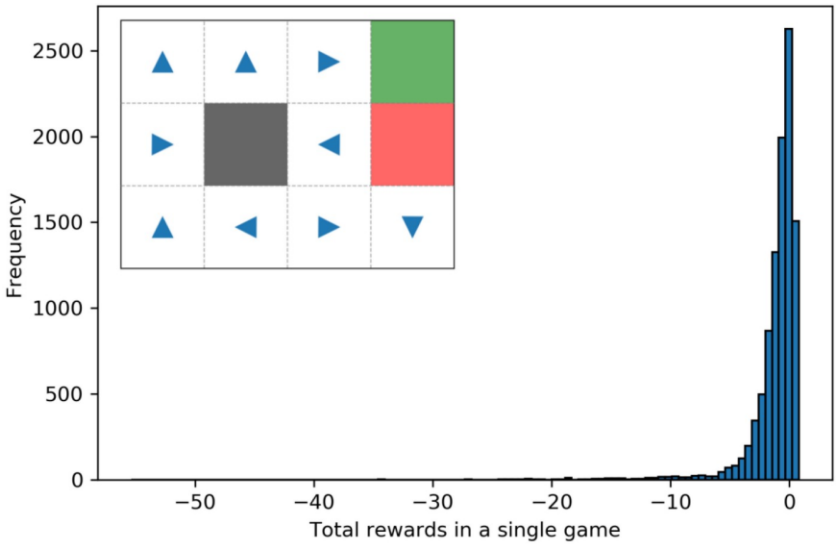




Monte Carlo approach (Policy #1)

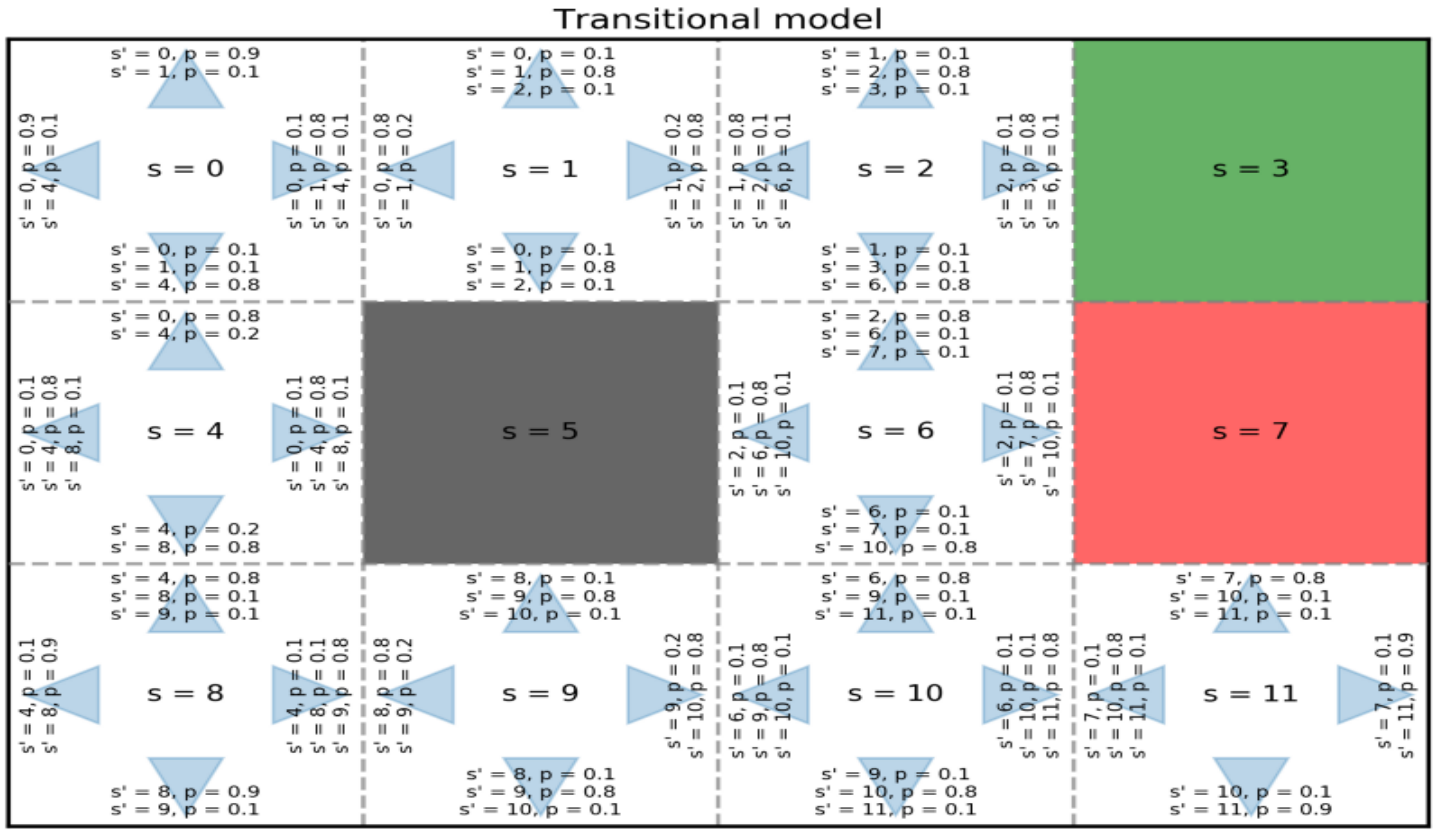
- let's just repeatedly execute the policy for 10,000 times
- The highest total rewards we got is -1.16, and the lowest is around -8.5. On average, we achieved -1.7 per game.

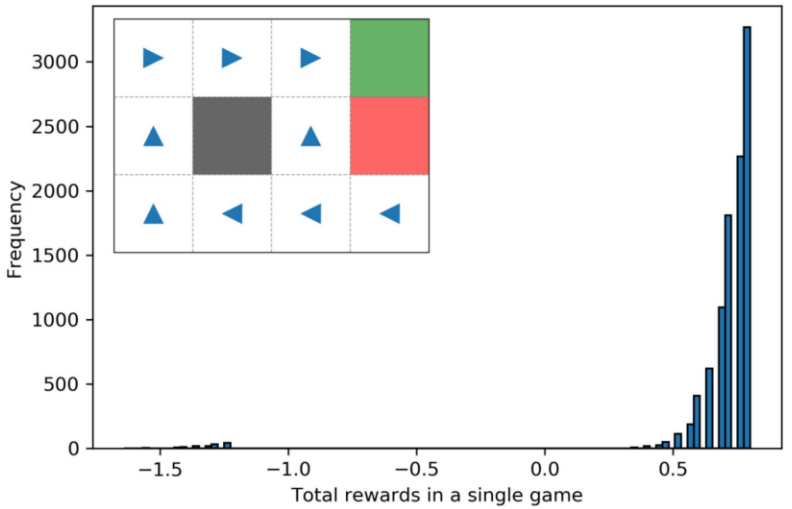




Monte Carlo approach (Policy #2)

- The highest is 0.8 which in fact is as good as we can get. The average is -1.2 which is better than the previous random policy #1. However, the lowest is -55 which is way worse than -8.5 of random policy #1.





Monte Carlo approach (Policy #3)

- This one seems pretty good. The highest is 0.8, the lowest is -1.64, and the average is 0.7. Overall, this one seems much better than the previous two.

How to find an optimal policy?

Looking back at our three random policies, we can say that #3 seems to be the best one. Is there another one that is even better? If so, how do we find it?

In our example, we have 9 states needing an assigned action and we have 4 possible actions for each state. That is a total of 4^9 policies.

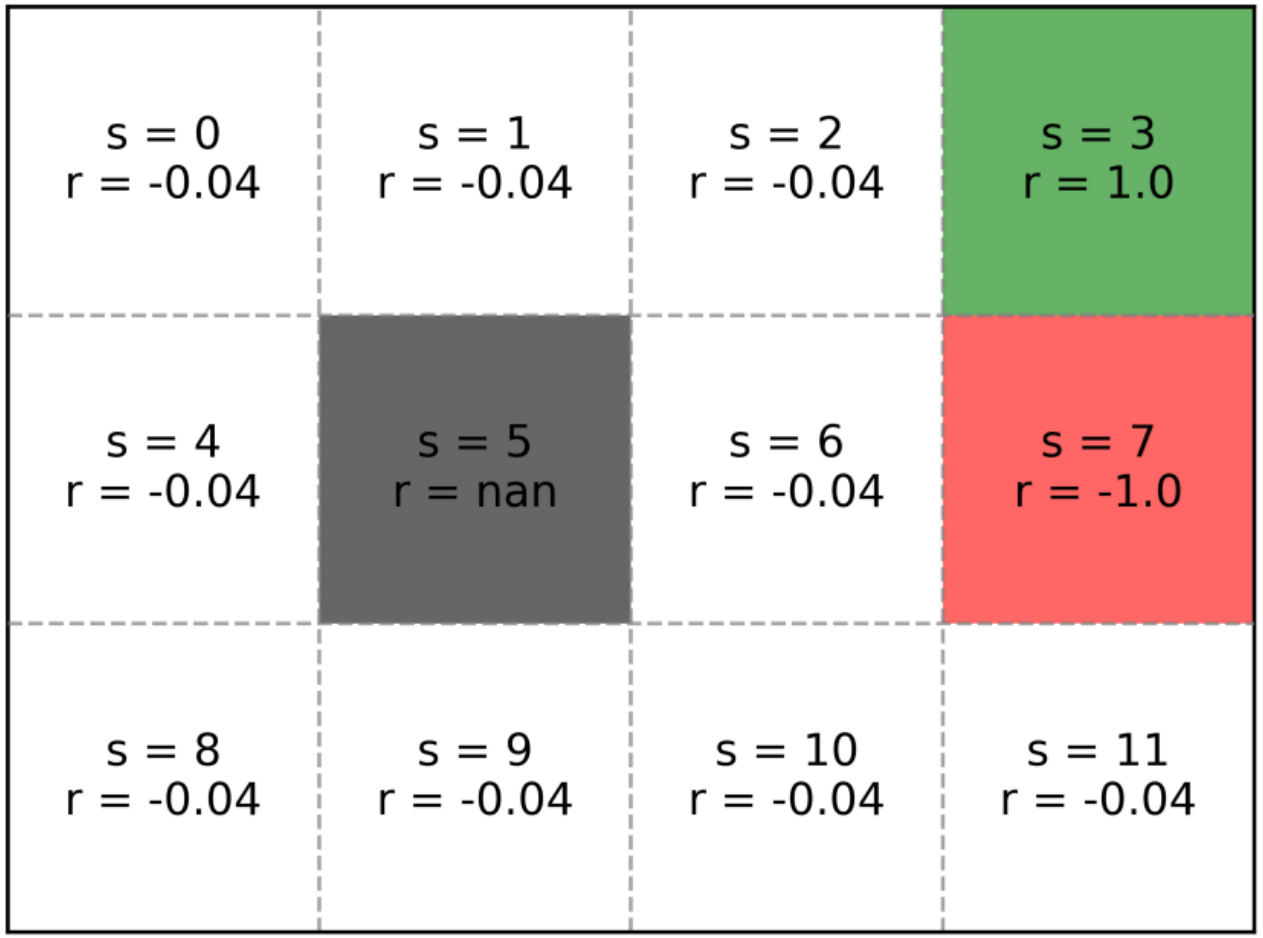
One which have higher expected total reward than the others. Such policy is therefore an optimum policy π^*. Obviously, this is not practical at all

In practice, we have two methods: *policy iteration* and *value iteration*.

Markov decision process: policy iteration

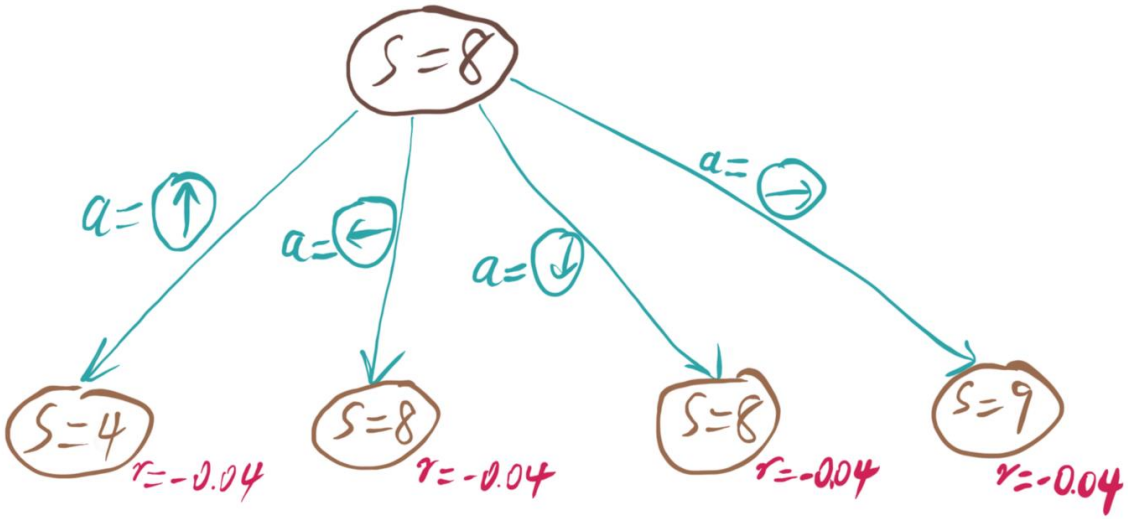
- Policy iteration is used to find the optimal policy
- In our grid world, a normal state has a reward of -0.04, a good green ending state has a reward of +1, and a bad red ending state has a reward of -1.

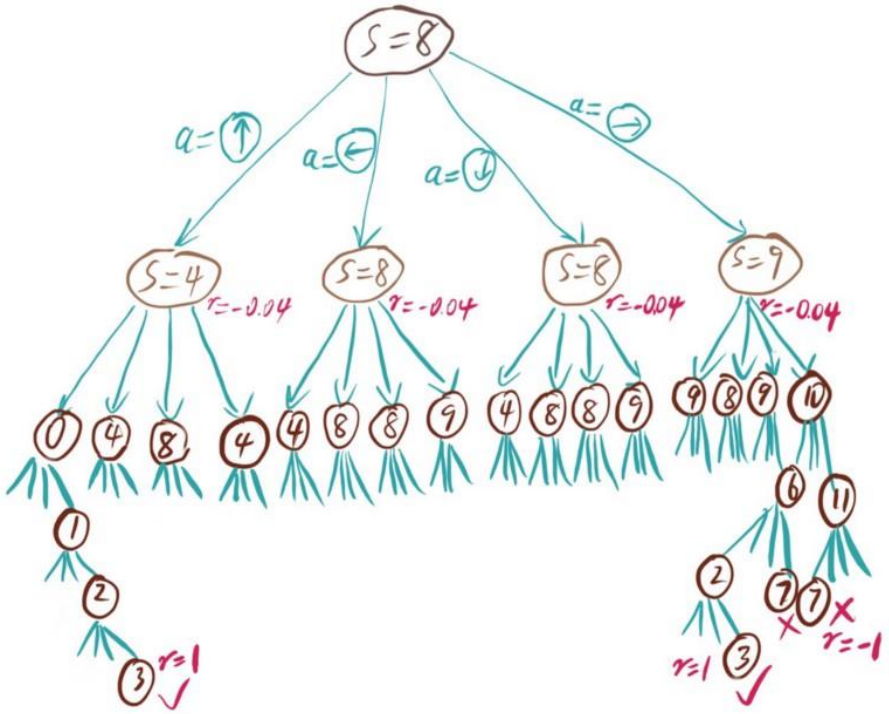
- Just imagine we are standing in our initial state *s* = 8 and we need to decide which action to take: either go right to *s* = 9 or go up to *s*= 4. The state *s* = 9 gives us a reward of -0.04 and so does the state *s* = 4.



Policy as a path of a tree

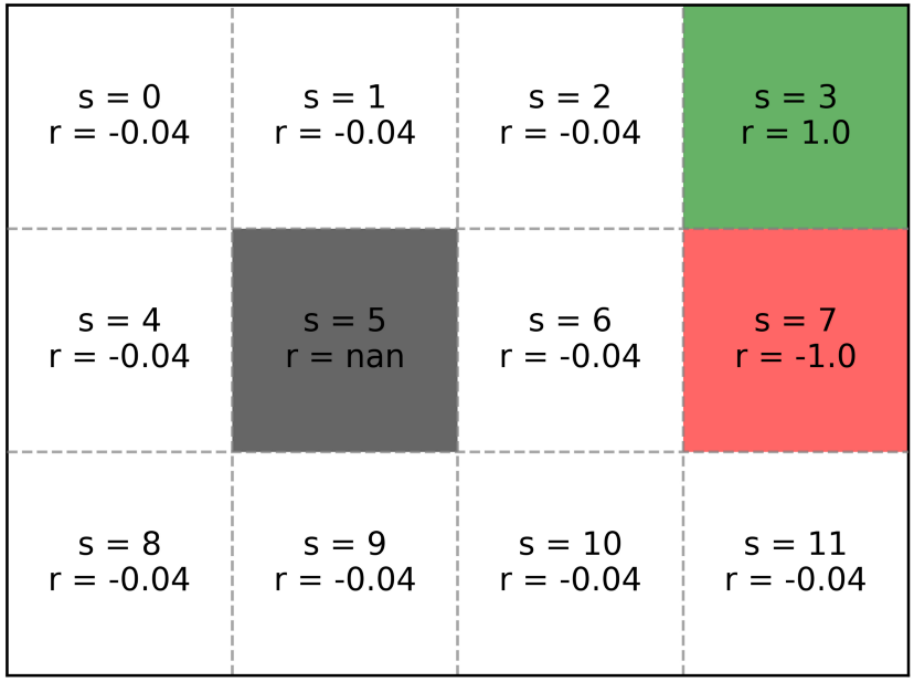
Backup Problem?





s = 0 r = -0.04	s = 1 r = -0.04	s = 2 r = -0.04	s = 3 r = 1.0
s = 4 r = -0.04	s = 5 r = nan	s = 6 r = -0.04	s = 7 r = -1.0
s = 8 r = -0.04	s = 9 r = -0.04	s = 10 r = -0.04	s = 11 r = -0.04

Utility of states



- To perform *backup*, we introduce a concept called *utility* or *value*. Each state has a utility that represents how good it is in terms of leading to a good future path.
- States may have same reward but has different utility values (Ex. Two companies)
- From s8 choose either s4 or s9
- If you choose s4 the let take path [4 0 1 2 3]
- V[4]=-0.04-0.04-0.04-0.04+1=0.84
- If you choose s9 the let take path [9 10 11 7]
- V[9]=-0.04-0.04-0.04-1=-1.12
- So s4 is better than s9 for next move from s8

Utility of a state under a *given policy*

Finding utility of states is not this easy since different actions lead to different states.

For instance, we may move along the path [9, 10, 11, 7] or [9, 10, 6, 2, 3] or even [9, 10, 6, 2, 1, 0, 4…] after $s = 9$.

To calculate the utility of $s = 9$, among all these possible paths, which one should we use?

Under these assumptions, for any state, the utility value is the summary of the reward of itself and all the future states for the given policy.

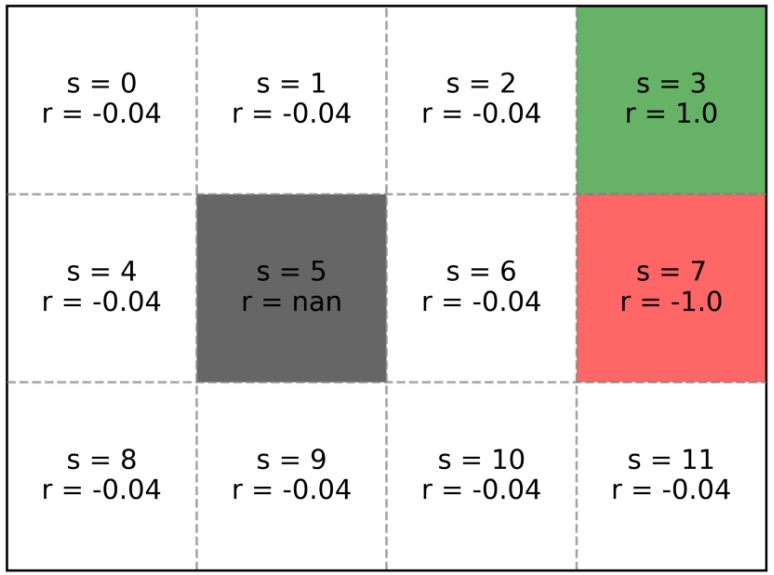
Example

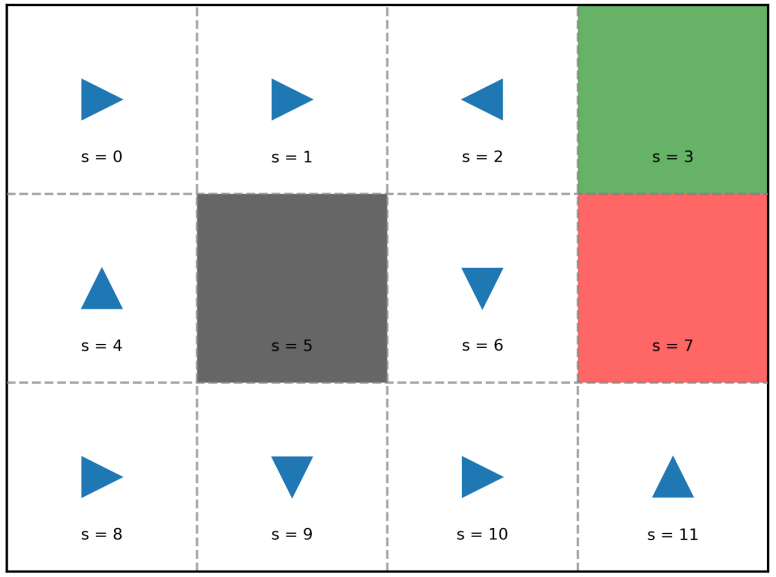
- V[6]=r[6]+r[10]+r[11]+r[7]=-1.2

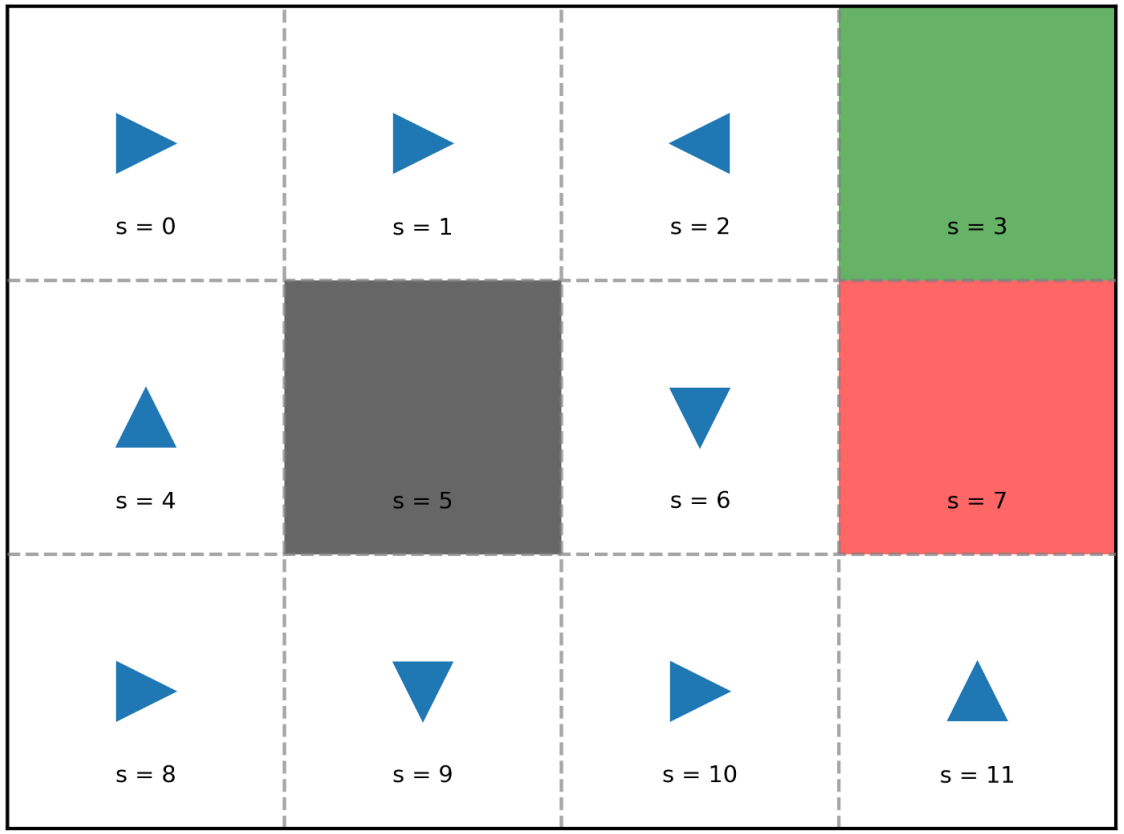
$$
\begin{aligned}
v(s_{t=0}) &= \text{rewards following policy}[s_{t=0}, s_{t=1}, s_{t=2}, s_{t=3}, \ldots] \\
&= r(s_{t=0}) + r(s_{t=1}) + r(s_{t=2}) + r(s_{t=3}) + \ldots \\
&= \sum_t r(s_t)
\end{aligned}
$$

$$v(s_{t=i}) = r(s_{t=i}) + v(s_{t=i+1})$$

$$v(s) = r(s) + v(s')$$







Discounted rewards

- Now let's look at another state $s = 0$
- Future states are [0, 1, 2, 1, 2, 1, 2, 1, 2, …]
- $v(0) = r(0) + r(1) + r(2) + r(1) + r(2) + \ldots$
- The utility value is either positive infinity or negative infinity.
- To address this issue, we introduce the concept of *discounted rewards*.

Discount Factor (γ)

- The key idea of *discounted rewards* is that the reward received in the future is less valuable than the reward received right now.
- In MDP, we use a discount factor γ to mimic the interest rate.

Now using *discounted rewards*, we have

$$\begin{aligned} v(s_{t=0}) &= \text{discounted rewards following policy}[s_{t=0}, s_{t=1}, s_{t=2}, s_{t=3}, \ldots] \\ &= r(s_{t=0}) + \gamma r(s_{t=1}) + \gamma^2 r(s_{t=2}) + \gamma^3 r(s_{t=3}) + \ldots \\ &= \sum_t \gamma^t \, r(s_t) \end{aligned}$$

With discounted factor, the relation between utility of a state and the utility of its successor state is

$$v(s) = r(s) + \gamma v(s')$$

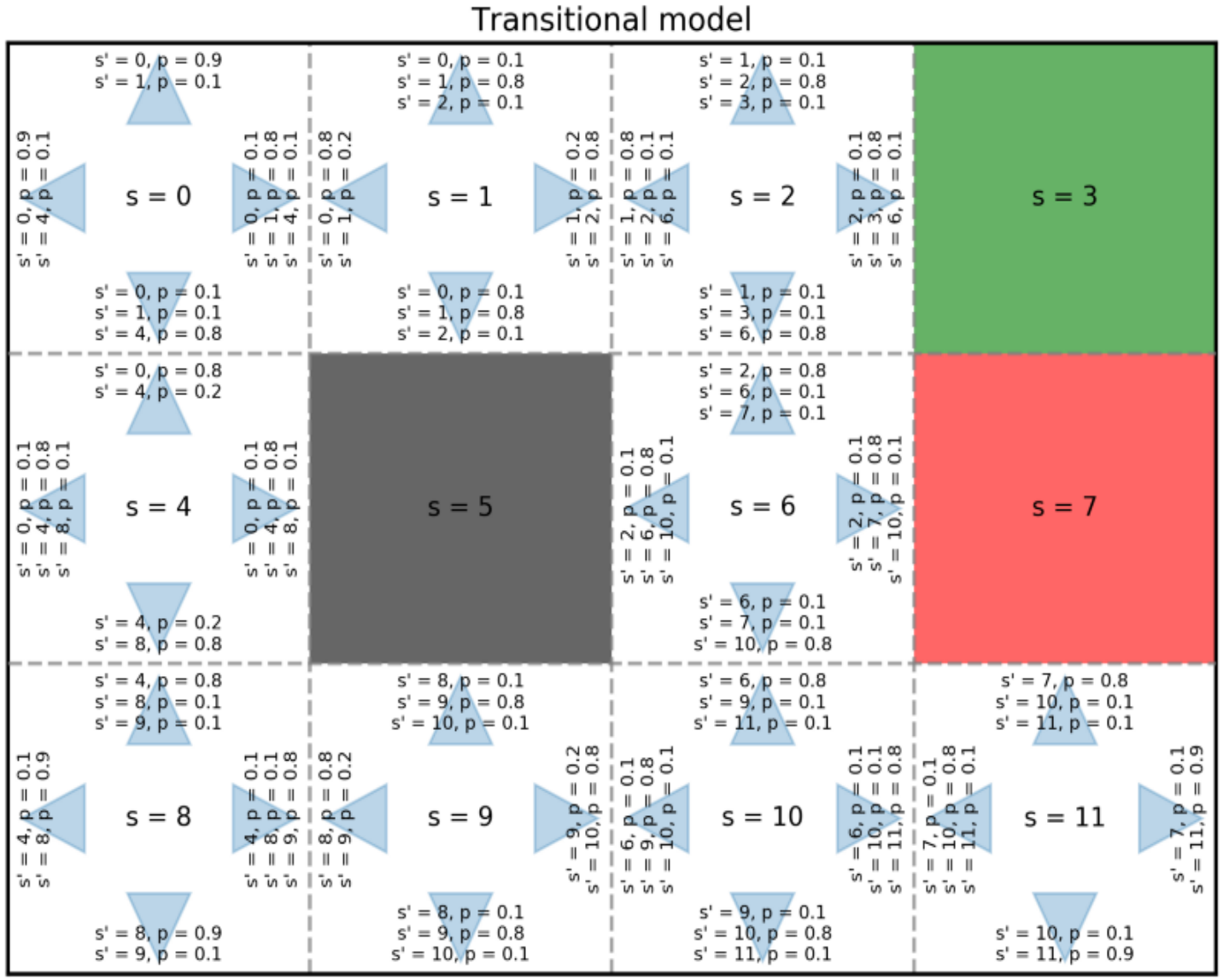
Before we have *additive rewards*, which is

$$\begin{aligned} v(s_{t=0}) &= \text{rewards following policy}[s_{t=0}, s_{t=1}, s_{t=2}, s_{t=3}, \ldots] \\ &= r(s_{t=0}) + r(s_{t=1}) + r(s_{t=2}) + r(s_{t=3}) + \ldots \\ &= \sum_t r(s_t) \end{aligned}$$

Stochastic World

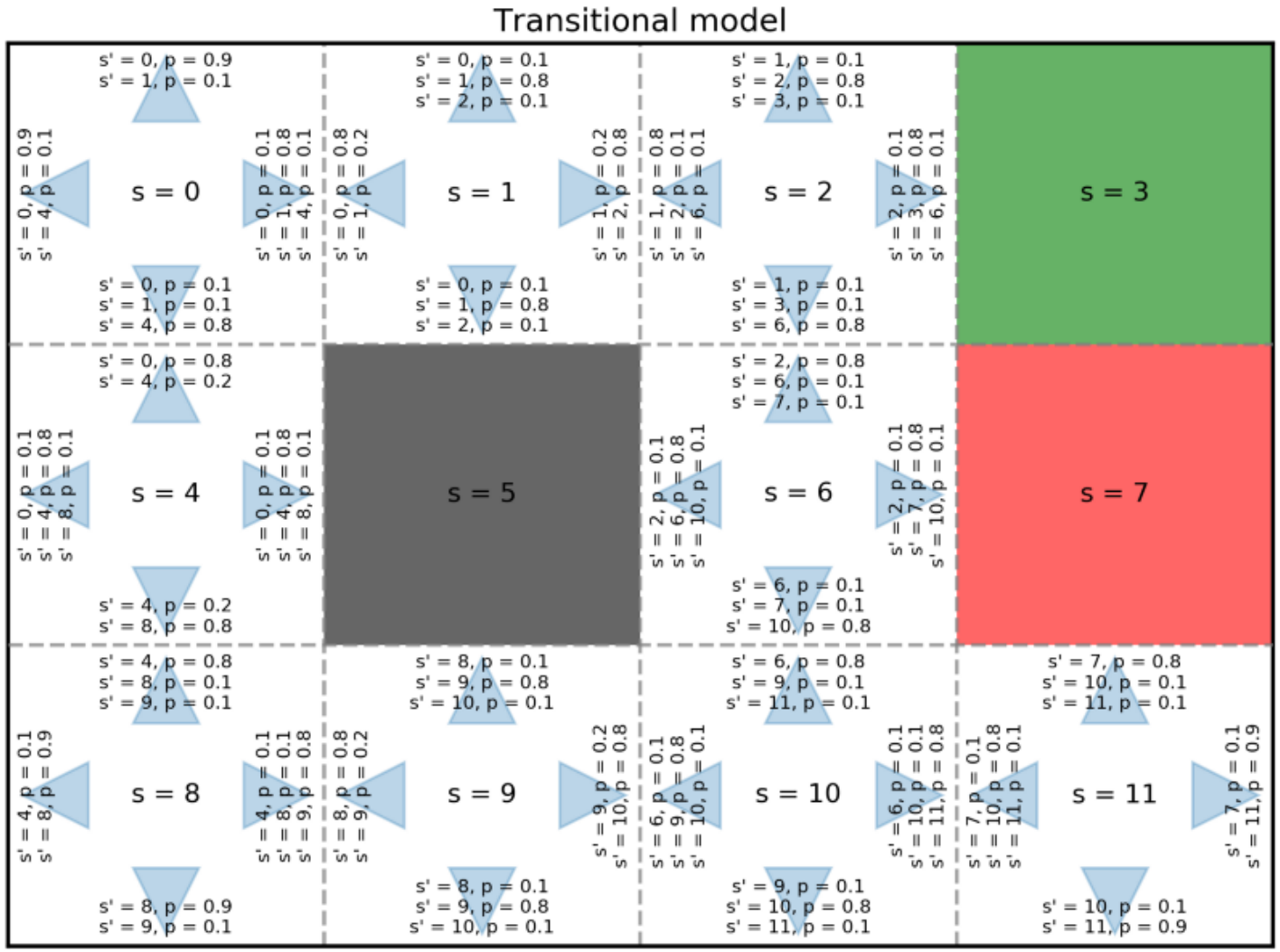
- In deterministic environment, we look at *s* = 6 we have a determined path of [6, 10, 11, 7] and utility value is –0.04-0.04-0.04-1=-1.2 with probability 1.

- Whereas in stochastic environment probability for this state utility value is only 08*0.8*0.8=0.512

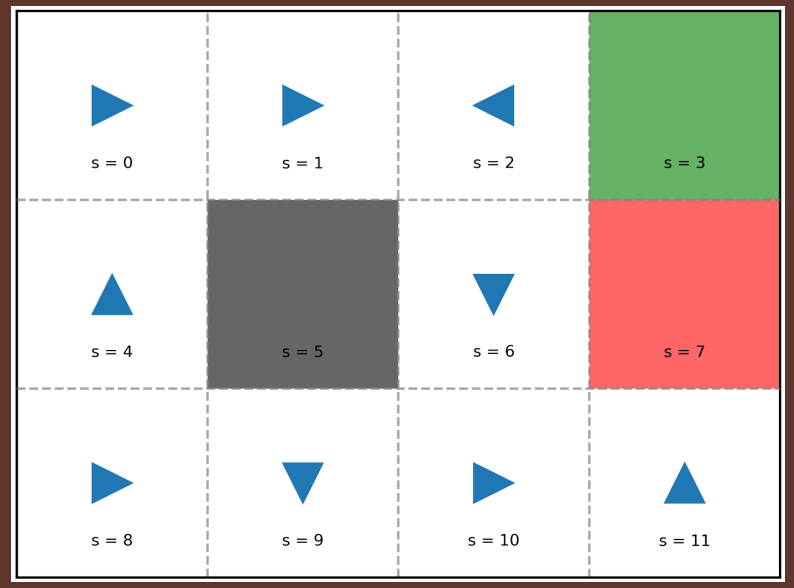


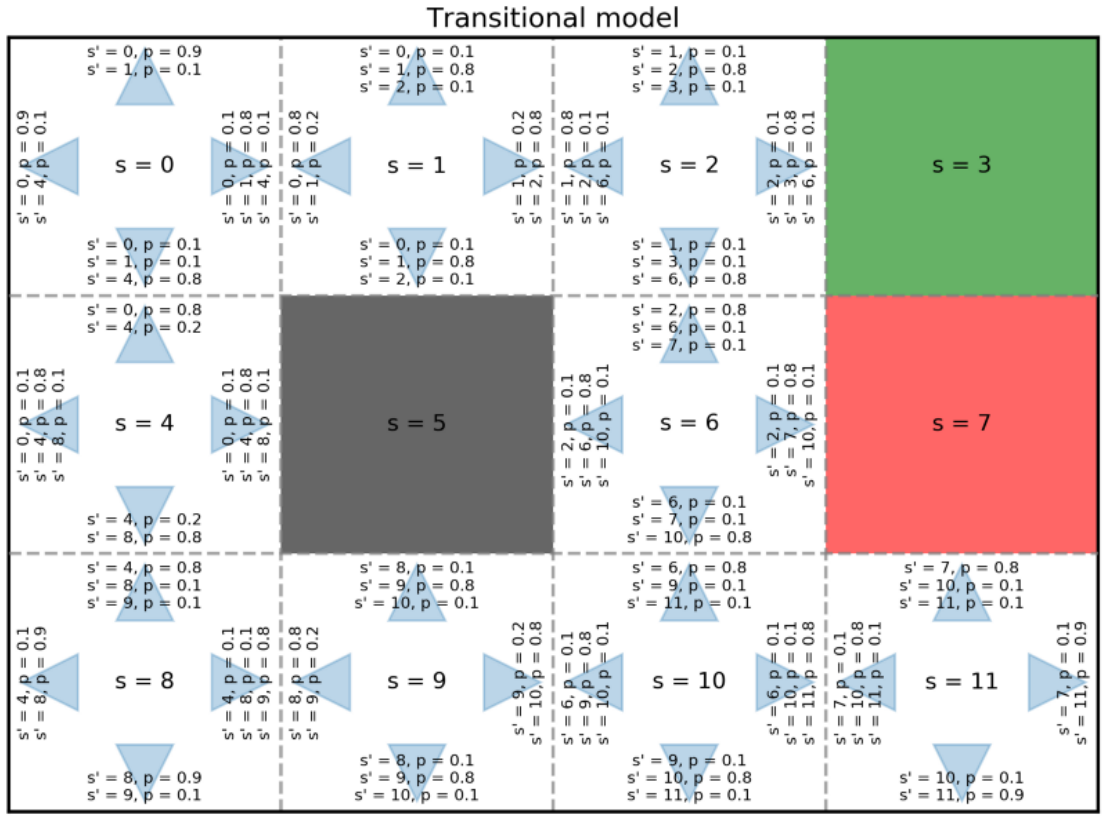
Path Probability

- Path1: 6 10 6 10 11 7
- Path2: 6 10 11 10 11 7
- Probability 0.04096



Calculation of Utility of the state in Stochastic World

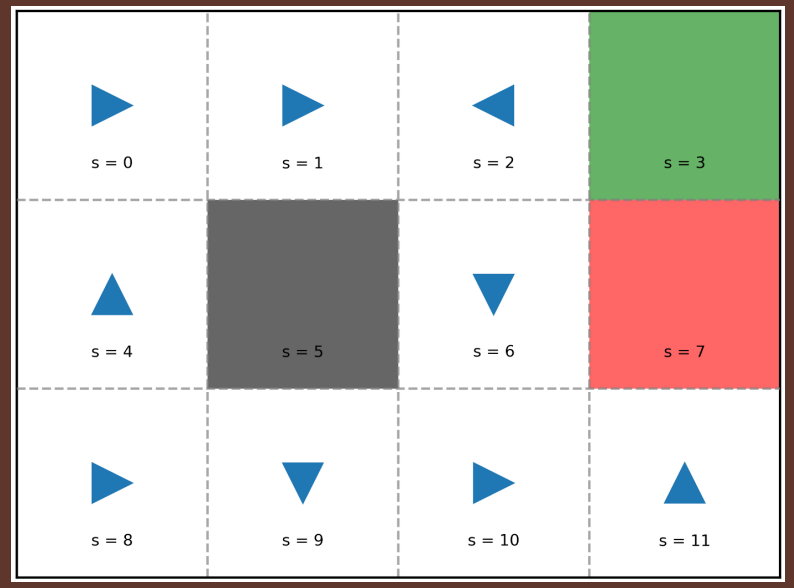


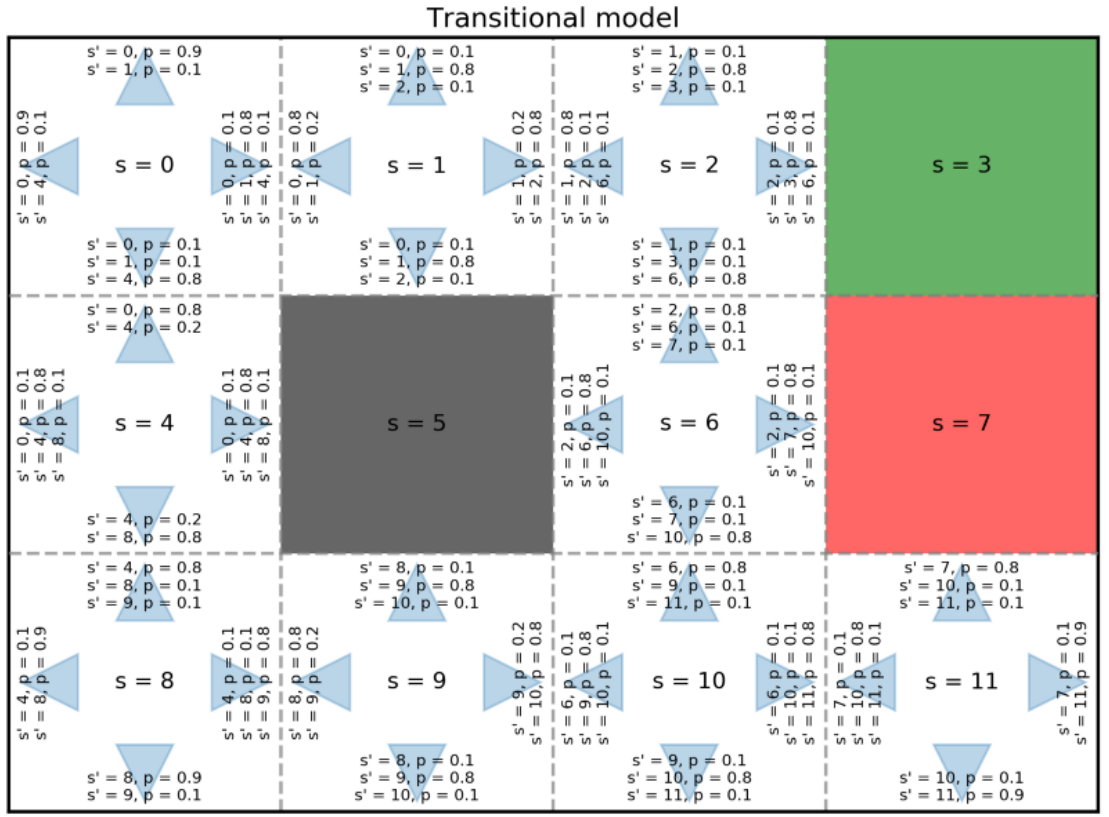


$$v(s) = r(s) + \gamma \sum_{s'} p(s'|s, a = \pi(s))v(s')$$

$$v(6) = r(6) + \gamma \left[0.8v(10) + 0.1v(6) + 0.1v(7)\right]$$

Calculation of Utility of the state in Stochastic World





$$v(s) = r(s) + \gamma \sum_{s'} p(s'|s, a = \pi(s))v(s')$$

$$v(6) = r(6) + \gamma \left[0.8v(10) + 0.1v(6) + 0.1v(7)\right]$$

Policy evaluation

- We can determine the utility of each state by directly solving these equations.
- This process is called *policy evaluation*: for a given policy
- We evaluate that policy by determining the utility of each state.

$$
\begin{aligned}
v(0) &= r(0) + \gamma\left[0.8v(1) + 0.1v(0) + 0.1v(4)\right] \\
v(1) &= r(1) + \gamma\left[0.8v(2) + 0.1v(1) + 0.1v(1)\right] \\
v(2) &= r(2) + \gamma\left[0.8v(1) + 0.1v(6) + 0.1v(2)\right] \\
v(3) &= r(3) + \gamma\left[v(3)\right] \\
v(4) &= r(4) + \gamma\left[0.8v(0) + 0.1v(4) + 0.1v(4)\right] \\
v(6) &= r(6) + \gamma\left[0.8v(10) + 0.1v(6) + 0.1v(7)\right] \\
v(7) &= r(7) + \gamma\left[v(7)\right] \\
v(8) &= r(8) + \gamma\left[0.8v(9) + 0.1v(4) + 0.1v(8)\right] \\
v(9) &= r(9) + \gamma\left[0.8v(9) + 0.1v(8) + 0.1v(10)\right] \\
v(10) &= r(10) + \gamma\left[0.8v(11) + 0.1v(6) + 0.1v(10)\right] \\
v(11) &= r(11) + \gamma\left[0.8v(7) + 0.1v(10) + 0.1v(11)\right]
\end{aligned}
$$

Iterative policy evaluation

- First Sweep
- set γ as 0.5
- first initialize the utility of each state as zero

$$
\begin{aligned}
v(0) &= r(0) + \gamma\,[0.8v(1) + 0.1v(0) + 0.1v(4)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times 0 + 0.1 \times 0] = -0.04 \\
v(1) &= r(1) + \gamma\,[0.8v(2) + 0.1v(1) + 0.1v(1)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times 0 + 0.1 \times 0] = -0.04 \\
v(2) &= r(2) + \gamma\,[0.8v(1) + 0.1v(6) + 0.1v(2)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04) + 0.1 \times 0 + 0.1 \times 0] = -0.056 \\
v(3) &= r(3) + \gamma\,[v(3)] \\
&= 1 + 0.5 \times 0 = 1 \\
v(4) &= r(4) + \gamma\,[0.8v(0) + 0.1v(4) + 0.1v(4)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04) + 0.1 \times 0 + 0.1 \times 0] = -0.056 \\
v(6) &= r(6) + \gamma\,[0.8v(10) + 0.1v(6) + 0.1v(7)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times 0 + 0.1 \times 0] = -0.04 \\
v(7) &= r(7) + \gamma\,[v(7)] \\
&= -1 + 0.5 \times 0 = -1 \\
v(8) &= r(8) + \gamma\,[0.8v(9) + 0.1v(4) + 0.1v(8)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times (-0.056) + 0.1 \times 0] = -0.0428 \\
v(9) &= r(9) + \gamma\,[0.8v(9) + 0.1v(8) + 0.1v(10)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times (-0.0428) + 0.1 \times 0] = -0.04214 \\
v(10) &= r(10) + \gamma\,[0.8v(11) + 0.1v(6) + 0.1v(10)] \\
&= -0.04 + 0.5 \times [0.8 \times 0 + 0.1 \times (-0.04) + 0.1 \times 0] = -0.042 \\
v(11) &= r(11) + \gamma\,[0.8v(7) + 0.1v(10) + 0.1v(11)] \\
&= -0.04 + 0.5 \times [0.8 \times (-1) + 0.1 \times (-0.042) + 0.1 \times 0] = -0.4421
\end{aligned}
$$

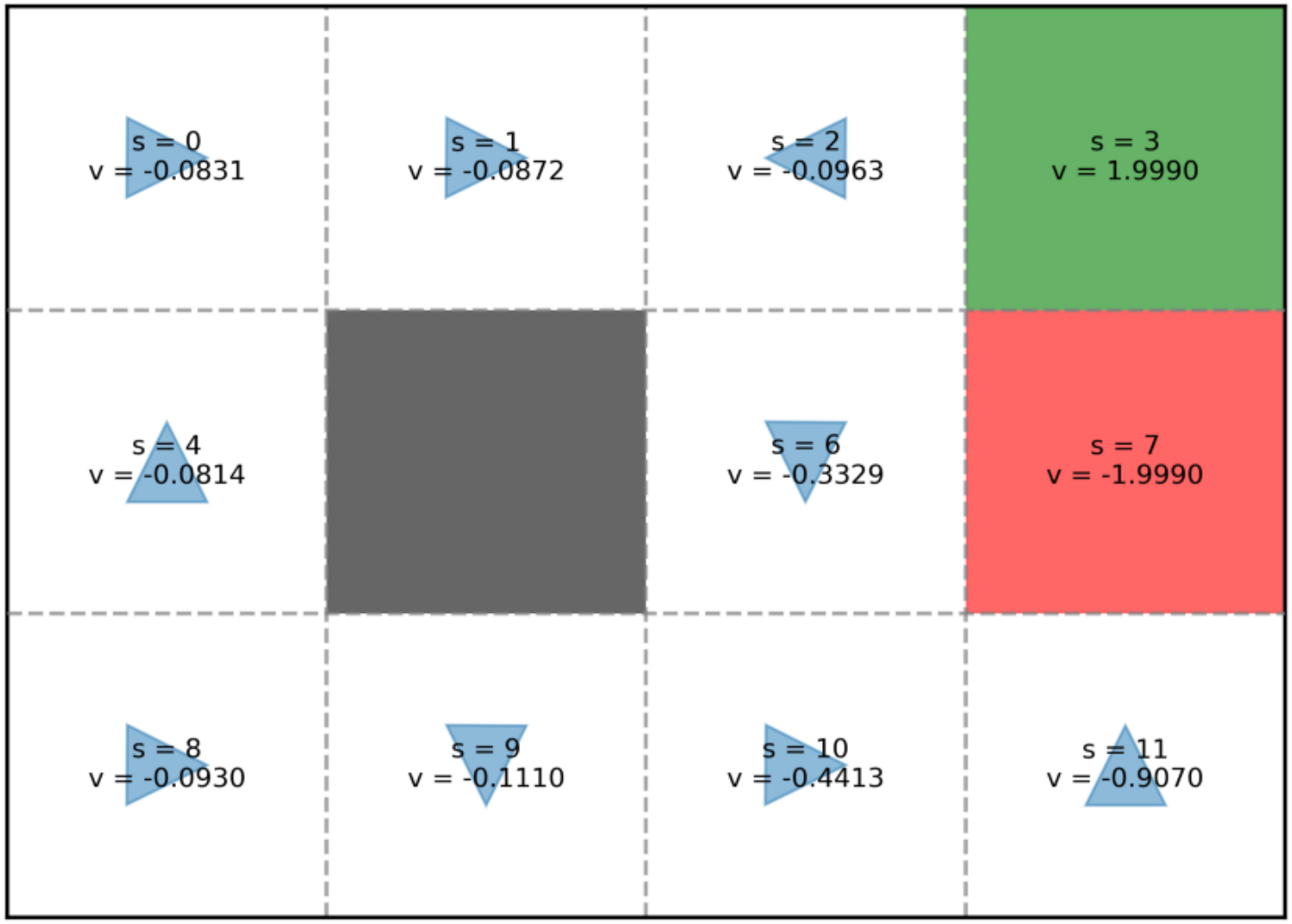
Second Sweep

Iterative policy evaluation

$$
\begin{aligned}
v(0) &= r(0) + \gamma\,[0.8v(1) + 0.1v(0) + 0.1v(4)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04) + 0.1 \times (-0.04) + 0.1 \times (-0.056)] = -0.0608 \\
v(1) &= r(1) + \gamma\,[0.8v(2) + 0.1v(1) + 0.1v(1)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.056) + 0.1 \times (-0.04) + 0.1 \times (-0.04)] = -0.0664 \\
v(2) &= r(2) + \gamma\,[0.8v(1) + 0.1v(6) + 0.1v(2)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.0664) + 0.1 \times (-0.04) + 0.1 \times (-0.056)] = -0.07136 \\
v(3) &= r(3) + \gamma\,[v(3)] \\
&= 1 + 0.5 \times 1 = 1.5 \\
v(4) &= r(4) + \gamma\,[0.8v(0) + 0.1v(4) + 0.1v(4)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.0608) + 0.1 \times (-0.056) + 0.1 \times (-0.056)] = -0.06992 \\
v(6) &= r(6) + \gamma\,[0.8v(10) + 0.1v(6) + 0.1v(7)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.042) + 0.1 \times (-0.04) + 0.1 \times (-1)] = -0.1088 \\
v(7) &= r(7) + \gamma\,[v(7)] \\
&= -1 + 0.5 \times (-1) = -1.5 \\
v(8) &= r(8) + \gamma\,[0.8v(9) + 0.1v(4) + 0.1v(8)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04214) + 0.1 \times (-0.06992) + 0.1 \times (-0.0428)] = -0.062492 \\
v(9) &= r(9) + \gamma\,[0.8v(9) + 0.1v(8) + 0.1v(10)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.04214) + 0.1 \times (-0.062492) + 0.1 \times (-0.042)] \\
&= -0.0620806 \\
v(10) &= r(10) + \gamma\,[0.8v(11) + 0.1v(6) + 0.1v(10)] \\
&= -0.04 + 0.5 \times [0.8 \times (-0.4421) + 0.1 \times (-0.1088) + 0.1 \times (-0.042)] \\
&= -0.22438 \\
v(11) &= r(11) + \gamma\,[0.8v(7) + 0.1v(10) + 0.1v(11)] \\
&= -0.04 + 0.5 \times [0.8 \times (-1.5) + 0.1 \times (-0.22438) + 0.1 \times (-0.4421)] \\
&= -0.673324
\end{aligned}
$$

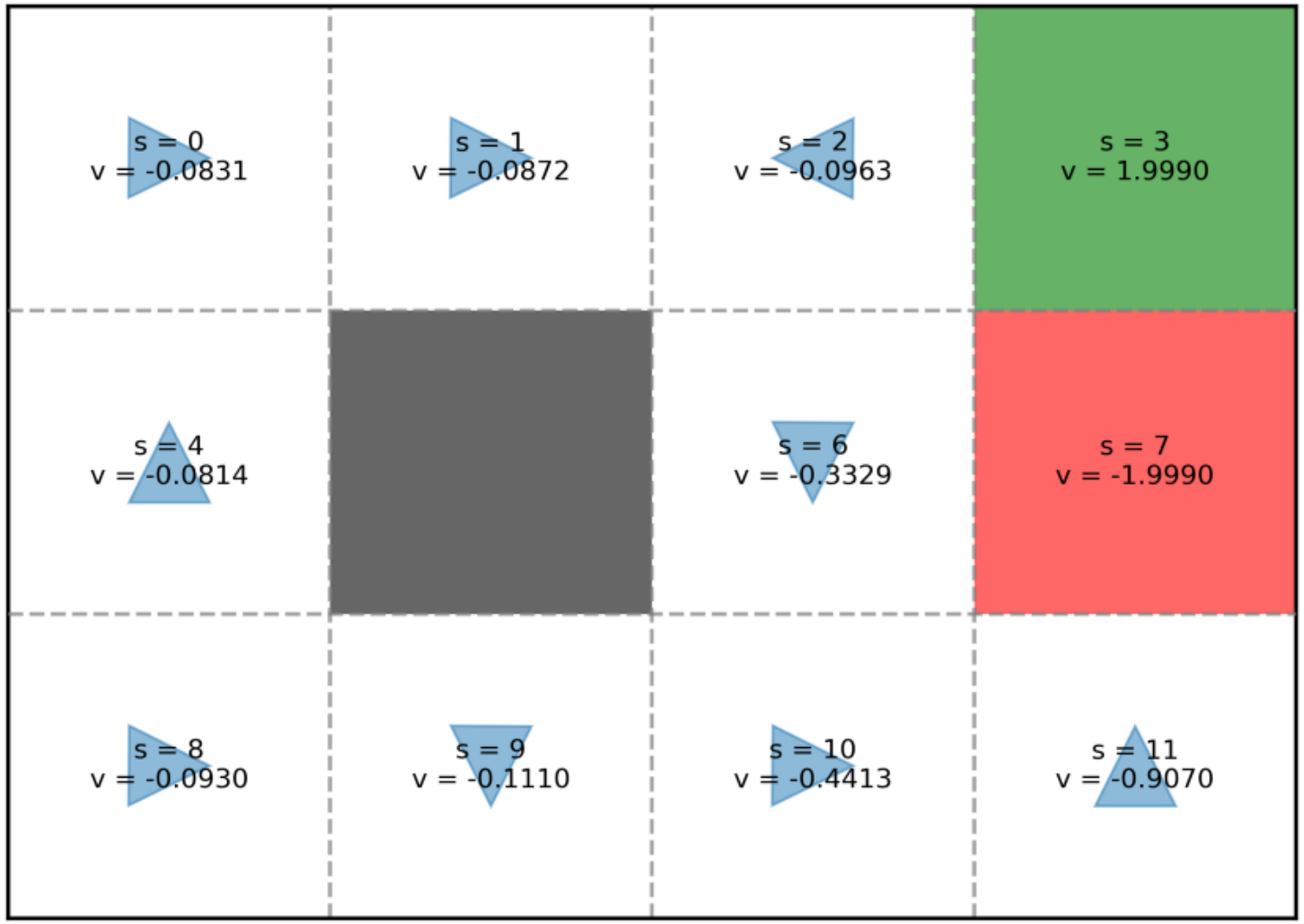
Utility after 11 sweeps

- We repeat sweeping like this for several times until the changes of utility values between consecutive sweeps are marginal.
- Here is the utility we get after 11 sweeps.

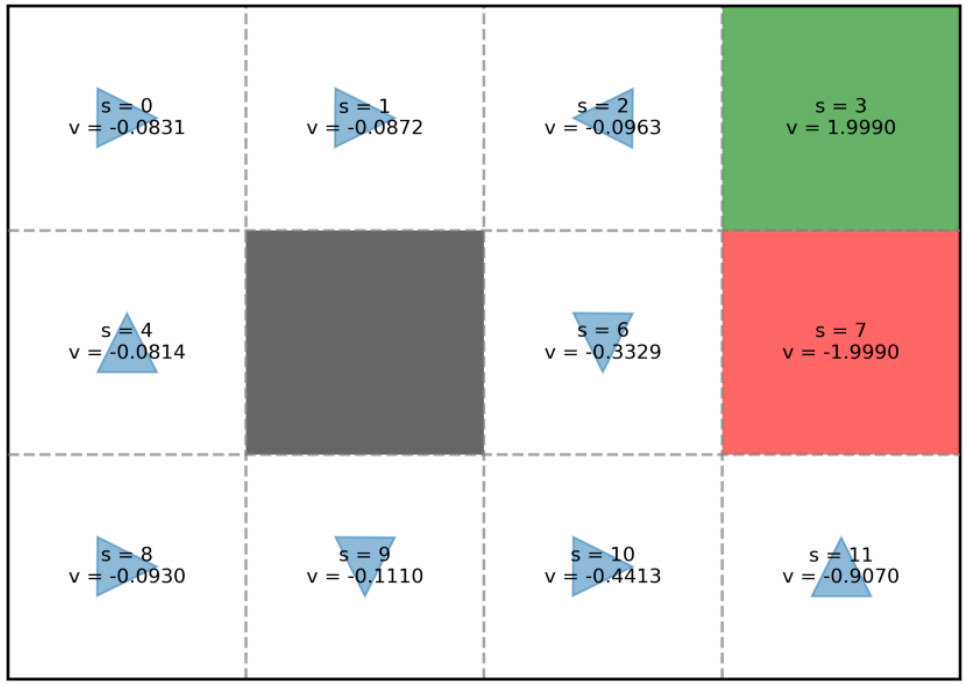


Policy improvement

- After we get the utility for this policy, it is time to *improve* the policy.
- For instance, if we look at state $s = 2$, potential successor states include 1,3, and 6. Clearly $s = 3$ is the best choice as it has the highest utility of 1.999.
- if we look at state $s = 8$, potential successor states include 4 and 9 with utility of -0.0814 and -0.1110 respectively. As such, s=4 is preferable than s=9.
- Since our MDP is stochastic, selecting a preferable successor state does not guarantee we will reach it. Rather than successor states, what we should compare is actions.

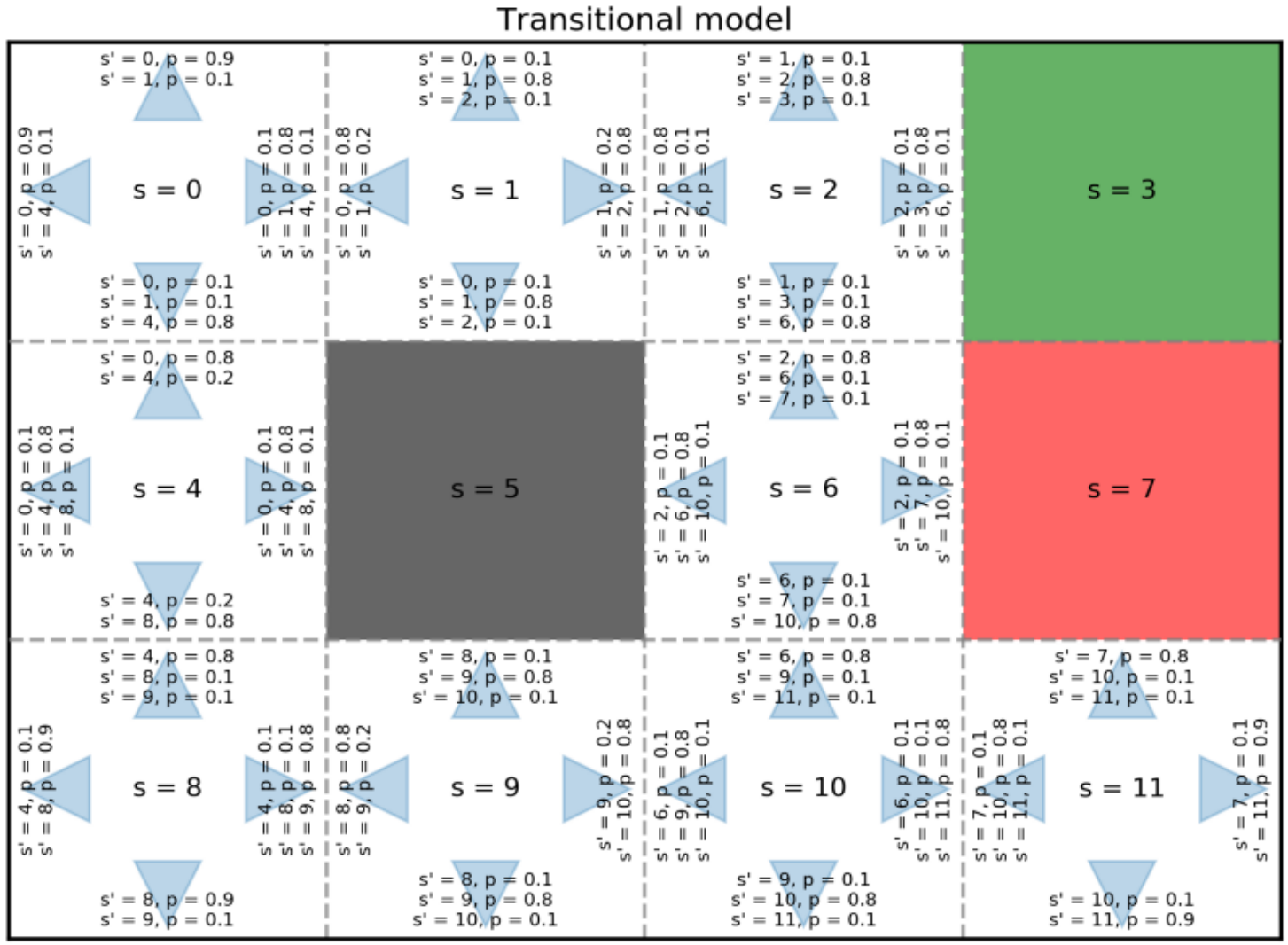


Utility of state based on actions



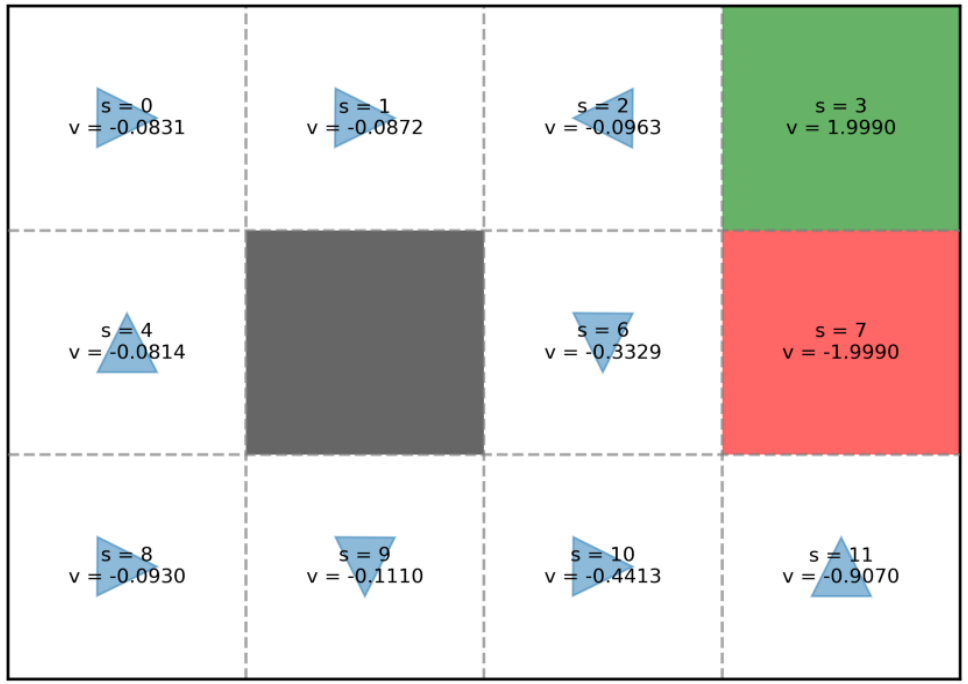
$$0.8v(2) + 0.1v(6) + 0.1v(7)$$

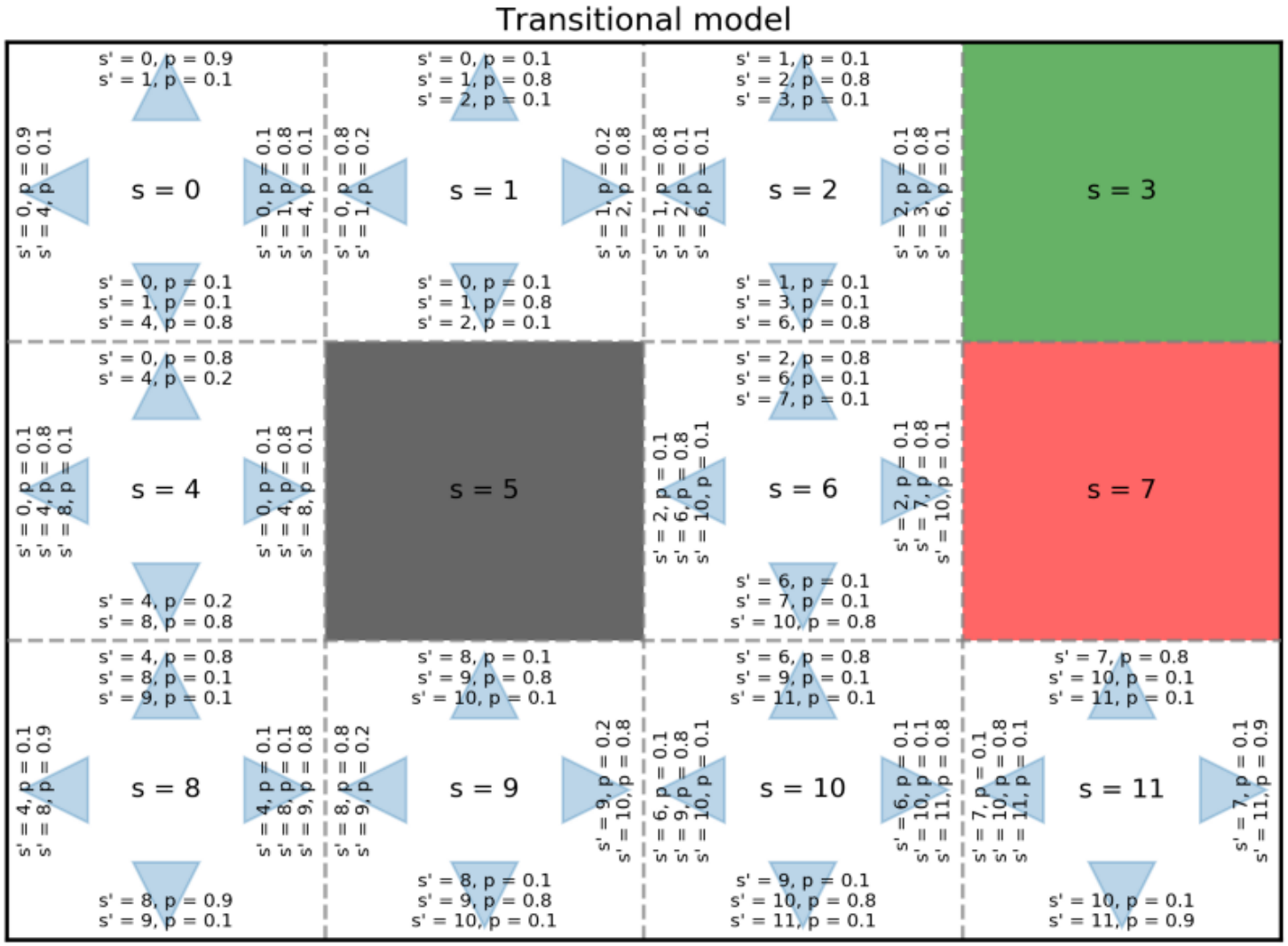
$$= 0.8 \times (-0.0963) + 0.1 \times (-0.3329) + 0.1 \times (-1.990) = -0.3093$$



- State S6
- Action UP

Utility of state based on actions

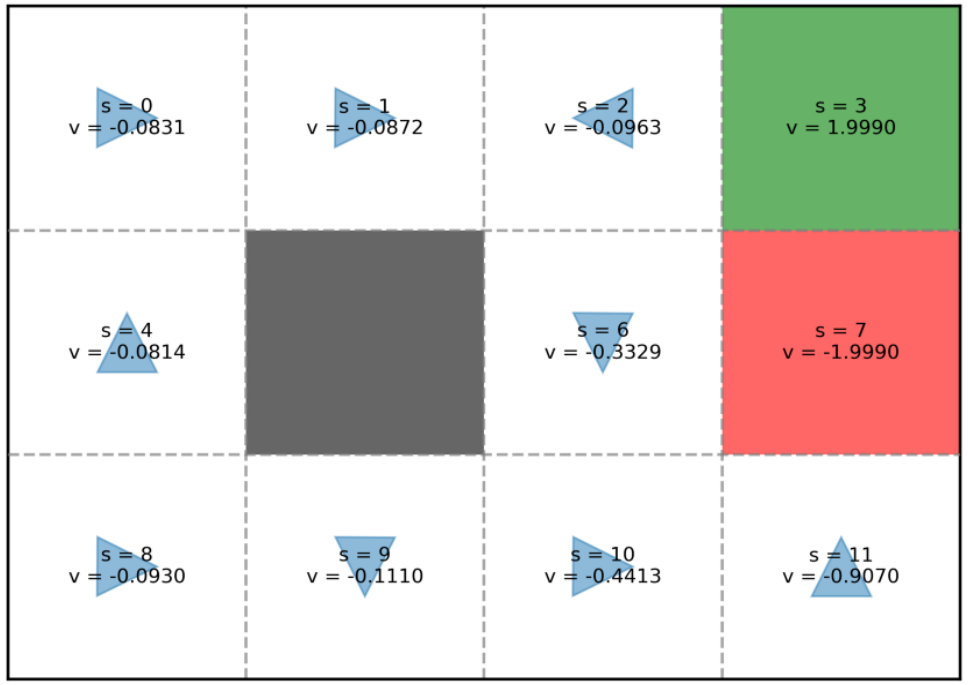


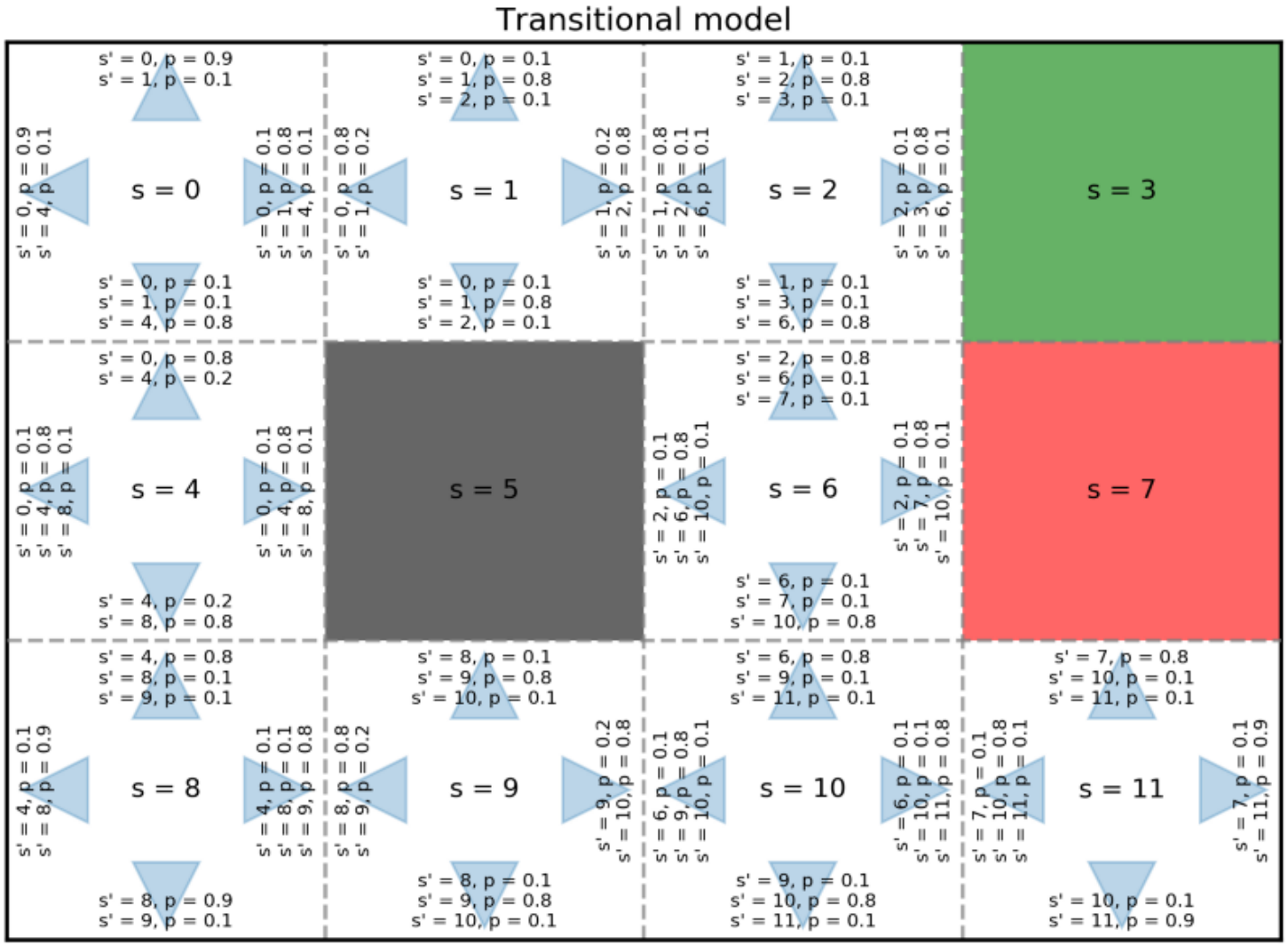


$$0.8v(6) + 0.1v(10) + 0.1v(2)$$
$$= 0.8 \times (-0.3329) + 0.1 \times (-0.4413) + 0.1 \times (-0.0963) = -0.3201$$

- State S6
- Action LEFT

Utility of state based on actions

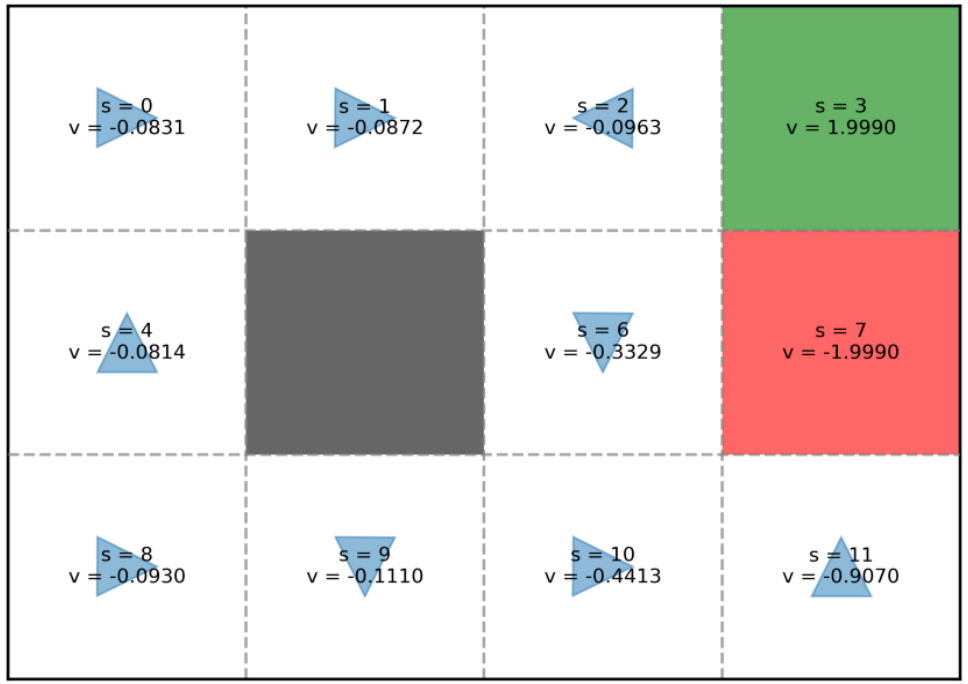


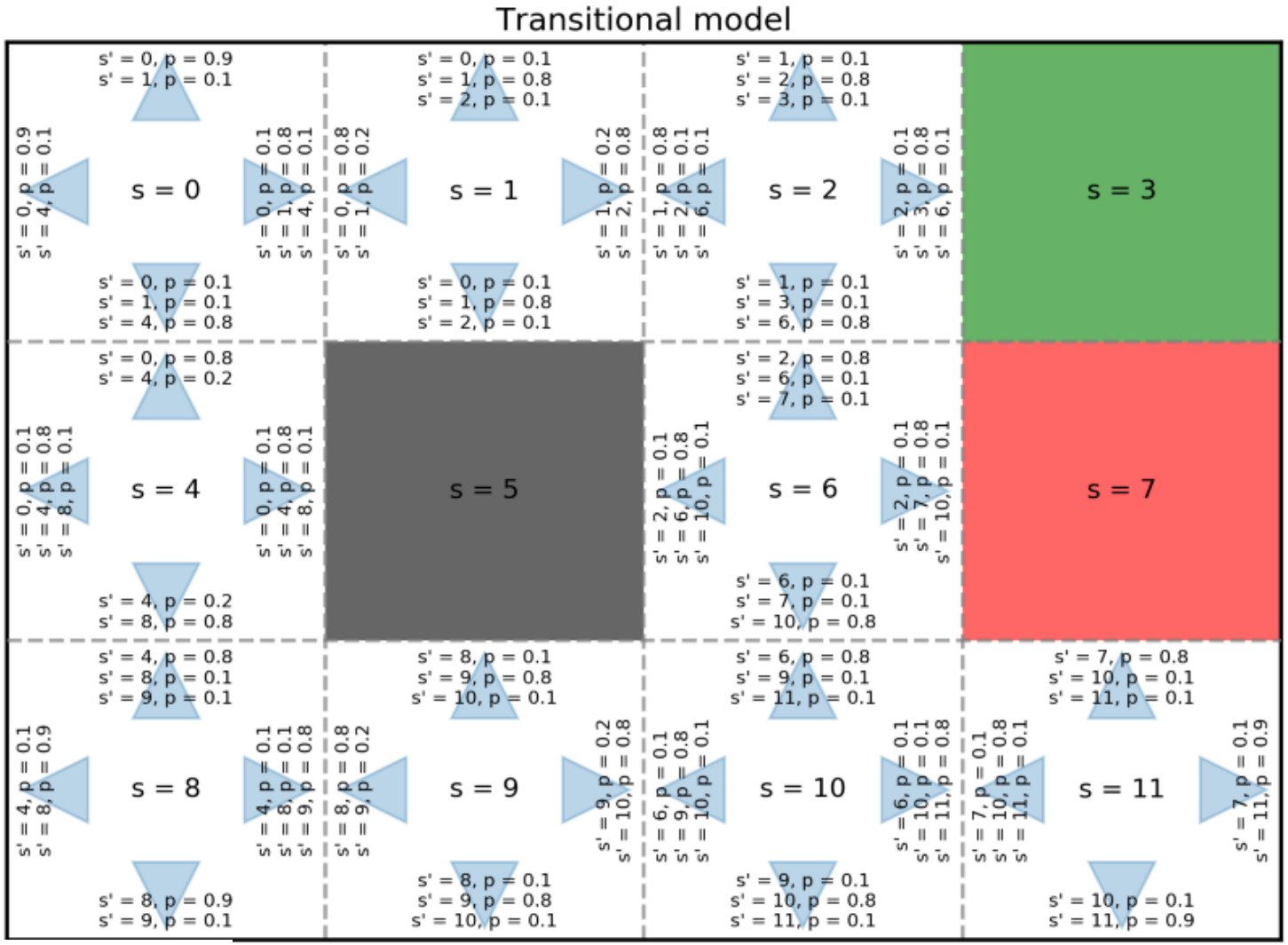


$$0.8v(10) + 0.1v(6) + 0.1v(7)$$
$$= 0.8 \times (-0.4413) + 0.1 \times (-0.3329) + 0.1 \times (-1.9990) = -0.5862$$

- State S6
- Action Down

Utility of state based on actions



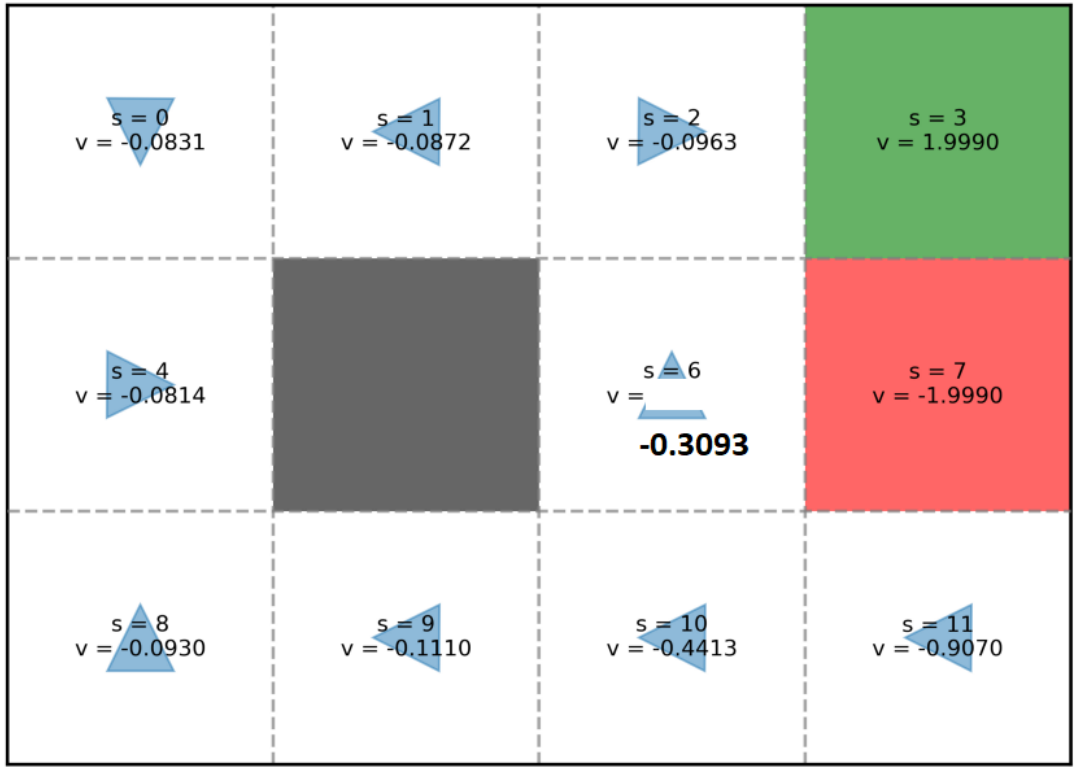


$$0.8v(7) + 0.1v(2) + 0.1v(10)$$
$$= 0.8 \times (-1.9990) + 0.1 \times (-0.0963) + 0.1 \times (-0.4413) = -1.6530$$

- State S6
- Action Right

Utility of state based on actions

- Comparing the outcomes of these four possible actions, clearly *UP* is the best choice. Therefore, we should update the policy of the state $s = 6$ to *UP*.



$$\pi(s) \leftarrow \text{argmax}_a \left[\sum_{s'} p(s'|s,a) v(s') \right]$$

Policy iteration

We do get a policy better than our initial one, but it is not necessarily the best one.

What we need is to repeat this whole process of policy evaluation and then policy improvement again and again.

When we perform policy evaluation again, we do not need to initialize the utility to all zero.

Policy iteration

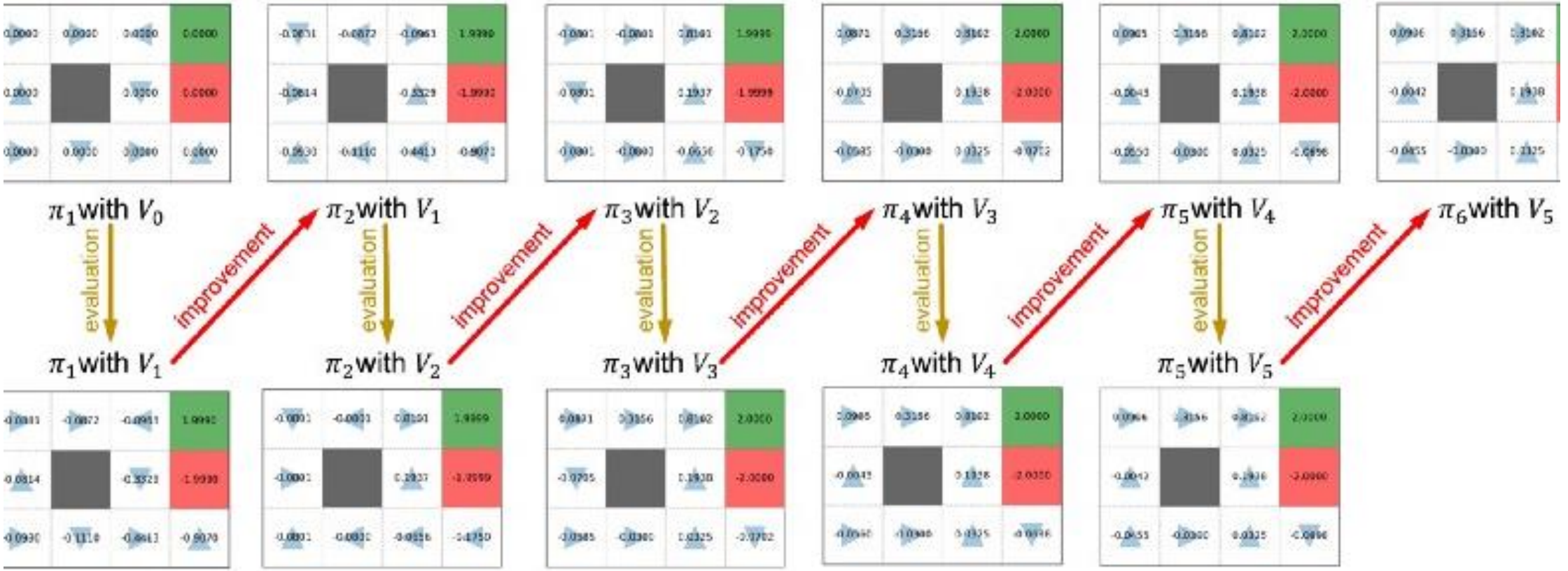
What policy evaluation does is to update the utility values while keeping policy constant (from $\pi[i]$, $V[i-1]$ to $\pi[i]$, $V[i]$).

What policy improvement does is to update the policy while keeping the utility values constant (from $\pi[i]$, $V[i]$ to $\pi[i+1]$, $V[i]$).

When $\pi[i]$ is the same as $\pi[i+1]$, there is no need to continue and we call that convergence.

Policy iteration

- In this iterative process, we start from a randomly generated policy π[1] and eventually converge at the optimal policy π[6].



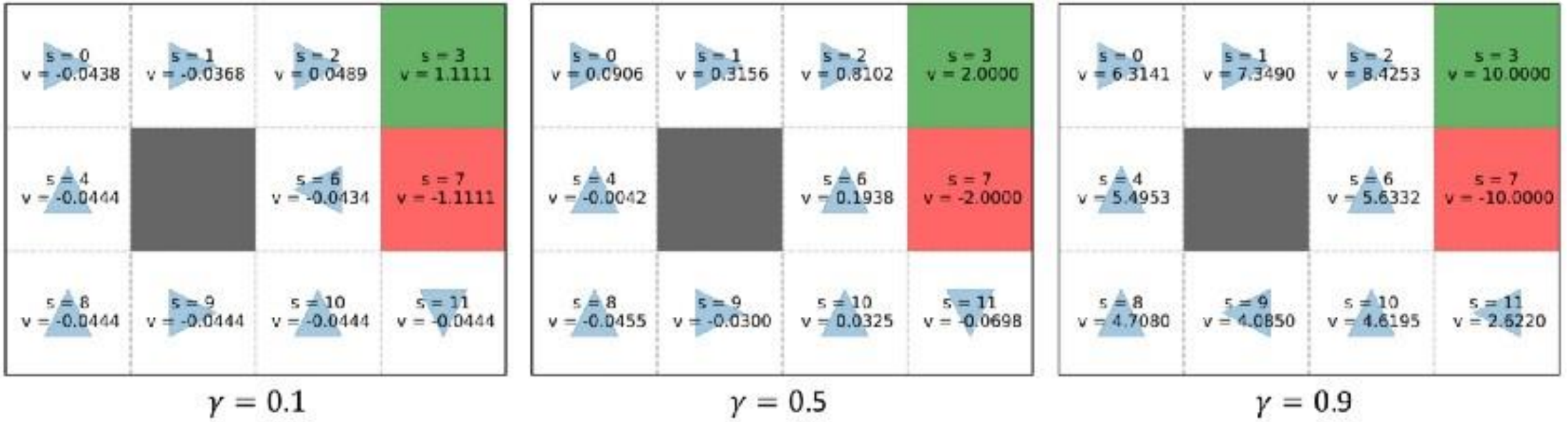
$$v(s_{t=0}) = r(s_{t=0}) + \gamma r(s_{t=1}) + \gamma^2 r(s_{t=2}) + \gamma^3 r(s_{t=3}) + \ldots$$
$$= \gamma^t \sum_t r(s_t)$$

Effects of discounted factor (γ)

$$v(s_{t=0}) = r(s_{t=0}) + 0.1r(s_{t=1}) + 0.1^2 r(s_{t=2}) + 0.1^3 r(s_{t=3}) + \ldots$$
$$= r(s_{t=0}) + 0.1r(s_{t=1}) + 0.01r(s_{t=2}) + 0.001r(s_{t=3}) + \ldots$$

$$v(s_{t=0}) = r(s_{t=0}) + 0.9r(s_{t=1}) + 0.9^2 r(s_{t=2}) + 0.9^3 r(s_{t=3}) + \ldots$$
$$= r(s_{t=0}) + 0.9r(s_{t=1}) + 0.81r(s_{t=2}) + 0.729r(s_{t=3}) + \ldots$$

- Discounted factor γ is used to *discount* the rewards of future states when adding them together.
- In the case of γ = 0.1, the reward we received in the future has little effect on the utility of a state.
- when γ =0.9, the reward at *t* = 3 has a factor of 0.729. Changes of that reward may considerably change the utility.



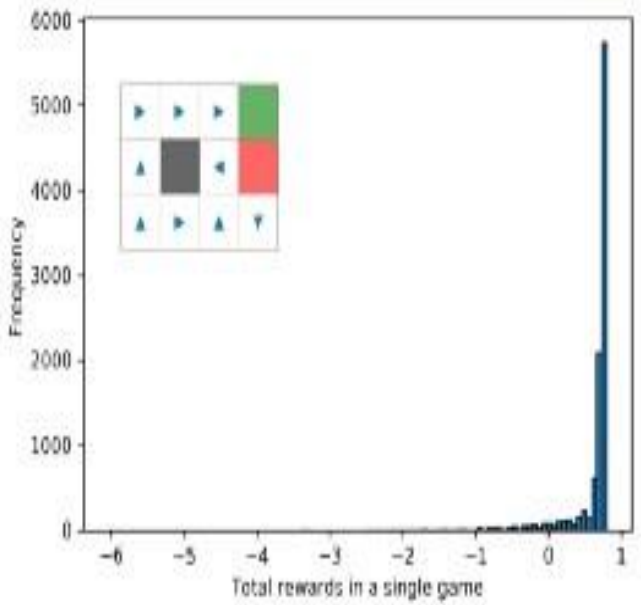
Effects of discounted factor (γ)

- Smaller γ emphasizes the immediate reward of the current state while larger γ emphasizes long term reward in the future.

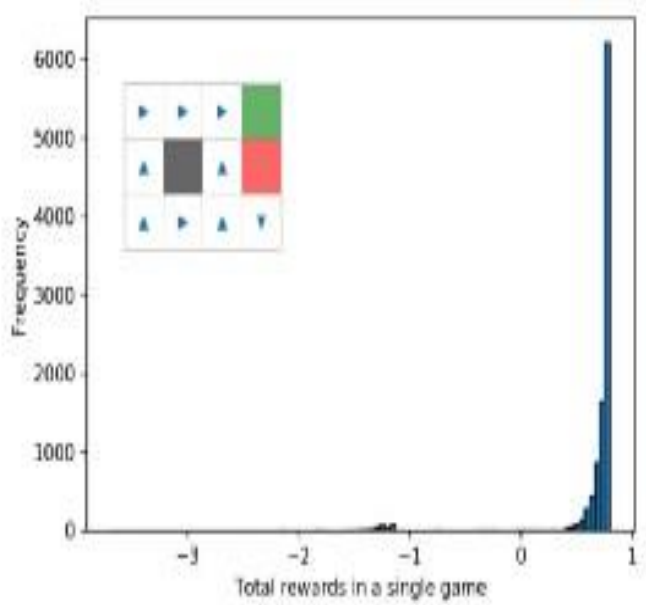
Optimal policy based on γ

γ	Mean	Min
0.1	0.63	-6.04
0.5	0.68	-3.68
0.9	0.70	-1.6

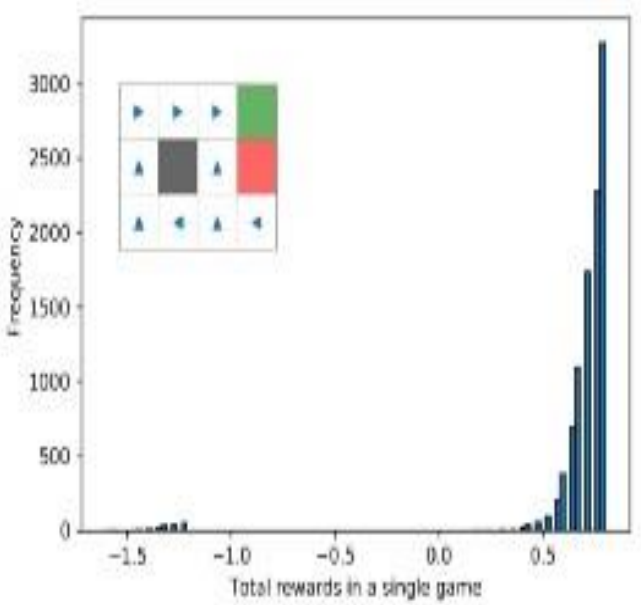
- Monte Carlo approach to assess these optima policies corresponding to different γ values
- Optimal γ is based on higher mean and higher minimum total reward.



$\gamma = 0.1$



$\gamma = 0.5$



$\gamma = 0.9$

Optimal policy based on γ

Larger γ requires more sweeps and thus requires more computational time.

Larger γ works better in this problem and tends to work better in many problems.

However, this does not mean that it works better in every problem.

Pseudo code of policy evaluation

input : reward function $r(s)$, transitional model $p(s'|s,a)$,
discounted factor γ, convergence threshold θ
policy $\pi(s)$, value $v(s)$
output: converged value $v(s)$

```
converge ← false
while converge = false do
    Δ ← 0
    for s ∈ S do
        temp ← v(s)
        v(s) ← r(s) + γ Σ_{s'} p(s'|s, a = π(s)) v(s')
        Δ ← max(Δ, |temp − v(s)|)
    end
    if Δ < θ then
        converge ← true
    end
end
return v(s)
```

Pseudo code of policy improvement

input : transitional model $p(s'|s,a)$,
policy $\pi(s)$, value $v(s)$
output: updated policy $\pi(s)$,
binary indicating whether any change occurs

```
change ← false
for s ∈ S do
    temp ← π(s)
    π(s) ← argmax_a Σ_{s'} p(s'|s,a)v(s')
    if temp ≠ π(s) then
        change ← true
    end
end
return π(s), change
```

Pseudo code of policy iteration

input : reward function $r(s)$, transitional model $p(s'|s,a)$,
discounted factor γ, convergence threshold θ
output: optimal policy $\pi^*(s)$

initialize $\pi(s)$ randomly
initialize $v(s)$ with zeros
stable $\leftarrow$ false
while stable = false **do**
    $v(s) \leftarrow$ `policy evaluation`$(r(s), p(s'|s,a), \gamma, \theta, \pi(s), v(s))$
    $\pi(s)$, change $\leftarrow$ `policy improvement`$(p(s'|s,a), \pi(s), v(s))$
    if change = false **then**
        stable $\leftarrow$ true
    end
end
$\pi^*(s) \leftarrow \pi(s)$
return $\pi^*(s)$

$$v(s) = \max_{a_i} \left[r(s) + \gamma \sum_{s'} p(s'|s, a = a_i) v(s') \right]$$

$$= r(s) + \gamma \max_{a} \left[\sum_{s'} p(s'|s, a) v(s') \right]$$

- **Bellman equation**

Markov decision process: value iteration

Value Iteration

- Similar as we did in policy iteration, we start from initializing the utility of every state as zero and we set γ as 0.5.

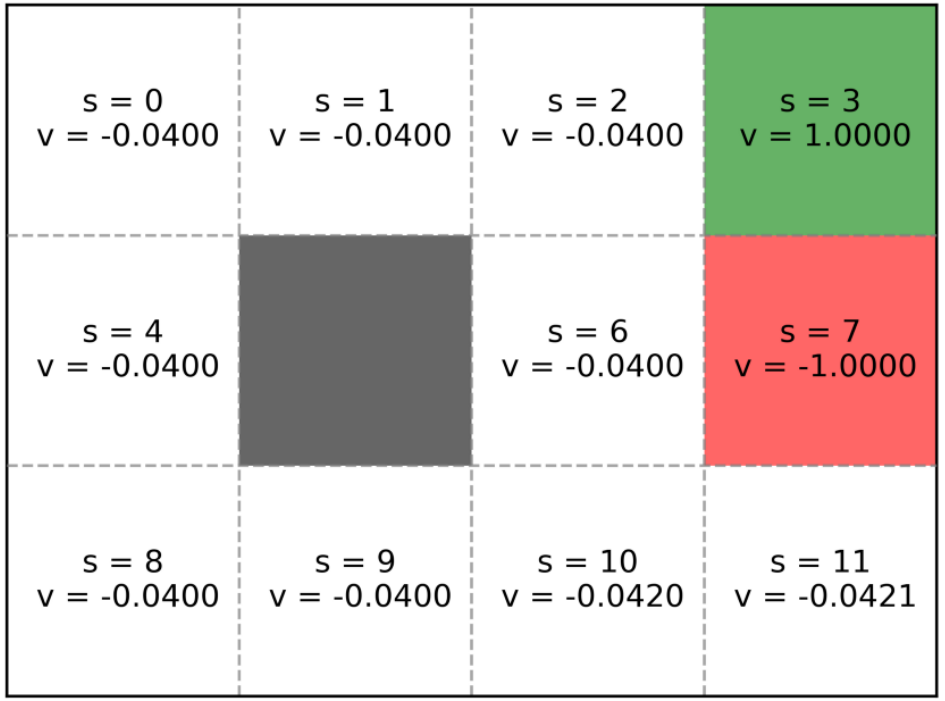
$$v(0) = r(0) + \gamma \max_a \left[\sum_{s'} p(s'|s=0,a) v(s') \right]$$

$$= r(0) + \gamma \max \begin{bmatrix} \sum_{s'} p(s'|s=0, a=\text{UP}) v(s') \\ \sum_{s'} p(s'|s=0, a=\text{LEFT}) v(s') \\ \sum_{s'} p(s'|s=0, a=\text{DOWN}) v(s') \\ \sum_{s'} p(s'|s=0, a=\text{RIGHT}) v(s') \end{bmatrix}$$

$$= r(0) + \gamma \max \begin{bmatrix} 0.8v(0) + 0.1v(0) + 0.1v(1) \\ 0.8v(0) + 0.1v(4) + 0.1v(0) \\ 0.8v(4) + 0.1v(1) + 0.1v(0) \\ 0.8v(1) + 0.1v(0) + 0.1v(4) \end{bmatrix}$$

$$= -0.04 + 0.5 \times \max \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = -0.04$$

State 2: V(2)

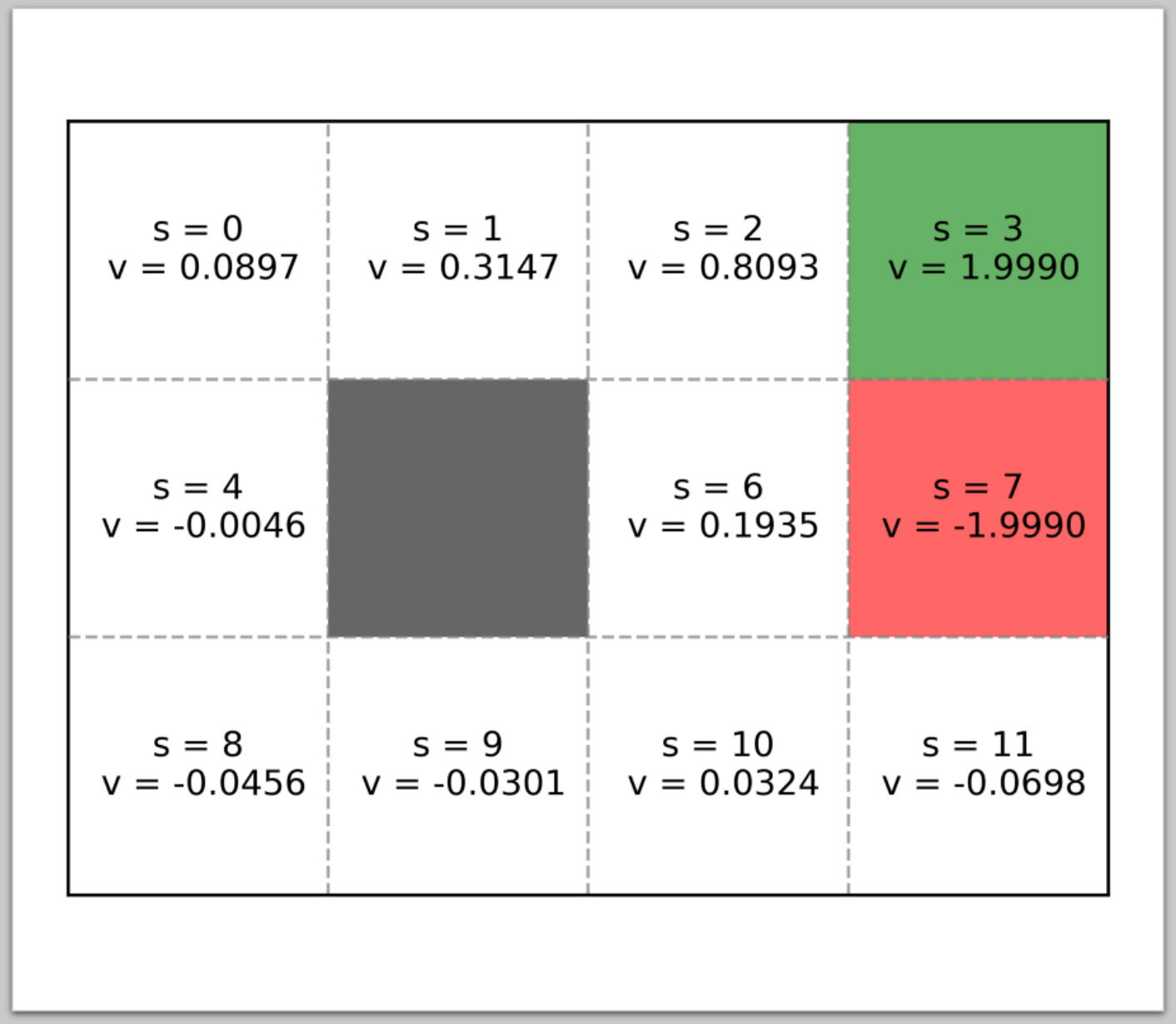


$$v(1) = r(1) + \gamma \max \begin{bmatrix} 0.8v(1) + 0.1v(0) + 0.1v(2) \\ 0.8v(0) + 0.1v(1) + 0.1v(1) \\ 0.8v(1) + 0.1v(2) + 0.1v(0) \\ 0.8v(2) + 0.1v(1) + 0.1v(1) \end{bmatrix}$$

$$= -0.04 + 0.5 \times \max \begin{bmatrix} 0.1 \times (-0.04) \\ 0.8 \times (-0.04) \\ 0.1 \times (-0.04) \\ 0 \end{bmatrix} = -0.04$$

Sweep 11

- We repeat iteration until the change of utility between two consecutive iterations are marginal. After 11 iterations, the change of utility value of any state is smaller than 0.001. We stop here and the utility we get is the utility associated with the optimal policy.



Pseudo-code of value iteration

input : reward function $r(s)$, transitional model $p(s'|s,a)$,
discounted factor γ, convergence threshold θ
output: optimal policy π^*

initialize $v(s)$ with zeros
converge $\leftarrow$ false
while converge $=$ false **do**
    $\Delta \leftarrow 0$
    for $s \in S$ **do**
        temp $\leftarrow v(s)$
        $v(s) \leftarrow r(s) + \gamma \max_a \sum_{s'} p(s'|s,a)v(s')$
        $\Delta \leftarrow \max(\Delta, |\text{temp} - v(s)|)$
    end
    if $\Delta < \theta$ **then**
        converge $\leftarrow$ true
    end
end
for $s \in S$ **do**
    $\pi^*(s) \leftarrow \text{argmax}_a \sum_{s'} p(s'|s,a)v(s')$
end
return π^*

Important Questions

- *Define Markov decision process (Slide 2)*
- *Explain about grid world problem (Slide 3)*
- *Explain about Agent, Environment, state, action, reward, additive reward, transition model, fully observable. Sequential, policy* ***(Slides 4-13)***
- *Explain about probability distribution of state change in deterministic and stochastic grid world game (Slide 14 - 15)*

Important Questions

How we execute a policy In Stochastic Environment (Slides 17-20)

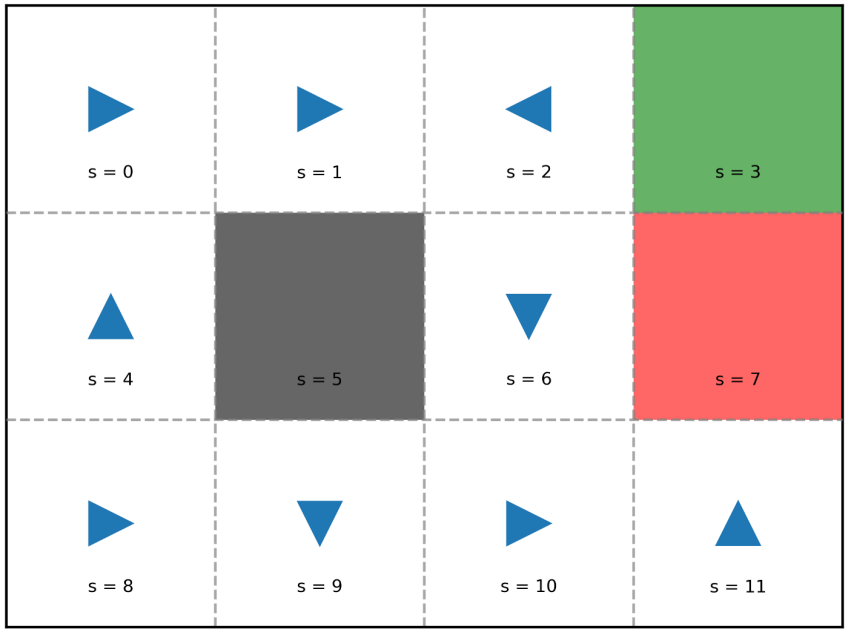
Explain about calculation of utility of states in the absence of discount factor in deterministic world (Slides 25-27)

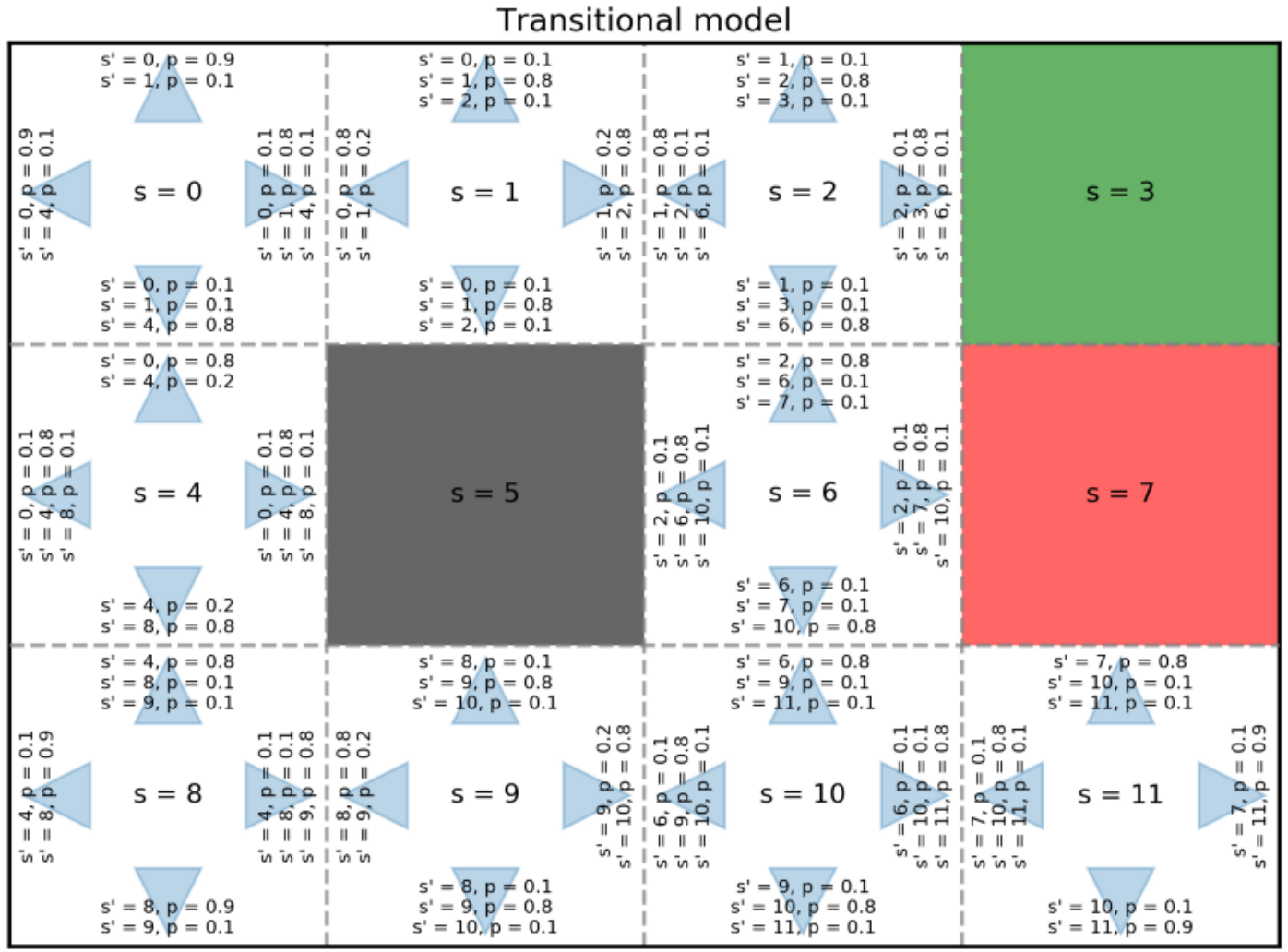
Why we need discount factor? Explain about calculation of utility of states in the presence of discount factor in deterministic world (Slides 28-29)

Explain about calculation of utility of states in the presence of discount factor in stochastic world (Slide 33)

Important Questions

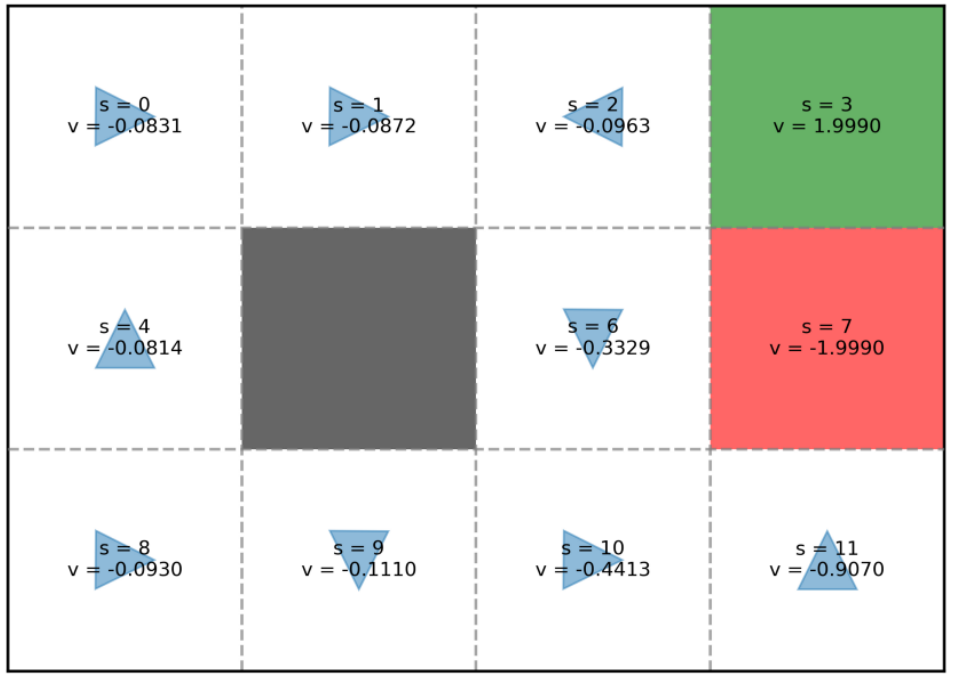
- Final the utility value of each state in the give grid world problem after three sweeps using policy evaluation (Slides 34 - 36)

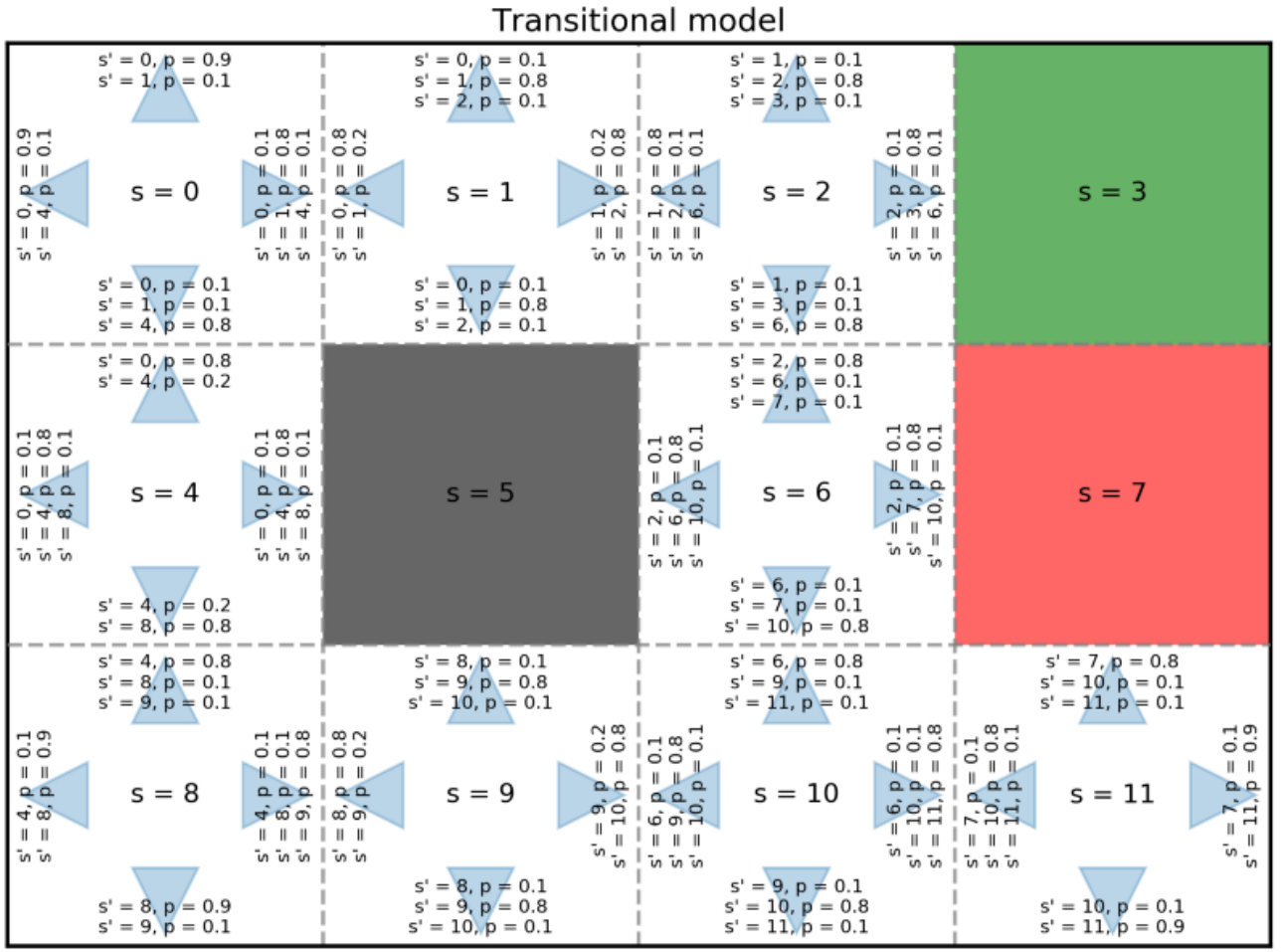




Important Questions

- Improve the action for state 2 and 8 using policy improvement (slides 38-43)



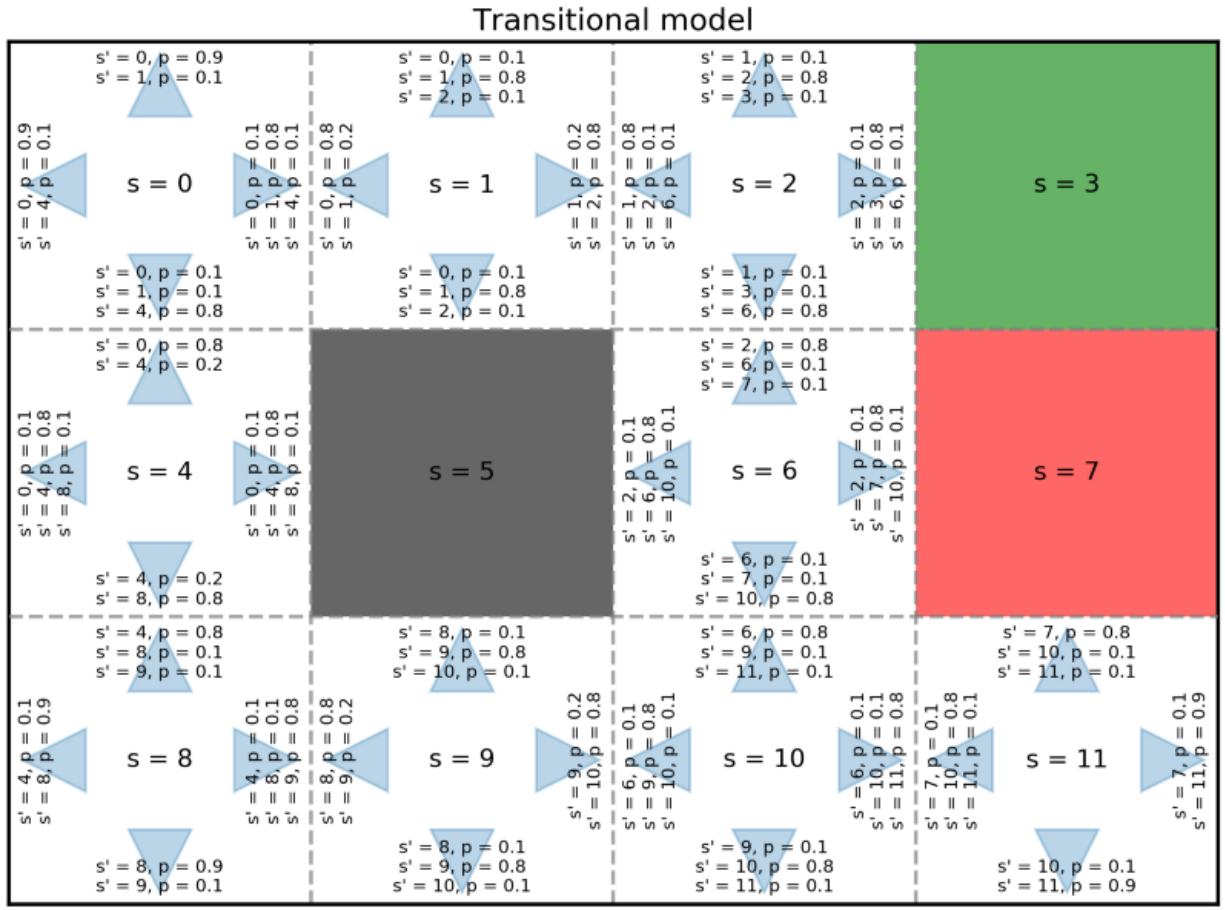


Important Questions

- What is the impact of discount factor while finding optimal policy using policy iteration (Slides 47-50)
- Explain policy evaluation using flowchart or pseudo code (slide 51)
- Explain policy improvement using flowchart or pseudo code (slide 52)
- Explain policy iteration using flowchart or pseudo code (slide 53)
- Define optimum policy in grid world game
- Explain procedure for optimal policy identification using value iteration using flowchart or pseudo code (slide 58)

Important Question

- Calculate the utility values of each state in 3X4 grid world problem after one iteration using value iteration method



References

- https://medium.com/@ngao7/markov-decision-process-policy-iteration-42d35ee87c82
- https://medium.com/@ngao7/markov-decision-process-basics-3da5144d3348#c25d
- https://medium.com/@ngao7/markov-decision-process-value-iteration-2d161d50a6ff