# 7
# Reinforcement Learning

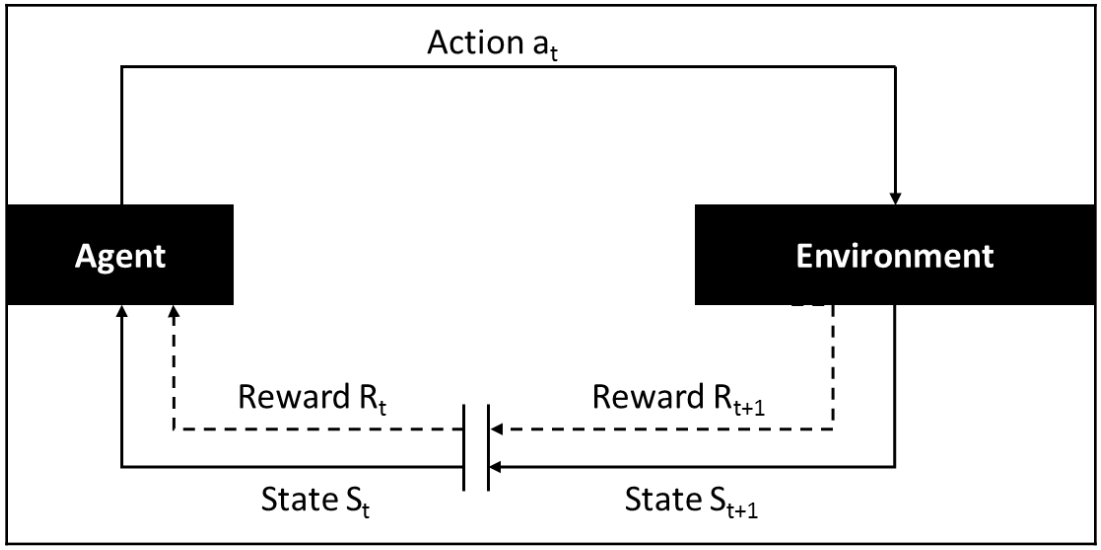
The current chapter will introduce Reinforcement Learning. We will cover the following topics:

- Setting up a Markov Decision Process
- Performing model-based learning
- Performing model-free learning

## Introduction

**Reinforcement Learning** (**RL**) is an area in machine learning that is inspired by psychology, such as how agents (software programs) can take actions in order to maximize cumulative rewards.

The RL is reward-based learning where the reward comes at the end or is distributed during the learning. For example, in chess, the reward will be assigned to winning or losing the game whereas in games such as tennis, every point won is a reward. Some of the commercial examples of RL are DeepMind from Google uses RL to master parkour. Similarly, Tesla is developing AI-driven technology using RL. An example of reinforcement architecture is shown in the following figure:




Interaction of an agent with environment in Reinforcement Learning

The basic notations for RL are as follows:

- **T(s, a, s')**: Represents the transition model for reaching state $s'$ when action $a$ is taken at state $s$
- $\boldsymbol{P}$: Represents a policy which defines what action to take at every possible state $s \in S$
- **R(s)**: Denotes the reward received by agent at state $s$

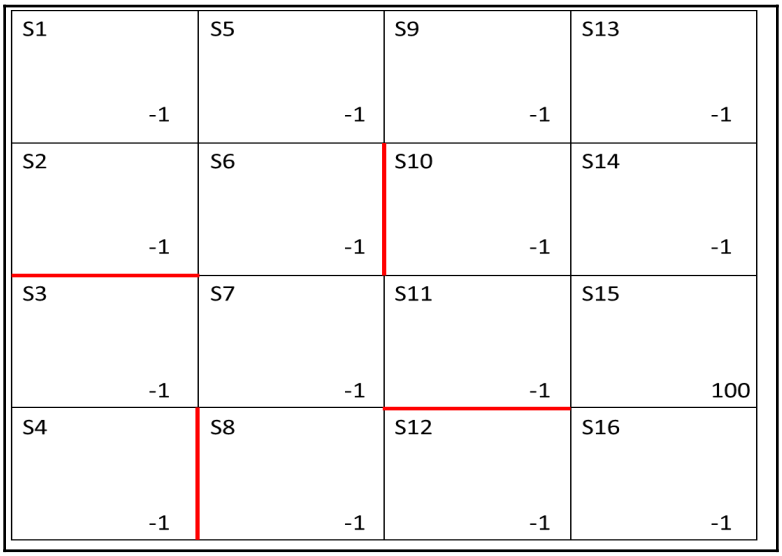
The current chapter will look at how to set up reinforcement models using R. The next sub-section will introduce `MDPtoolbox` from R.

# Setting up a Markov Decision Process

The **Markov Decision Process** (**MDP**) forms the basis of setting up RL, where the outcome of a decision is semi-controlled; that is, it is partly random and partly controlled (by the decision-maker). An MDP is defined using a set of possible states (**S**), a set of possible actions (**A**), a real-values reward function (**R**), and a set of transition probabilities from one state to another state for a given action (**T**). In addition, the effects of an action performed on one state depends only on that state and not on its previous states.

# Getting ready

In this section, let us define an agent travelling across a 4 x 4 grid, as shown in following figure:




A sample 4 x 4 grid of 16 states

This grid has 16 states (*S1*, *S2*....*S16*). In each state, the agent can perform four actions (*up*, *right*, *down*, *left*). However, the agent will be restricted to some actions based on the following constraints:

- The states across the edges shall be restricted to actions which point only toward states in the grid. For example, an agent in *S1* is restricted to the *right* or *down* action.

- Some state transitions have barriers, marked in red. For example, the agent cannot go *down* from *S2* to *S3*.

Each state is also assigned to a reward. The objective of the agent is to reach the destination with minimum moves, thereby achieving the maximum reward. Except state *S15* with a reward value of 100, all the remaining states have a reward value of *-1*.

Here, we will use the `MDPtoolbox` package in R.

# How to do it...

This section will show you how to set up RL models using `MDPtoolbox` in R:

1. Install and load the required package:

```
Install.packages("MDPtoolbox")
library(MDPtoolbox)
```

2. Define the transition probabilities for action. Here, each row denotes `from state` and each column denotes `to state`. As we have 16 states, the transition probability matrix of each action shall be a 16 x 16 matrix, with each row adding upto 1:

```
up<- matrix(c(1      ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                0.7   ,     0.2   ,     0     ,     0     ,     0
,     0.1   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0.8   ,     0.05  ,     0
,     0   ,     0.15  ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0.7   ,     0.3   ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                0.1   ,     0     ,     0     ,     0     ,     0.7
,     0.1   ,     0     ,     0     ,     0.1   ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                0     ,     0.05  ,     0     ,     0     ,     0.7
,     0.15    ,     0.1   ,     0     ,     0     ,     0     ,
0     ,     0     ,   0     ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0.05  ,     0     ,     0
,     0.7   ,     0.15  ,     0.05  ,     0     ,     0     ,
0.05  ,     0     ,   0     ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
```

```
,     0   ,     0.7   ,     0.2   ,     0     ,     0     ,     0
,     0.1   ,   0     ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0.05
,     0   ,     0     ,     0     ,     0.85  ,     0.05  ,     0
,     0     ,   0.05  ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0.7   ,     0.2   ,
0.05  ,     0     ,   0     ,     0.05  ,     0     ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0.05  ,     0     ,     0     ,     0.7   ,     0.2
,     0     ,   0     ,     0     ,     0.05  ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0.05  ,     0     ,     0     ,     0
,     0.9   ,   0     ,     0     ,     0     ,     0.05  ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0.1   ,     0     ,     0
,     0     ,   0.9   ,     0     ,     0     ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0.1   ,     0
,     0     ,   0.7   ,     0.2   ,     0     ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,
0.05  ,     0     ,   0     ,     0.8   ,     0.15  ,     0     ,
                0     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0.8   ,     0.2   ),
nrow=16, ncol=16, byrow=TRUE)
left<- matrix(c(1     ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                 0.05       ,     0.9   ,     0     ,     0     ,
0     ,   0.05  ,     0     ,     0     ,     0     ,     0     ,
0     ,     0   ,     0     ,     0     ,     0     ,     0     ,
                 0   ,     0     ,     0.9   ,     0.05  ,     0
,     0   ,     0.05  ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                 0   ,     0     ,     0.05  ,     0.9   ,     0
,     0   ,     0     ,     0.05  ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                 0.8 ,     0     ,     0     ,     0     ,     0.1
,     0.05   ,     0     ,     0     ,     0.05  ,     0     ,
0     ,     0     ,   0     ,     0     ,     0     ,     0     ,
                 0   ,     0.8   ,     0     ,     0     ,     0.05
,     0.1   ,     0.05  ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                 0   ,     0     ,     0.8   ,     0     ,     0
,     0.05   ,     0.1   ,     0.05  ,     0     ,     0     ,
0     ,     0     ,   0     ,     0     ,     0     ,     0     ,
```

```
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0.1   ,     0.8   ,     0     ,     0     ,     0
,     0.1   ,   0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0.8
,     0   ,     0     ,     0     ,     0.1   ,     0.05  ,     0
,     0     ,   0.05  ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0.8   ,     0     ,     0     ,     0.05  ,     0.1   ,
0.05  ,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0.8   ,     0     ,     0     ,     0.1   ,     0.1
,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0.8   ,     0     ,     0     ,     0
,     0.2   ,   0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0.8   ,     0     ,     0
,     0     ,   0.2   ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0.8   ,     0
,     0     ,   0.05  ,     0.1   ,     0.05  ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0.8
,     0     ,   0     ,     0.05  ,     0.1   ,     0.05  ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0.8   ,   0     ,     0     ,     0.05  ,     0.15),
nrow=16, ncol=16, byrow=TRUE)
down<- matrix(c(0.1   ,     0.8   ,     0     ,     0     ,     0.1
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0.05      ,     0.9   ,     0     ,     0     ,
0     ,   0.05  ,     0     ,     0     ,     0     ,     0     ,
0     ,     0   ,     0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0.1   ,     0.8   ,     0
,     0   ,     0.1   ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0.1   ,     0.9   ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0.05      ,     0     ,     0     ,     0     ,
0.15  ,   0.8   ,     0     ,     0     ,     0     ,     0     ,
0     ,     0   ,     0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0.2   ,     0.8   ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0.2   ,     0.8   ,     0     ,     0     ,     0
```

```
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0.1   ,     0.9   ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0.05
,     0   ,     0     ,     0     ,     0.1   ,     0.8   ,     0
,     0     ,   0.05  ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0.2   ,     0.8
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0.05  ,     0.8
,     0     ,   0     ,     0     ,     0.05  ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0.05  ,     0     ,     0     ,     0
,     0.9   ,   0     ,     0     ,     0     ,     0.05  ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0.2   ,     0.8   ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0.05  ,     0.15  ,     0.8   ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0.2   ,     0.8   ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     1),
nrow=16, ncol=16, byrow=TRUE)
right<- matrix(c(0.2 ,     0.1   ,     0     ,     0     ,     0.7
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                0.1     ,     0.1   ,     0     ,     0     ,
0     ,   0.8   ,     0     ,     0     ,     0     ,     0     ,
0     ,     0   ,     0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0.2   ,     0     ,     0
,     0   ,     0.8   ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0.1   ,     0.9   ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0.2
,     0.1   ,     0     ,     0     ,     0.7   ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
,     0.9   ,     0.1   ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     0     ,
                  0   ,     0     ,     0     ,     0     ,     0
```

```
,     0.05   ,     0.1   ,     0     ,     0     ,     0     ,
0.85  ,     0     ,   0     ,     0     ,     0     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0.1   ,     0.2   ,     0     ,     0     ,     0
,     0.7   ,   0     ,     0     ,     0     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0.2   ,     0     ,     0
,     0     ,   0.8   ,     0     ,     0     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0.1   ,     0
,     0     ,   0     ,     0.9   ,     0     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0.1
,     0     ,   0     ,     0     ,     0.9   ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0.2   ,   0     ,     0     ,     0     ,     0.8   ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   1     ,     0     ,     0     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     1     ,     0     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     1     ,     0     ,
                    0  ,     0     ,     0     ,     0     ,     0
,     0   ,     0     ,     0     ,     0     ,     0     ,     0
,     0     ,   0     ,     0     ,     0     ,     1),
nrow=16, ncol=16, byrow=TRUE)
```

3. Define a list of transition probability matrices:

```
TPMs <- list(up=up, left=left,
down=down, right=right)
```

4. Define a reward matrix of dimensions: 16 (number of states) x 4 (number of actions):

```
Rewards<- matrix(c(-1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
              -1, -1, -1, -1,
```

```
                    -1, -1, -1, -1,
                    -1, -1, -1, -1,
                    -1, -1, -1, -1,
                    -1, -1, -1, -1,
                    -1, -1, -1, -1,
                    100, 100, 100, 100,
                    -1, -1, -1, -1),
nrow=16, ncol=4, byrow=TRUE)
```

5. Test whether the defined `TPMs` and `Rewards` satisfy a well-defined MDP. If it returns an empty string, then the MDP is valid:

```
mdp_check(TPMs, Rewards)
```

# Performing model-based learning

As the name suggests, the learning is augmented using a predefined model. Here, the model is represented in the form of transition probabilities and the key objective is to determine the optimal policy and value functions using these predefined model attributes (that is, `TPMs`). The policy is defined as a learning mechanism of an agent, traversing across multiple states. In other words, identifying the best action of an agent in a given state, to traverse to a next state, is termed a policy.

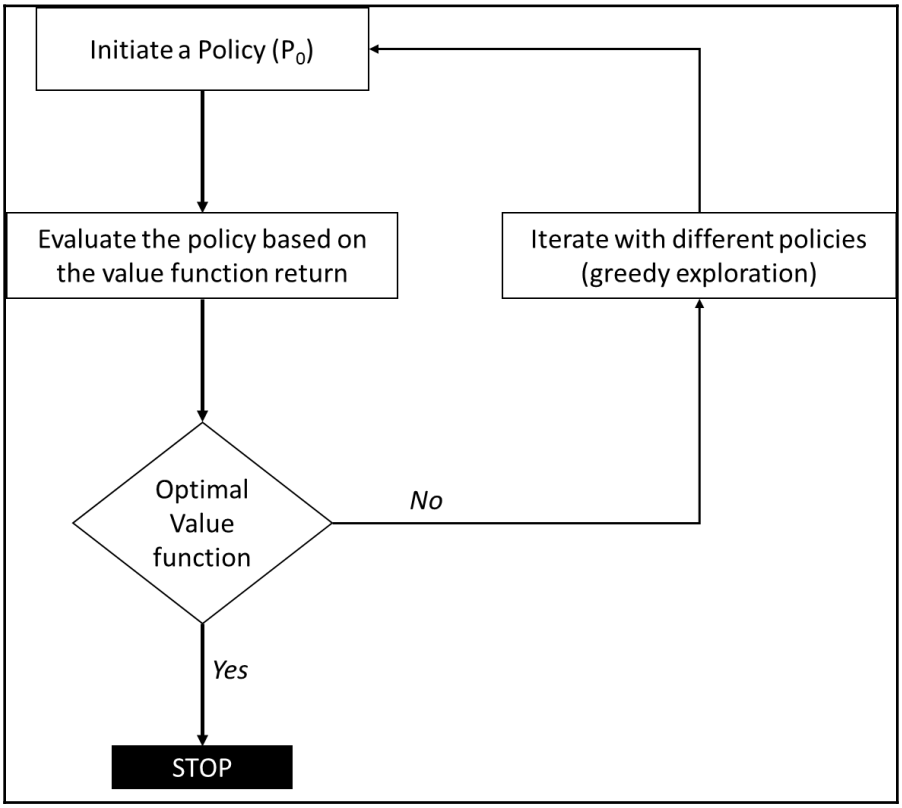
The objective of the policy is to maximize the cumulative reward of transitioning from the start state to the destination state, defined as follows, where *P(s)* is the cumulative policy *P* from a start state *s*, and *R* is the reward of transitioning from state *st* to state $s_{t+1}$ by performing an action at.

$$P(s) = Max[\sum_{t} R^{a_t}_{s_t s_{t+1}}]$$

The value function is of two types: the state-value function and the state-action value function. In the state-value function, for a given policy, it is defined as an expected reward to be in a particular state (including start state), whereas in the state-action value function, for a given policy, it is defined as an expected reward to be in a particular state (including the start state) and undertake a particular action.

Now, a policy is said to be optimal provided it returns the maximum expected cumulative reward, and its corresponding states are termed optimal state-value functions or its corresponding states and actions are termed optimal state-action value functions.

In model-based learning, the following iterative steps are performed in order to obtain an optimum policy, as shown in the following figure:




Iterative steps to find an optimum policy

In this section, we shall evaluate the policy using the state-value function. In each iteration, the policies are dynamically evaluated using the Bellman equation, as follows, where $V_i$ denotes the value at iteration $i$, $P$ denotes an arbitrary policy of a given state $s$ and action $a$, $T$ denotes the transition probability from state $s$ to state $s'$ due to an action $a$, $R$ denotes the reward at state $s'$ while traversing from the state $s$ post an action $a$, and $\Upsilon$ denotes a discount factor in the range of (0,1). The discount factor ensures higher importance to starting learning steps than later.

$$V_{i+1}(s) = \sum_{a} P(s,a) \sum_{s'} T^{a}_{ss'}[R^{a}_{ss'} + \Upsilon V_i(s')]$$

# How to do it...

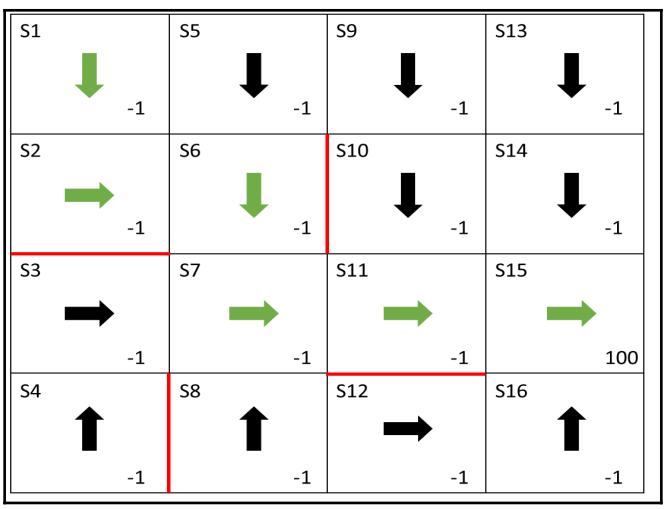
This section shows you how to set up model-based RL:

1. Run the policy iteration using the state-action value function with the discount factor $\Upsilon = 0.9$:

```
mdp_policy<- mdp_policy_iteration(P=TPMs, R=Rewards, discount=0.9)
```

2. Get the best (optimum) policy P* as shown in the following figure. The arrows marked in green show the direction of traversing *S1* to *S15*:

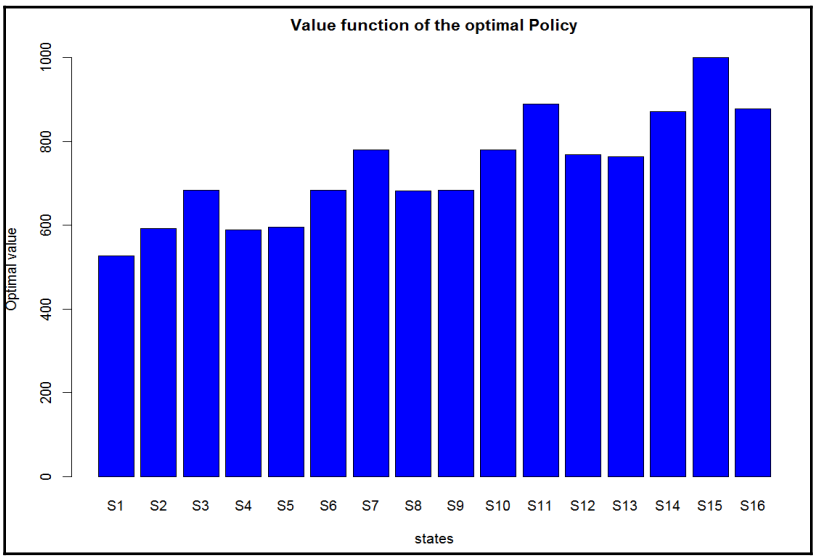
```
mdp_policy$policy
names(TPMs)[mdp_policy$policy]
```




Optimum policy using model-based iteration with an optimum path from *S1* to *S15*

3. Get the optimum value function V* for each state and plot them as shown in the following figure:

```
mdp_policy$V
names(mdp_policy$V) <- paste0("S",1:16)
barplot(mdp_policy$V,col="blue",xlab="states",ylab="Optimal
value",main="Value function of the optimal Policy",width=0.5)
```




Value functions of the optimal policy

# Performing model-free learning

Unlike model-based learning, where dynamics of transitions are explicitly provided (as transition probabilities from one state to another state), in model-free learning, the transitions are supposed to be deduced and learned directly from the interaction between states (using actions) rather explicitly provided. Widely used frameworks of mode-free learning are **Monte Carlo** methods and the **Q-learning** technique. The former is simple to implement but convergence takes time, whereas the latter is complex to implement but is efficient in convergence due to off-policy learning.

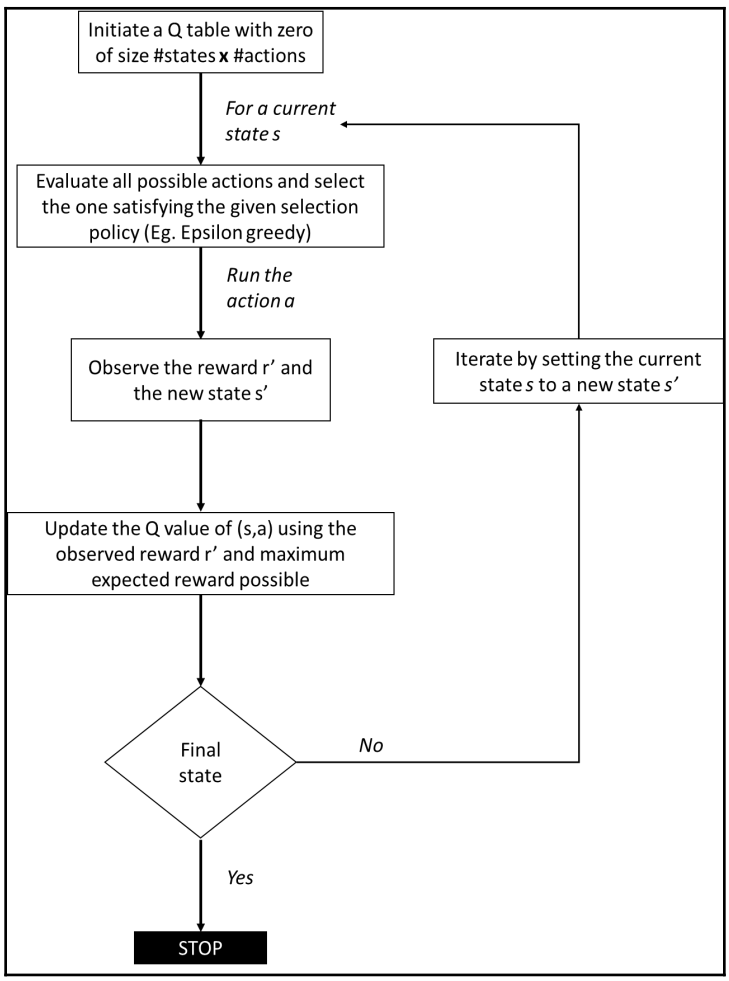
# Getting ready

In this section, we will implement the Q-learning algorithm in R. The simultaneous exploration of the surrounding environment and exploitation of existing knowledge is termed off-policy convergence. For example, an agent in a particular state first explores all the possible actions of transitioning into next states and observes the corresponding rewards, and then exploits current knowledge to update the existing state-action value using the action generating the maximum possible reward.

The Q learning returns a 2D Q-table of the size of the number of states x the number of actions. The values in the Q-table are updated based on the following formula, where $Q$ denotes the value of state $s$ and action $a$, $r'$ denotes the reward of the next state for a selected action $a$, $\Upsilon$ denotes the discount factor, and $\alpha$ denotes the learning rate:

$$Q_{new}(s,a) = Q_{old}(s,a) + \alpha\left[r' + \Upsilon * \max_{a'} Q_{expected\ optimal\ value}(s',a') - Q_{old}(s,a)\right]$$

The framework for Q-learning is shown in the following figure:




Framework of Q-learning

# How to do it...

The section provide steps for how to set up Q-learning:

1. Define 16 states:

```
states <- c("s1", "s2", "s3", "s4", "s5", "s6", "s7", "s8", "s9",
"s10", "s11", "s12", "s13", "s14", "s15", "s16")
```

2. Define four actions:

```
actions<- c("up", "left", "down", "right")
```

3. Define the `transitionStateAction` function, which can simulate the transitions from one state *s* to another state *s'* using an action *a*. The function takes in the current state *s* and selected action *a*, and it returns the next state *s'* and corresponding reward *r'*. In case of constrained action, the next state returned is the current state *s* and the existing reward *r*:

```
transitionStateAction<- function(state, action) {
  # The default state is the existing state in case of constrained
action
next_state<- state
if (state == "s1"&& action == "down") next_state<- "s2"
if (state == "s1"&& action == "right") next_state<- "s5"
if (state == "s2"&& action == "up") next_state<- "s1"
if (state == "s2"&& action == "right") next_state<- "s6"
if (state == "s3"&& action == "right") next_state<- "s7"
if (state == "s3"&& action == "down") next_state<- "s4"
if (state == "s4"&& action == "up") next_state<- "s3"
if (state == "s5"&& action == "right") next_state<- "s9"
if (state == "s5"&& action == "down") next_state<- "s6"
if (state == "s5"&& action == "left") next_state<- "s1"
if (state == "s6"&& action == "up") next_state<- "s5"
if (state == "s6"&& action == "down") next_state<- "s7"
if (state == "s6"&& action == "left") next_state<- "s2"
if (state == "s7"&& action == "up") next_state<- "s6"
if (state == "s7"&& action == "right") next_state<- "s11"
if (state == "s7"&& action == "down") next_state<- "s8"
if (state == "s7"&& action == "left") next_state<- "s3"
if (state == "s8"&& action == "up") next_state<- "s7"
if (state == "s8"&& action == "right") next_state<- "s12"
if (state == "s9"&& action == "right") next_state<- "s13"
if (state == "s9"&& action == "down") next_state<- "s10"
if (state == "s9"&& action == "left") next_state<- "s5"
if (state == "s10"&& action == "up") next_state<- "s9"
if (state == "s10"&& action == "right") next_state<- "s14"
```

```
if (state == "s10"&& action == "down") next_state<- "s11"
if (state == "s11"&& action == "up") next_state<- "s10"
if (state == "s11"&& action == "right") next_state<- "s15"
if (state == "s11"&& action == "left") next_state<- "s7"
if (state == "s12"&& action == "right") next_state<- "s16"
if (state == "s12"&& action == "left") next_state<- "s8"
if (state == "s13"&& action == "down") next_state<- "s14"
if (state == "s13"&& action == "left") next_state<- "s9"
if (state == "s14"&& action == "up") next_state<- "s13"
if (state == "s14"&& action == "down") next_state<- "s15"
if (state == "s14"&& action == "left") next_state<- "s10"
if (state == "s15"&& action == "up") next_state<- "s14"
if (state == "s15"&& action == "down") next_state<- "s16"
if (state == "s15"&& action == "left") next_state<- "s11"
if (state == "s16"&& action == "up") next_state<- "s15"
if (state == "s16"&& action == "left") next_state<- "s12"
  # Calculate reward
if (next_state == "s15") {
reward<- 100
  } else {
reward<- -1
  }

return(list(state=next_state, reward=reward))
}
```

4. Define a function to perform Q-learning using `n` iterations:

```
Qlearning<- function(n, initState, termState,
epsilon, learning_rate) {
  # Initialize a Q-matrix of size #states x #actions with zeroes
Q_mat<- matrix(0, nrow=length(states), ncol=length(actions),
dimnames=list(states, actions))
  # Run n iterations of Q-learning
for (i in 1:n) {
Q_mat<- updateIteration(initState, termState, epsilon,
learning_rate, Q_mat)
  }
return(Q_mat)
}
   updateIteration<- function(initState, termState, epsilon,
learning_rate, Q_mat) {
state<- initState # set cursor to initial state
while (state != termState) {
    # Select the next action greedily or randomnly
if (runif(1) >= epsilon) {
action<- sample(actions, 1) # Select randomnly
    } else {
```

```
action<- which.max(Q_mat[state, ]) # Select best action
    }
    # Extract the next state and its reward
response<- transitionStateAction(state, action)
    # Update the corresponding value in Q-matrix (learning)
Q_mat[state, action] <- Q_mat[state, action] + learning_rate *
      (response$reward + max(Q_mat[response$state, ]) -
Q_mat[state, action])
state<- response$state # update with next state
  }
return(Q_mat)
}
```

5. Set learning parameters such as `epsilon` and `learning_rate`:

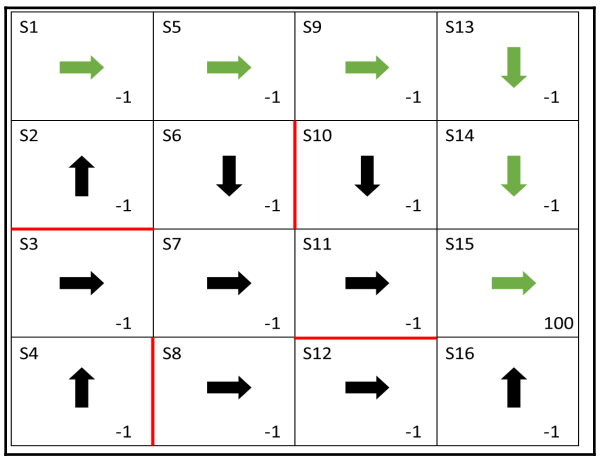
```
epsilon<- 0.1
learning_rate<- 0.9
```

6. Get the Q-table after 500k iterations:

```
Q_mat<- Qlearning(500, "s1", "s15", epsilon, learning_rate)
Q_mat
```

7. Get the best (optimum) policy P*, as shown in the following figure. The arrows marked in green shows the direction of traversing *S1* to *S15:*

```
actions[max.col(Q_mat)]
```




Optimum policy using model-free iteration with an optimum path from *S1* to *S15*