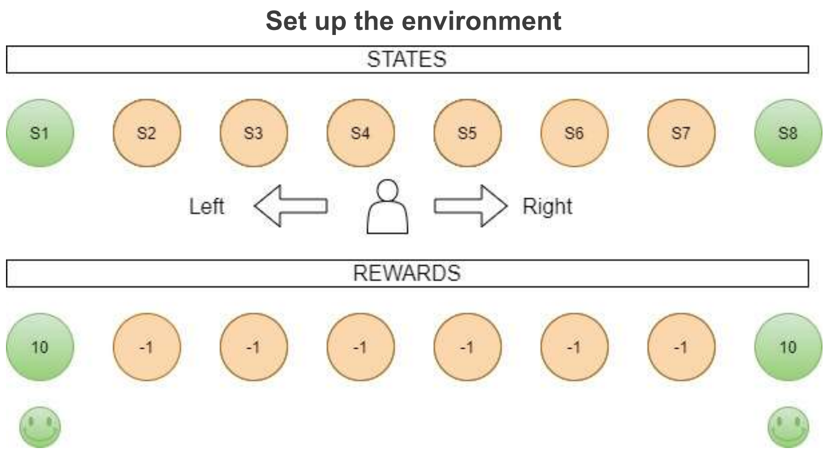
Creating a simple MDP MATLAB Environment with a Q learning Agent

The topics covered in this introducotry module are:

1. Creating and loading an MDP MATLAB Environment
2. Extract and modify Reward and Transition probability matrix.
3. Define and initialize a Q-learning agent.
4. Define training parameters
5. Extract Q-table and see how the values are updated according to Bellman Equation.
6. Completely Train a Q-learning agent

The list of RL toolbox functions used in this module is given in the end.



The model consists of 8 states with 2 terminal states [S1 and S8]. The goal of the agent is to reach to the terminal by taking an Action [either a left of a right].

Creating an MDP Matlab Environment.

We start with creating an envinronment using function [`createMDP(states,actions)`](),which takes in 2 inputs.

The properties of the environment can be accessed by 'dot' notation.

```
MDP = createMDP(8,["left";"right"])
```

```
MDP = 
  GenericMDP with properties:

      CurrentState: "s1"
            States: [8×1 string]
           Actions: [2×1 string]
                 T: [8×8×2 double]
                 R: [8×8×2 double]
    TerminalStates: [0×1 string]
```

CurrentState — Name of the current state string Name of the current state, specified as a string.

States — State names string vector State names, specified as a scalar 'states'

Actions — Action names string vector Action names, specified as a string vector.

T — State transition matrix 3D array State transition matrix, specified as a 3-D array, which determines the possible movements of the agent in an environment. State transition matrix T is a probability matrix that indicates how likely the agent will move from the current state s to any possible next state s' by performing action a. T is given by, T s, s′, a = probability s′ s, a

R — Reward transition matrix 3D array Reward transition matrix, specified as a 3-D array, determines how much reward the agent receives after performing an action in the environment. R has the same shape and size as state transition matrix T. Reward transition matrix R is given by, r = R s, s′, a .

TerminalStates — Terminal state names in the grid world string vector Terminal state names in the grid world, specified as a string vector.

```
MDP.States
```

```
ans = 8×1 string
"s1"
"s2"
"s3"
"s4"
"s5"
"s6"
"s7"
"s8"
```

```
MDP.Actions
```

```
ans = 2×1 string
```

```
"left"
"right"
```

```
MDP.T
```

```
ans = 
ans(:,:,1) =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0


ans(:,:,2) =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

```
MDP.R
```

```
ans = 
ans(:,:,1) =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0


ans(:,:,2) =

     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Define Rewards and Transition Probabilities as per the model

```
MDP.TerminalStates = ["s1";"s8"];

nS = numel(MDP.States);
nA = numel(MDP.Actions);

MDP.R = -1*ones(nS,nS,nA);
MDP.R(:,state2idx(MDP,MDP.TerminalStates),:) = 10;
```

```
% State 1 transition
MDP.T(1,1,1) = 1;
MDP.T(1,2,2) = 1;

% State 2 transition
MDP.T(2,1,1) = 1;
MDP.T(2,3,2) = 1;

% State 3 transition
MDP.T(3,2,1) = 1;
MDP.T(3,4,2) = 1;

% State 4 transition
MDP.T(4,3,1) = 1;
MDP.T(4,5,2) = 1;

% State 5 transition
MDP.T(5,4,1) = 1;
MDP.T(5,6,2) = 1;

% State 6 transition
MDP.T(6,5,1) = 1;
MDP.T(6,7,2) = 1;

% State 7 transition
MDP.T(7,6,1) = 1;
```

```
MDP.T(7,8,2) = 1;

% State 8 transition
MDP.T(8,7,1) = 1;
MDP.T(8,8,2) = 1;
```

```
MDP.T
```

```
ans = 
ans(:,:,1) =

     1     0     0     0     0     0     0     0
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0


ans(:,:,2) =

     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
```

```
MDP.R
```

```
ans = 
ans(:,:,1) =

    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10


ans(:,:,2) =

    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
    10    -1    -1    -1    -1    -1    -1    10
```

We use rlMDPEnv to create an environment using the MDP object. `rlMDPEnv(MDP)`

```
env = rlMDPEnv(MDP)
```

```
env = 
  rlMDPEnv with properties:

       Model: [1×1 rl.env.GenericMDP]
    ResetFcn: []
```

Define Q-table and Initialize Agent

Task: Display the information of States in the environment

Function: `getObservationInfo(env)`

Input: An RL Environment object

`Usage:` Obtain OBSERVATION/STATE data specifications from reinforcement learning environment or agent

```
%%%%% Start your code here %%%% ~ 1 line
state_information = getObservationInfo(env)
```

```
state_information = 
  rlFiniteSetSpec with properties:

       Elements: [8×1 double]
           Name: "MDP Observations"
    Description: [0×0 string]
      Dimension: [1 1]
       DataType: "double"
```

```
%%%%state_information% Code ends here %%%%
```

Task: Display the information of Actions in the environment

Function: getActionInfo(env)

Input: An RL Environment object

Usage: Obtain ACTION data specifications from reinforcement learning environment or agent

```
%%%%% Start your code here %%%% ~ 1 line
action_information = getActionInfo(env)
```

```
action_information =
  rlFiniteSetSpec with properties:

       Elements: [2×1 double]
           Name: "MDP Actions"
    Description: [0×0 string]
      Dimension: [1 1]
       DataType: "double"
```

```
%%%%% Code ends here %%%%
```

We want to make use of Q-learning agent. A Q-learning agent is represented by observations from the environment (state) and action available.

Task: first create a Q table using the observation in a varible name *qTable*

Function: rlTable(obsinfo,actinfo)

Inputs: Number of states **(line 48)** and Number of actions (**line 50)**

```
%%%%% Start your code here %%%% ~ 1 line
qTable = rlTable(state_information,action_information)
```

```
qTable =
  rlTable with properties:

    Table: [8×2 double]
```

```
%%%%% Code ends here %%%%
```

Task: Display the Q -table

```
%%%%% Start your code here %%%% ~ 1 line
qTable.Table
```

```
ans = 8×2
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
     0     0
```

```
%%%%% Code ends here %%%%
```

Task: Change all the values of q-table to 5

```
%%%%% Start your code here %%%% ~ 1 line
qTable.Table = ones(size(qTable.Table))*5
```

```
qTable =
  rlTable with properties:

    Table: [8×2 double]
```

```
qTable.Table
```

```
ans = 8×2
     5     5
     5     5
     5     5
     5     5
     5     5
     5     5
     5     5
     5     5
```

```
%%%%% Code ends here %%%%
```

Task: Create the Q-value function based `critic` (Brain of the agent) named as *qRepresentation*

Function: rlQValueRepresentation(tab,observationInfo,actionInfo)

Inputs:

1. q-value is a rlTable object containing a table with as many rows as the possible observations and as many columns as the possible actions (**line 54)**

2. State information **(line 48)**
3. Action Information **(line 50)**

Usage: Let MATLAB know that the table created will be used a Q-learning agent. Apply the Q-learning algorithm to this table while training.

```
%%%%% Start your code here %%%% ~ 1 line
qRepresentation = rlQValueRepresentation(qTable,state_information,action_information)
```

```
qRepresentation = 
  rlQValueRepresentation with properties:

         ActionInfo: [1×1 rl.util.rlFiniteSetSpec]
    ObservationInfo: [1×1 rl.util.rlFiniteSetSpec]
            Options: [1×1 rl.option.rlRepresentationOptions]
```

```
%%%%% Code ends here %%%%
```

Task: Display Options property of qRpresentation

```
%%%%% Start your code here %%%% ~ 1 line
qRepresentation.Options
```

```
ans = 
  rlRepresentationOptions with properties:

                  LearnRate: 0.0100
          GradientThreshold: Inf
    GradientThresholdMethod: "l2norm"
     L2RegularizationFactor: 1.0000e-04
                  UseDevice: "cpu"
                  Optimizer: "adam"
        OptimizerParameters: [1×1 rl.option.OptimizerParameters]
```

```
%%%%% Code ends here %%%%
```

During each control interval the agent selects a random action with probability ϵ, otherwise it selects an action greedily with respect to the value function with probability 1-ϵ. This greedy action is the action for which the value function is greatest.

```
qRepresentation.Options.L2RegularizationFactor=0;
qRepresentation.Options.LearnRate = 0.01;
```

Task: Create a variable named *agentOpts* to store and specifiy parameters for updating q-values

Function: `rlQAgentOptions`.

Assign agentOpts = function

Inputs: No inputs

```
%%%%% Start your code here %%%% ~ 1 line
agentOpts = rlQAgentOptions
```

```
agentOpts = 
  rlQAgentOptions with properties:

    EpsilonGreedyExploration: [1×1 rl.option.EpsilonGreedyExploration]
                  SampleTime: 1
              DiscountFactor: 0.9900
```

```
%%%%% Code ends here %%%%
```

Task: Display epsilon greeedy exploration property

```
%%%%% Start your code here %%%% ~ 1 line
agentOpts.EpsilonGreedyExploration
```

```
ans = 
  EpsilonGreedyExploration with properties:

    EpsilonDecay: 0.0050
         Epsilon: 1
      EpsilonMin: 0.0100
```

```
%%%%% Code ends here %%%%
```

Task: Change EpsilonDecay to 0.01

```
%%%%% Start your code here %%%% ~ 1 line
agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01
```

```
agentOpts = 
  rlQAgentOptions with properties:
```

```
    EpsilonGreedyExploration: [1×1 rl.option.EpsilonGreedyExploration]
                  SampleTime: 1
              DiscountFactor: 0.9900
```

```
%%%%% Code ends here %%%%
```

Task: create a Q-learning agent named *qAgent* using this Q-table representation and configuration of the epsilon-greedy exploration.

Function: `rlQAgent(critic,agentOptions)`

Inputs:

1. `critic (Brain if the agent)` **(line 63)**
2. `agentOpts (Learning and equation parameters)` **(line 70)**

```
%%%%% Start your code here %%%% ~ 1 line
qAgent = rlQAgent(qRepresentation,agentOpts)
```

```
qAgent = 
  rlQAgent with properties:

    AgentOptions: [1×1 rl.option.rlQAgentOptions]
```

```
%%%%% Code ends here %%%%
```

Train Agent

Task: specify the training parameters in a variable name *trainOpts*

Function: `rlTrainingOptions`.

```
%%%%% Start your code here %%%% ~ 1 line
trainOpts = rlTrainingOptions
```

```
trainOpts = 
  rlTrainingOptions with properties:

                   MaxEpisodes: 500
            MaxStepsPerEpisode: 500
    ScoreAveragingWindowLength: 5
          StopTrainingCriteria: "AverageSteps"
             StopTrainingValue: 500
             SaveAgentCriteria: "none"
                SaveAgentValue: "none"
            SaveAgentDirectory: "savedAgents"
                       Verbose: 0
                         Plots: "training-progress"
                   StopOnError: "on"
                   UseParallel: 0
```

```
%%%%% Code ends here %%%%
```

```
trainOpts.MaxStepsPerEpisode = 10;
trainOpts.MaxEpisodes = 100;
trainOpts.StopTrainingCriteria = "AverageReward";
trainOpts.StopTrainingValue = 13;
trainOpts.ScoreAveragingWindowLength = 30;
```

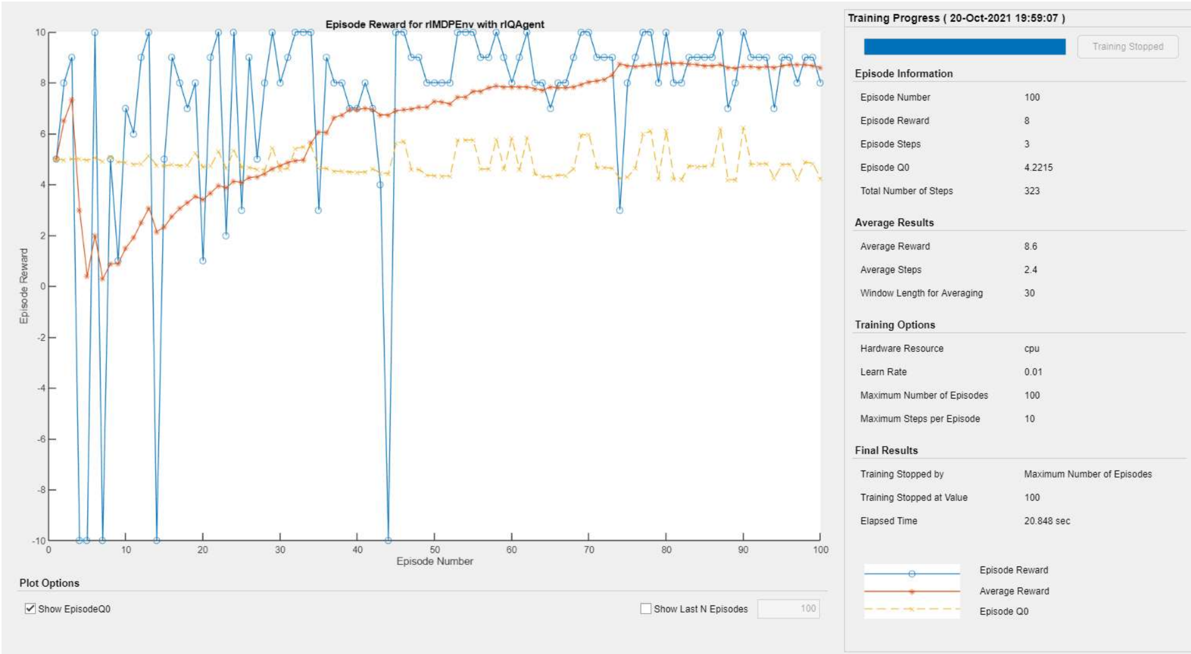
```
QTable0 = getLearnableParameters(getCritic(qAgent));
disp(QTable0{1})
```

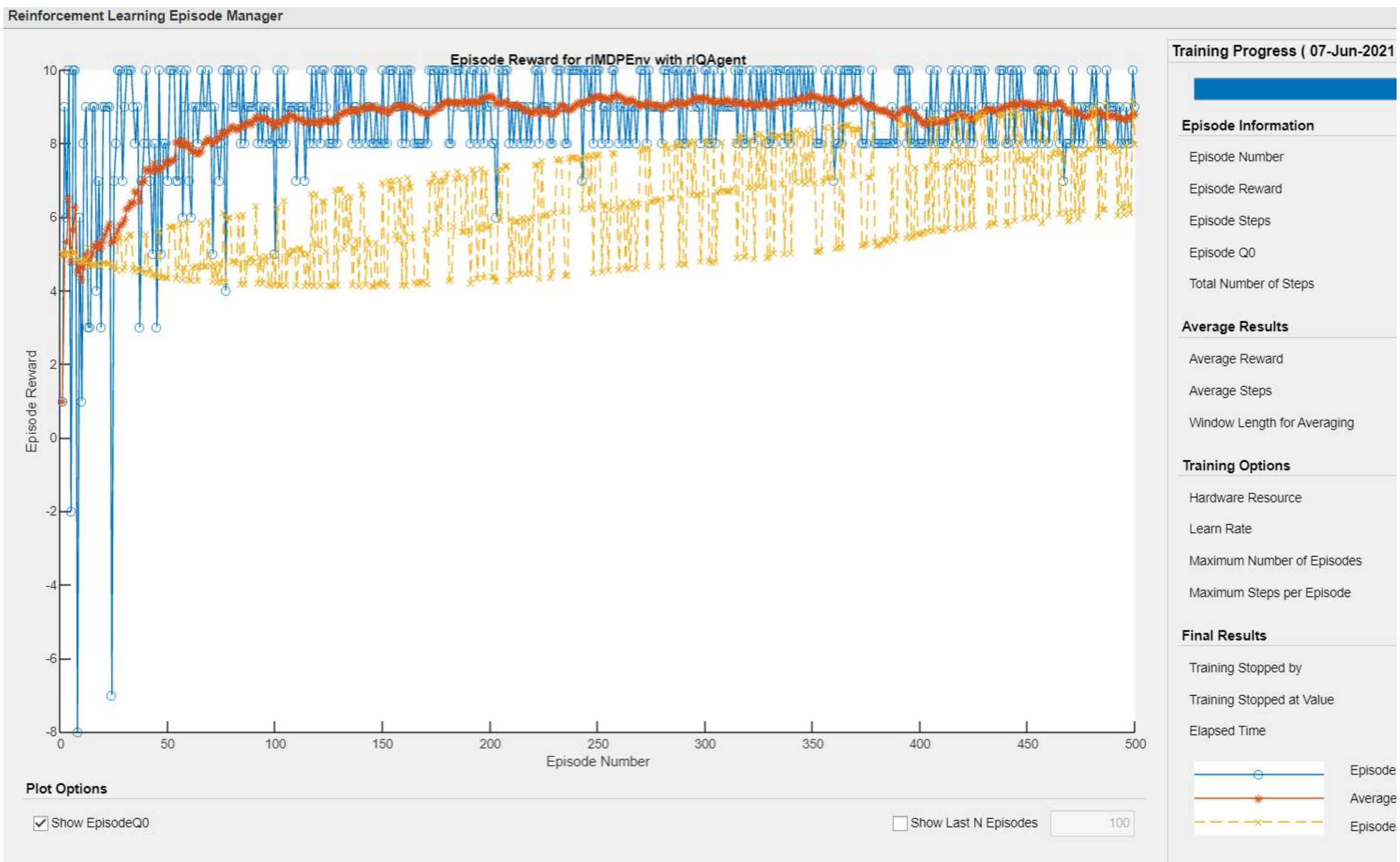
```
     5     5
     5     5
     5     5
     5     5
     5     5
     5     5
     5     5
     5     5
```

Begin Training

```
doTraining = true;

if doTraining
    % Train the agent.
    trainingStats = train(qAgent,env,trainOpts); %#ok<UNRCH>
else
    % Load pretrained agent for the example.
    load('genericMDPQAgent.mat','qAgent');
end
```





Store trained agent's properties in a varaible called Data. We do this to access and make use of the training solutions,

```
Data = sim(qAgent,env)
```

```
Data = struct with fields:
        Observation: [1×1 struct]
             Action: [1×1 struct]
             Reward: [1×1 timeseries]
             IsDone: [1×1 timeseries]
     SimulationInfo: [1×1 struct]
```

```
cumulativeReward = sum(Data.Reward)
```

```
cumulativeReward = 7
```

Obtain learnable parameter values from policy or value function representation: `getLearnableParameters(rep)`

`getLearnableParameters(rep)` returns the values of the learnable parameters from the reinforcement learning policy or value function representation `rep`.

```
QTable = getLearnableParameters(getCritic(qAgent));
QTable{1}
```

```
ans = 8×2
    5.0000    5.0000
    6.3718    4.8348
    4.9033    4.5692
    4.2215    4.2194
    4.1543    4.1466
    4.5253    4.8322
    4.6767    6.2420
    5.0000    5.0000
```