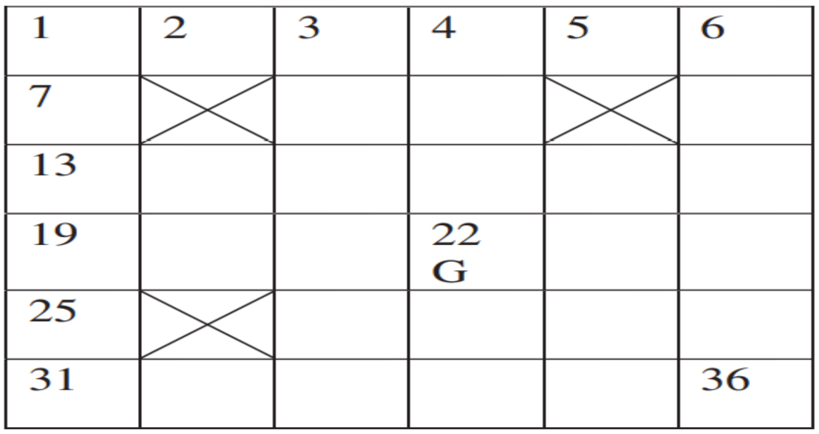
GRID WORLD PROBLEM

The grid considered is having 36 cells arranged in 6 rows and 6 columns. A robot can be at any one of the possible cells at any instant. We can refer the cell number as state of the robot. ‘G’ denotes the goal state to which the robot aim to reach and the crossed cells denote cells with some sort of obstacles. There is cost associated with each cell transition while the cost of passing through a cell with obstacle is much higher compared to other cells. Starting from any initial position in the grid, robot can reach the goal cell by following different paths and correspondingly cost incurred will also vary. The problem is to find an optimum path to reach the goal state (cell) starting from any one of the initial state.

In this simple grid world we assume a cost of 1 unit for transition to an ordinary cell while 10 units for a cell having some obstacles.



```
clear
```

First we need to create a 6*6 sized gridworld.

```
GW = createGridWorld(6,6)
```

```
GW = 
  GridWorld with properties:

          GridSize: [6 6]
      CurrentState: "[1,1]"
            States: [36×1 string]
           Actions: [4×1 string]
                 T: [36×36×4 double]
                 R: [36×36×4 double]
    ObstacleStates: [0×1 string]
    TerminalStates: [0×1 string]
```

Now, we need to initialise the current state of the robot, the terminal state and the states which are having obstacles.

```
GW.CurrentState = '[1,1]';
GW.TerminalStates = '[4,4]';
GW.ObstacleStates = ["[2,2]";"[2,5]";"[5,2]"];
```

```
nS = numel(GW.States) %Number of States
```

```
nS = 36
```

```
nA = numel(GW.Actions) %Number of Actions
```

```
nA = 4
```

The reward(here cost) associated with each action in the given state should be defined.

```
GW.R = -1*ones(nS,nS,nA); %Fill reward transition matrix with 1.
```

The cost of getting into a state having obstacle is higher(10).

```
GW.R(:,state2idx(GW,"[2,2]"),:) = -10;
GW.R(:,state2idx(GW,"[2,5]"),:) = -10;    %GW.R(s,s`,a)
GW.R(:,state2idx(GW,"[5,2]"),:) = -10;
```

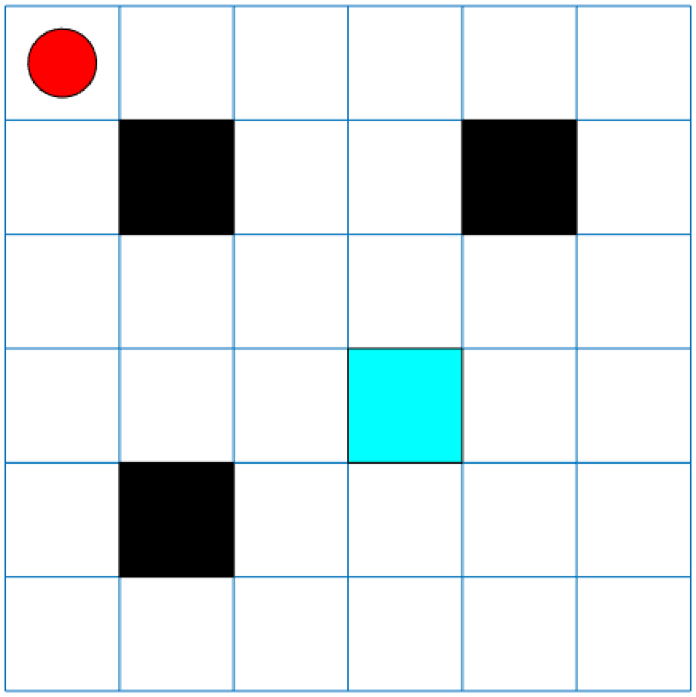
Now, we need to create a Reinforcement Learning MDP environment.

```
env = rlMDPEnv(GW)
```

```
env = 
  rlMDPEnv with properties:

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

```
plot(env)
```



```
env.ResetFcn = @() 1;
```

The above reset function is used to specify that after each training episode, the agent should start from the initial state i.e.[1,1]

```
qTable = rlTable(getObservationInfo(env),getActionInfo(env));
```

Create a Qtable by getting the observations and actions.

```
qRepresentation = rlQValueRepresentation(qTable,getObservationInfo(env),getActionInfo(env));
```

```
qRepresentation.Options.LearnRate = 1;
```

The learning rate is set to 1. Higher learning rate makes the learning slower, wheres as smaller learning rate could result in the convergence to a sub optimum value. So we have to carefully select the value of Learning rate.

```
agentOpts = rlQAgentOptions;
```

```
agentOpts.EpsilonGreedyExploration.Epsilon = .04;
```

To encourage exploration, we make use of epsilon-greedy algorithm. Epsilon value is set to 0.04.

```
qAgent= rlQAgent(qRepresentation,agentOpts);
```

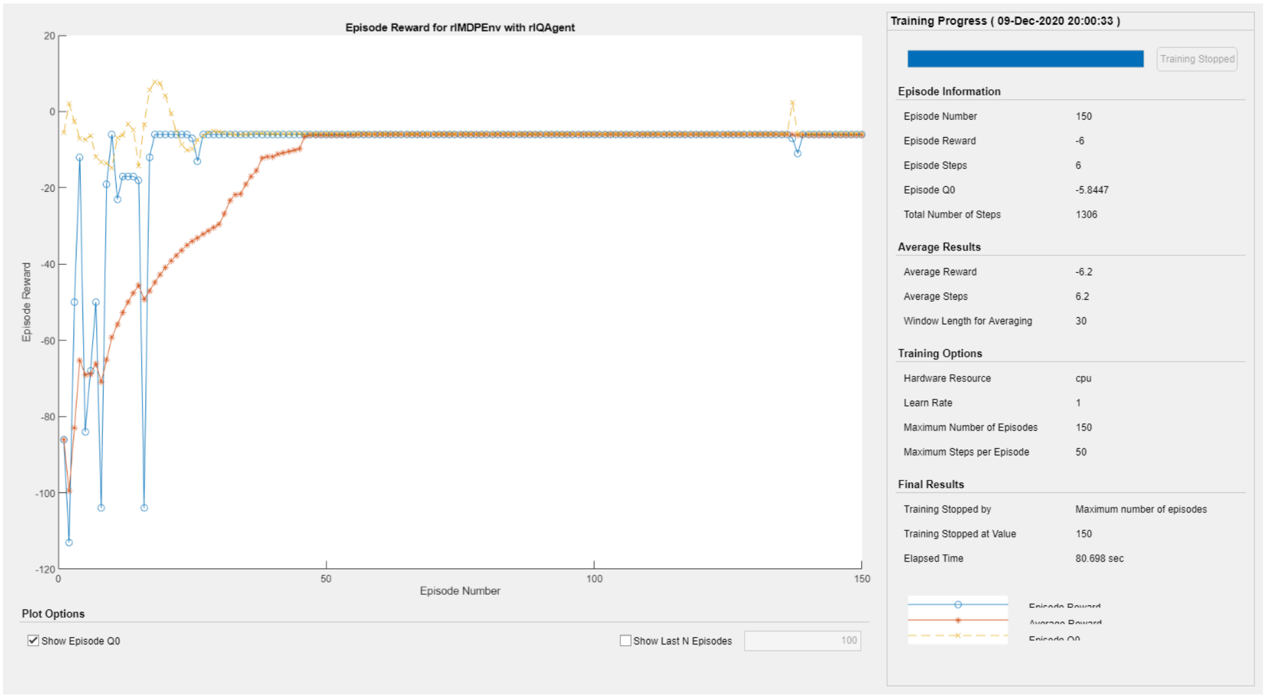
Agent is created with above code.

```
trainOpts = rlTrainingOptions;
trainOpts.MaxStepsPerEpisode = 50;
trainOpts.MaxEpisodes= 150;
trainOpts.StopTrainingCriteria = "AverageReward";
trainOpts.StopTrainingValue = 11;
trainOpts.ScoreAveragingWindowLength = 30; %Stop when the average reward for 30 steps consecuti
```

Parameters for training are set by the above code. The number of episodes for training is set to 150.

The training is done by making use of the below code.

```
doTraining = true;
if doTraining
 % Train the agent.
 trainingStats = train(qAgent,env,trainOpts);
% else
%  % Load the pretrained agent for the example.
%  load('basicGWQAgent.mat','qAgent')
end
```



Training Progress (09-Dec-2020 20:00:33)

Episode Information

Episode Number	150
Episode Reward	-6
Episode Steps	6
Episode Q0	-5.8447
Total Number of Steps	1306

Average Results

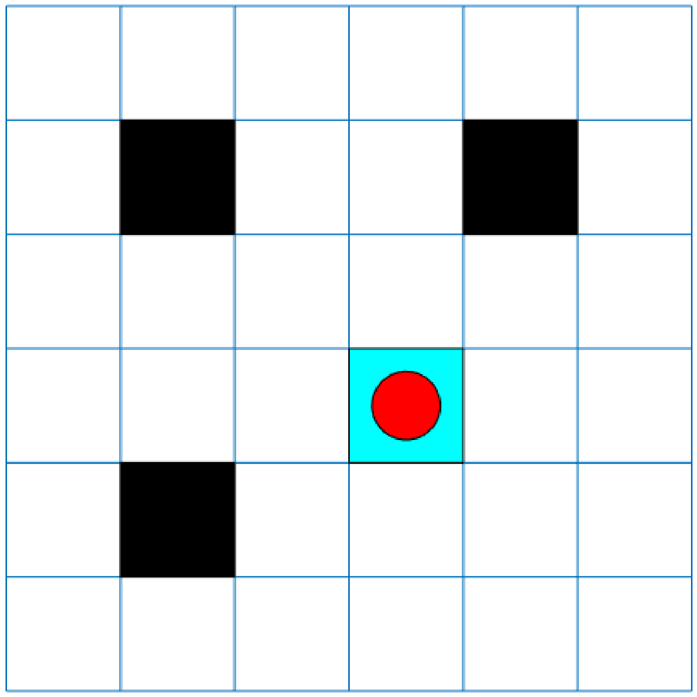
Average Reward	-6.2
Average Steps	6.2
Window Length for Averaging	30

Training Options

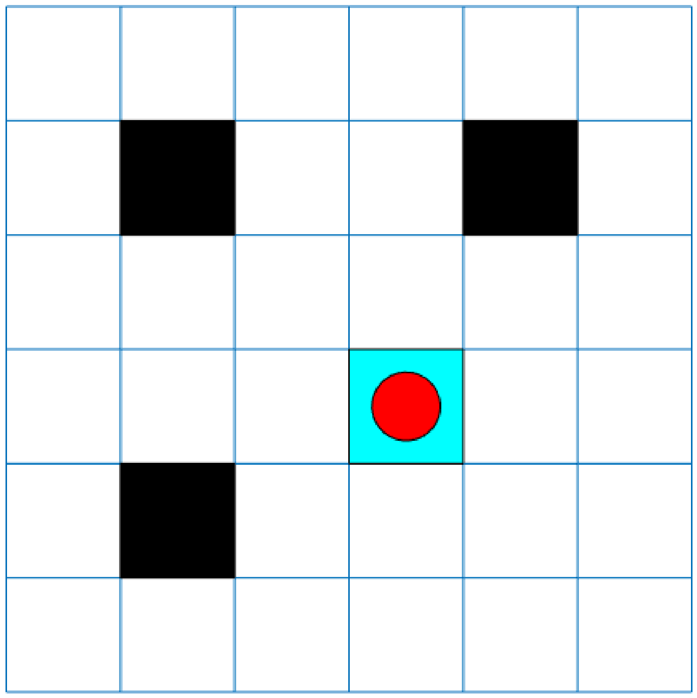
Hardware Resource	cpu
Learn Rate	1
Maximum Number of Episodes	150
Maximum Steps per Episode	50

Final Results

Training Stopped by	Maximum number of episodes
Training Stopped at Value	150
Elapsed Time	80.698 sec



```
plot(env)
```



To validate the training results, simulate the agent in the training environment.

Before running the simulation, visualize the environment and configure the visualization to maintain a trace of the agent states.

```
env.Model.Viewer.ShowTrace = true;
env.Model.Viewer.clearTrace;
```

```
sim(qAgent,env);
```