Reinforcement Learning

Homework 2

Ogechi Onuoha (s1459203)

1 Question 2

The problem was solved using the representation:
$Q(s, a) = \sum_s \phi_s^T \theta_s(a) \quad (i)$

Since the problem has 25 states and 4 actions in each state, we would need 25* 4 indicator functions to represent the lookup table.

Hence, in this problem, ϕ_s is a vector with the following properties:

$\phi_s(k) = 1 \ when \ k = s, \ and \ 0 \ otherwise$

Therefore, only the portion of (i) is returned when the parameters are set.

e.g for $Q(7, a) = \theta_s(a)$

Therefore, $\theta_s(a)$ can be viewed as the weight assigned to each action a when in state s. The value of being in state s can then be viewed as:
$V(s) = argmax_a \ Q(s, a)$

In my implementation, $\phi_s(k)$ is an s-dimensional vector while $\theta_s(a)$ is an s x a dimensional matrix.

Results: The task was implemented using Q learning with a lookup table and then with the indicator functions.

The exploration factor is increased to allow the algorithm perform a random walk at the earlier runs (episodes). This exploration factor is reduced as the number of episodes increases. This enables the algorithm collecting data to be used later in the algorithm.

From the version which uses Look up table, we have the following results:

Spatial representation of policy:

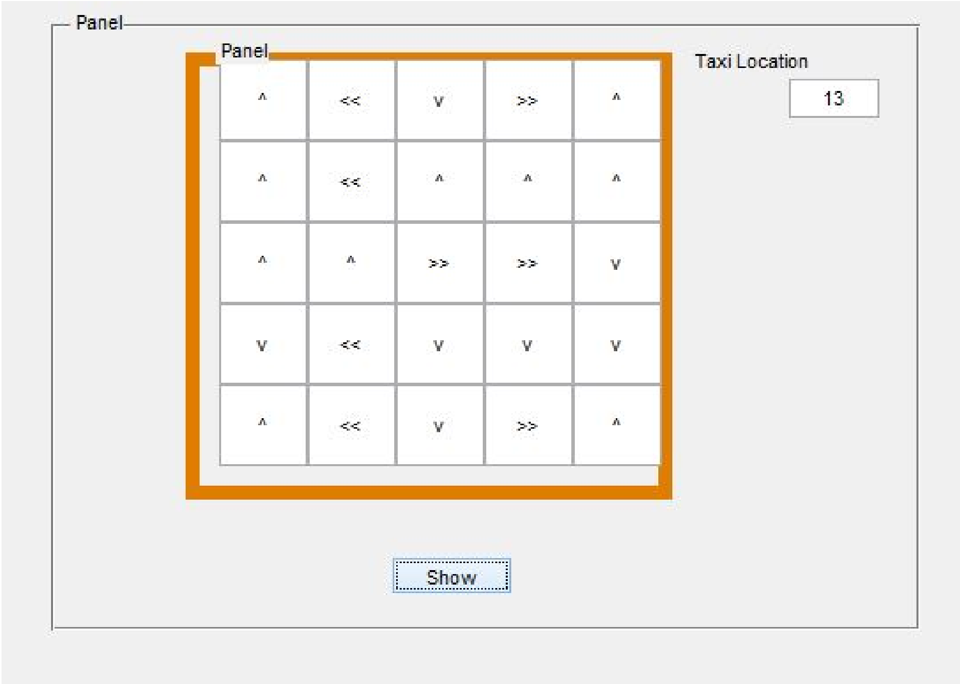
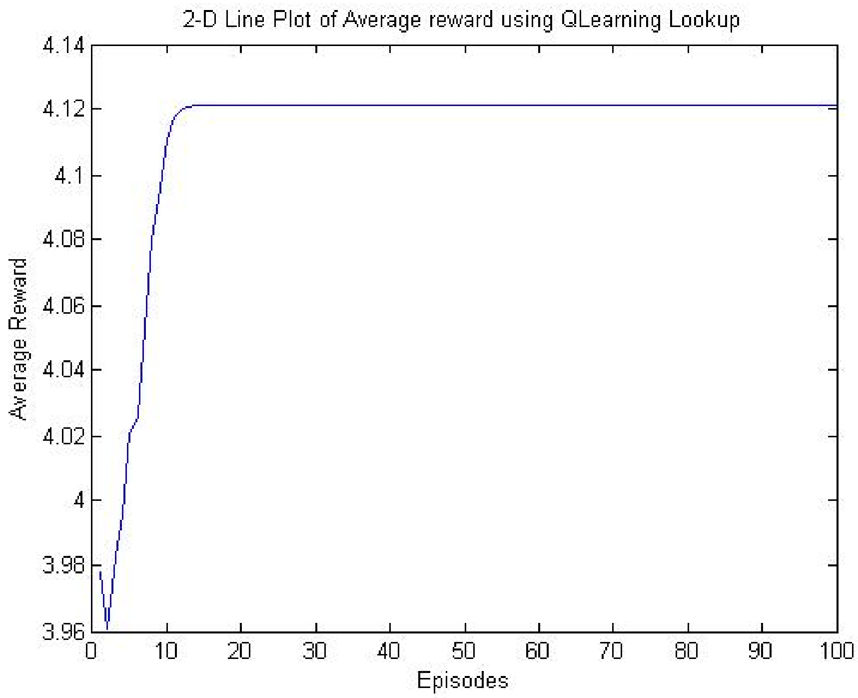


Figure 1: Spatial Plot of policy obtained from Qlearning using Table lookup



From the version that uses Indicator functions we have the following results:

Spatial representation of policy:

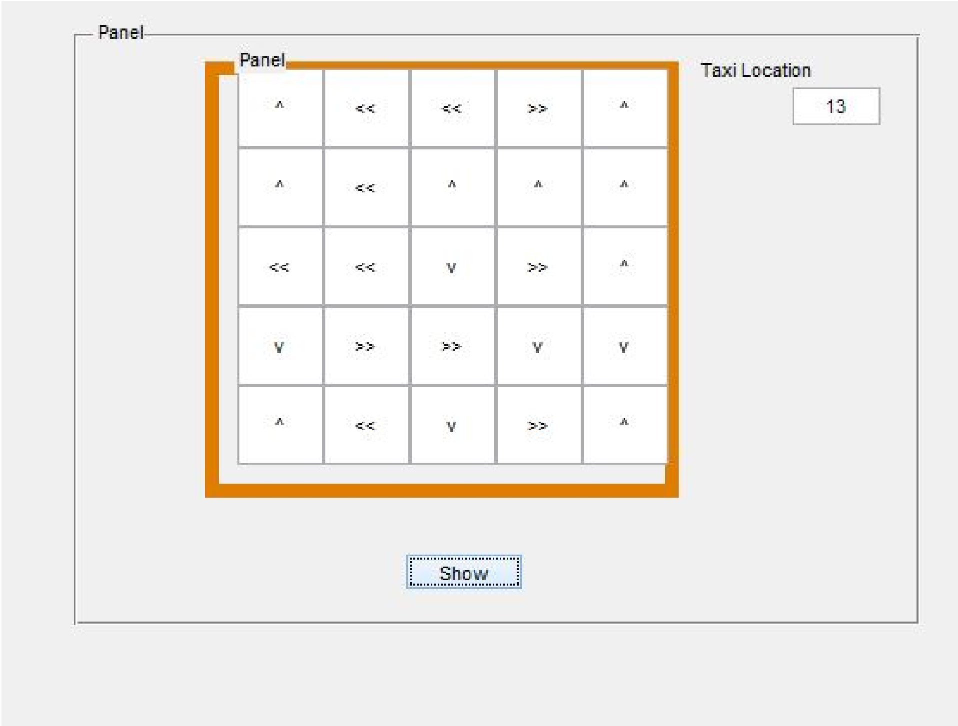
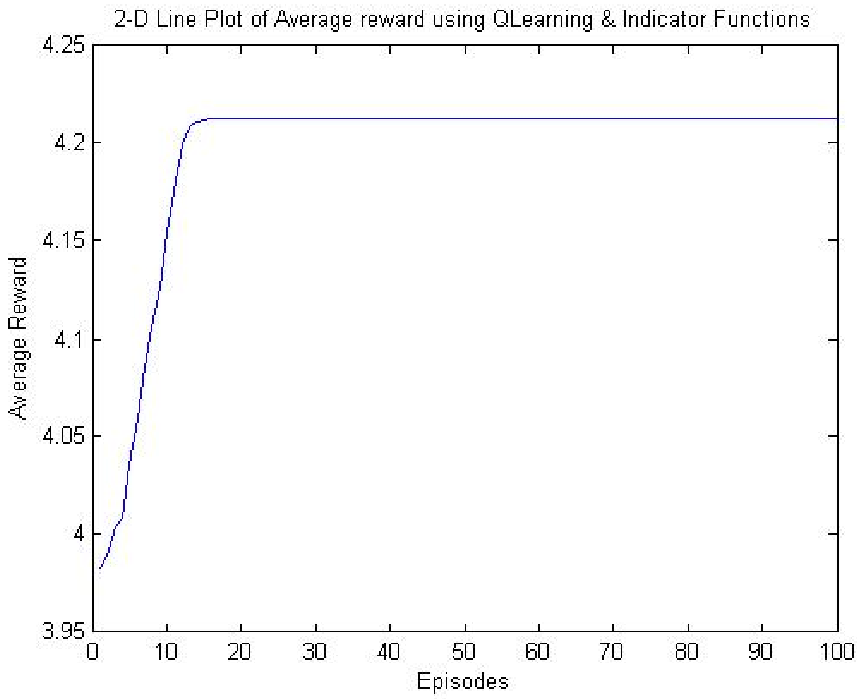


Figure 2: Spatial Plot of policy obtained from Qlearning using Indicator functions

Average cummulative reward:



Parameters used for simulation are as follows:

Parameter	value
Discount factor	0.95
Learning rate	0.7
Exploration Factor(ϵ)	0.8
Number of episodes	1000

The results show that in the long run using the table lookup method and the Indicator function method yield similar results. This is especially true for a problem with a small number of states*Actions. Hence, The table look up representation of the reinforcement learning problem is a special case of the function approximation method since indicator functions are used to perform a 'lookup' of the parameters instead of a search through the entire table.

2 Question 3

Using the same indicator functions, but now for a 10 by 10 grid, we find the following: The value function changes such that for each state in the 10 by 10 grid, we find the corresponding tile in the 5 by 5 grid that holds an approximation of its value. Hence, equation 1 becomes:

$Q(s,a) = \sum_k \phi_k^T \theta_k(a) \quad (ii)$

$where\ k = (k_x, k_y),$

$k_x = x - index\ of\ tile\ in\ 5x5\ grid = floor(s_x/m),$
$k_y = y - index\ of\ tile\ in\ 5x5\ grid = floor(s_y/m),$
$m = size\ of\ the\ area\ covered\ by\ each\ tile = 10/5 = 2$

or

$Q(s_x, s_y, a) = \sum_{sx,sy} \phi(s_x/m, s_y/m)^T \theta_k(a) \quad (ii)$

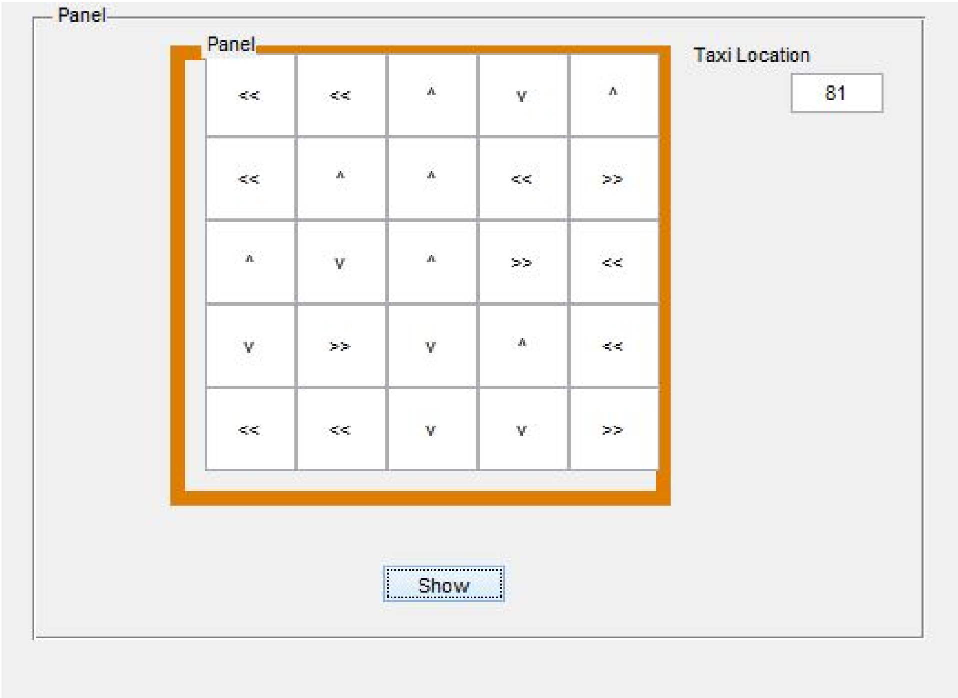
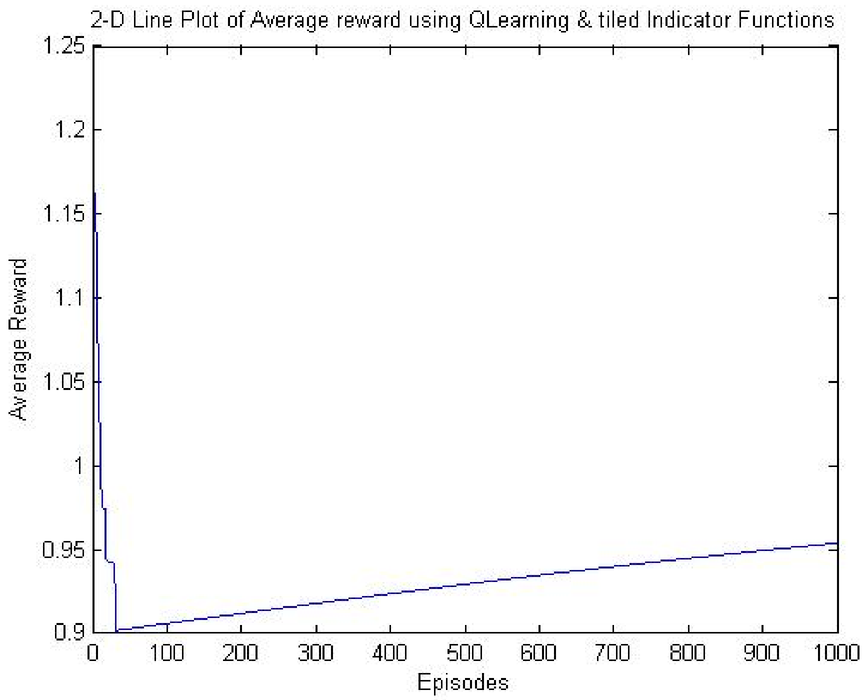


Figure 3: Spatial Plot of policy obtained from Qlearning using Indicator functions on a 10x10 grid



3 Question 4

For the 2D Gaussian radial basis functions the indicator function become:

$f_k(s) = f_(k_x, k_y)(s_x, s_y) = \exp(-(1/(2\sigma^2)) * (((s_x/m) - k_x)^2 + ((s_y/m) - k_y)^2))/(2\pi\sigma^2)$

The results are as shown below:

5x5 grid

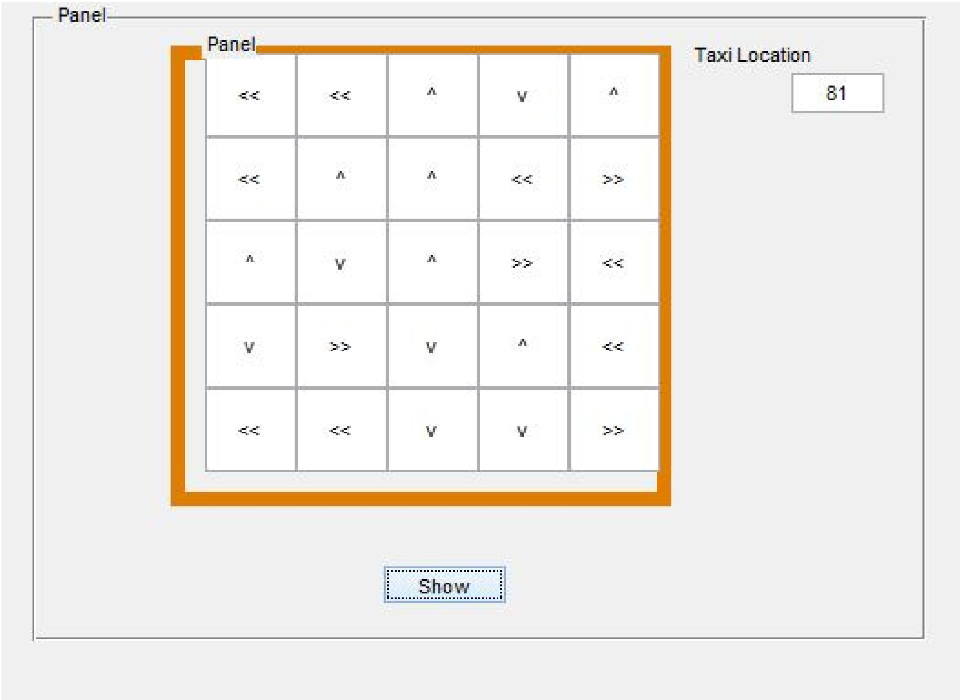


Figure 4: Spatial Plot of policy obtained from Qlearning using Radial Basis functions on a 10x10 grid

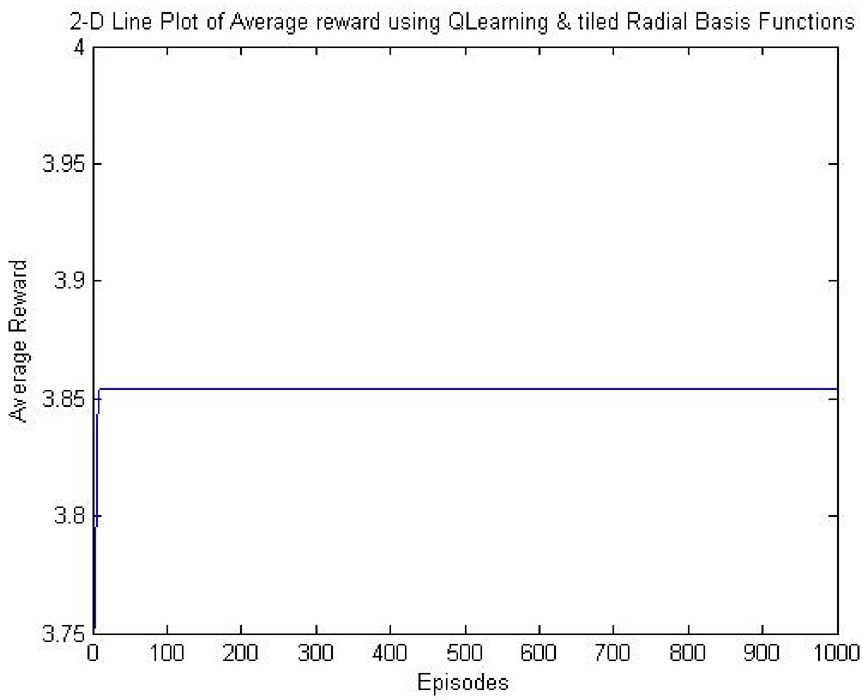


Figure 5: 5x5 grid

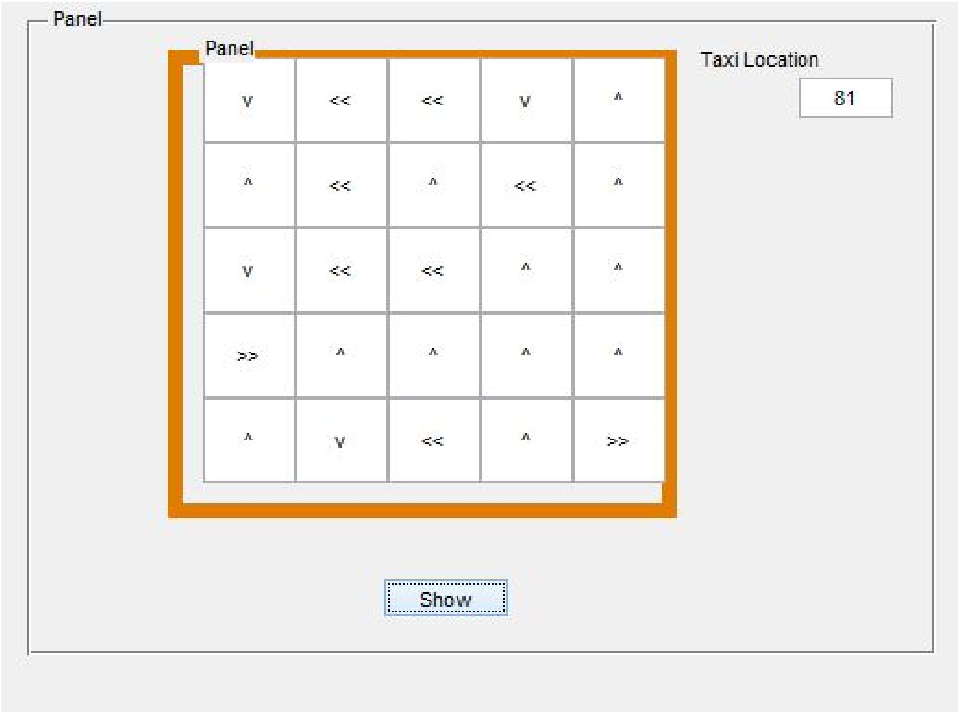


Figure 6: Spatial Plot of policy obtained from Qlearning using Radial Basis functions on a 10x10 grid

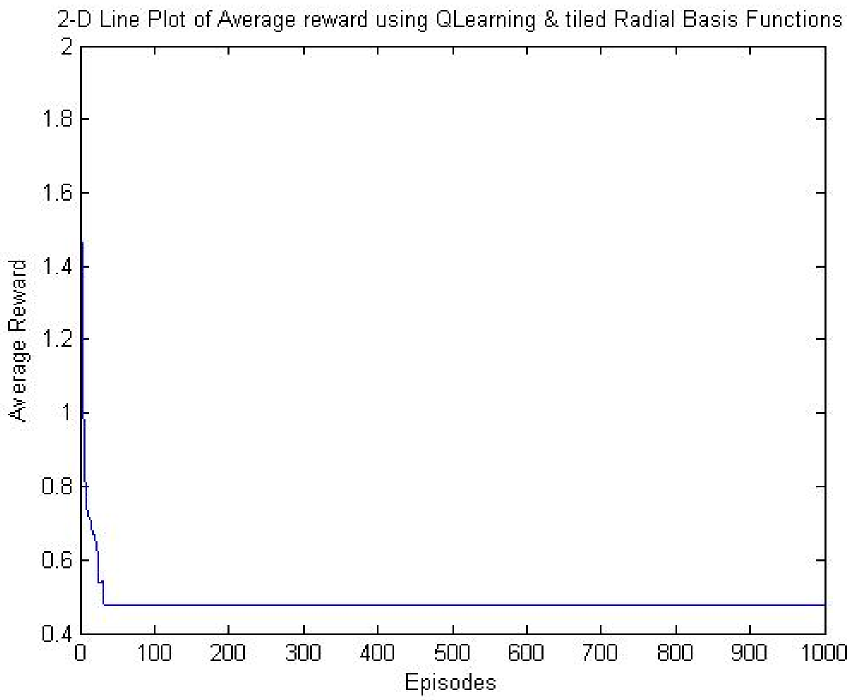


Figure 7: 10x10 grid

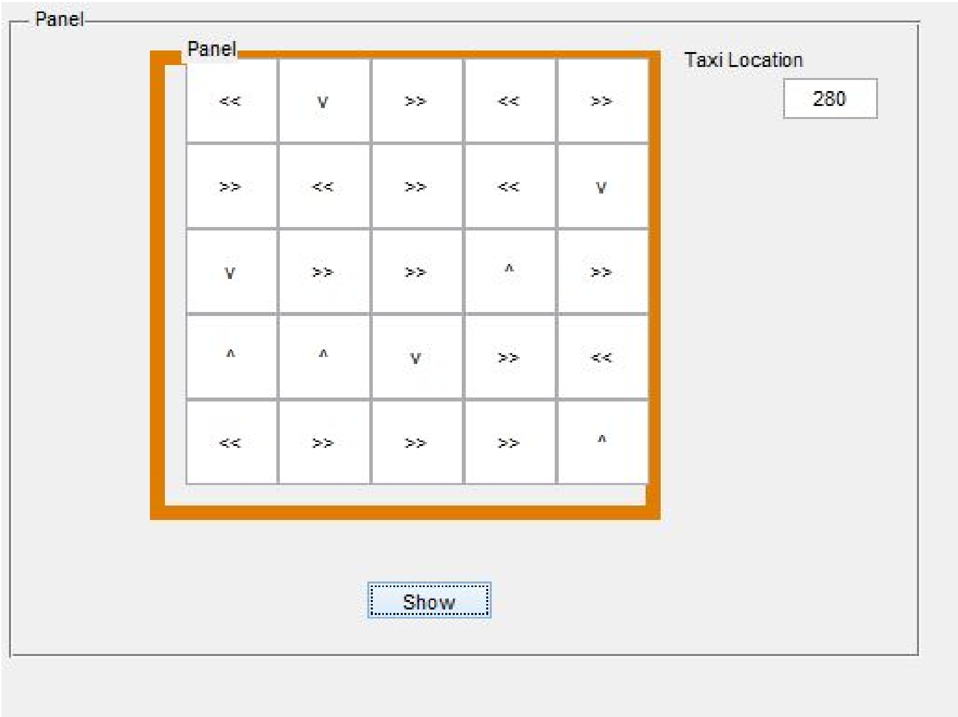


Figure 8: Spatial Plot of policy obtained from Qlearning using Radial Basis functions on a 10x10 grid

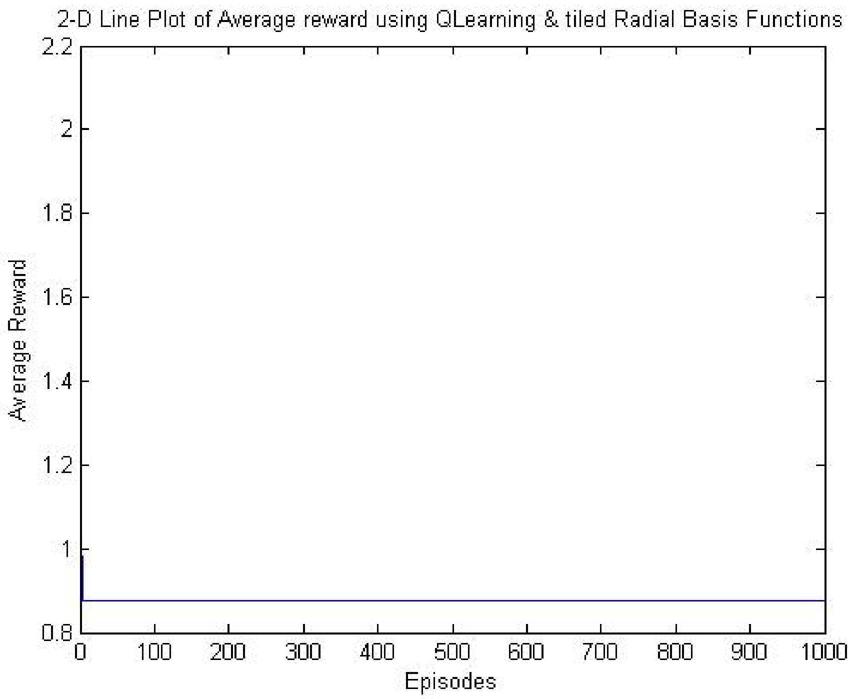


Figure 9: 25x25 grid

3.1 Effect of width on results

1.Width: The width of the bell shaped is an indication of the how the of the constituent functions are combined.

If the width is large, it results in a function that is over smoothed and is a poor representation of the underlying data or pattern. If the width is too small, the resulting function is over fitted to the data/examples it has seen. An optimum width is able to properly represent the underlying pattern in the data.

These results shown are obtained with the 10x10 grid

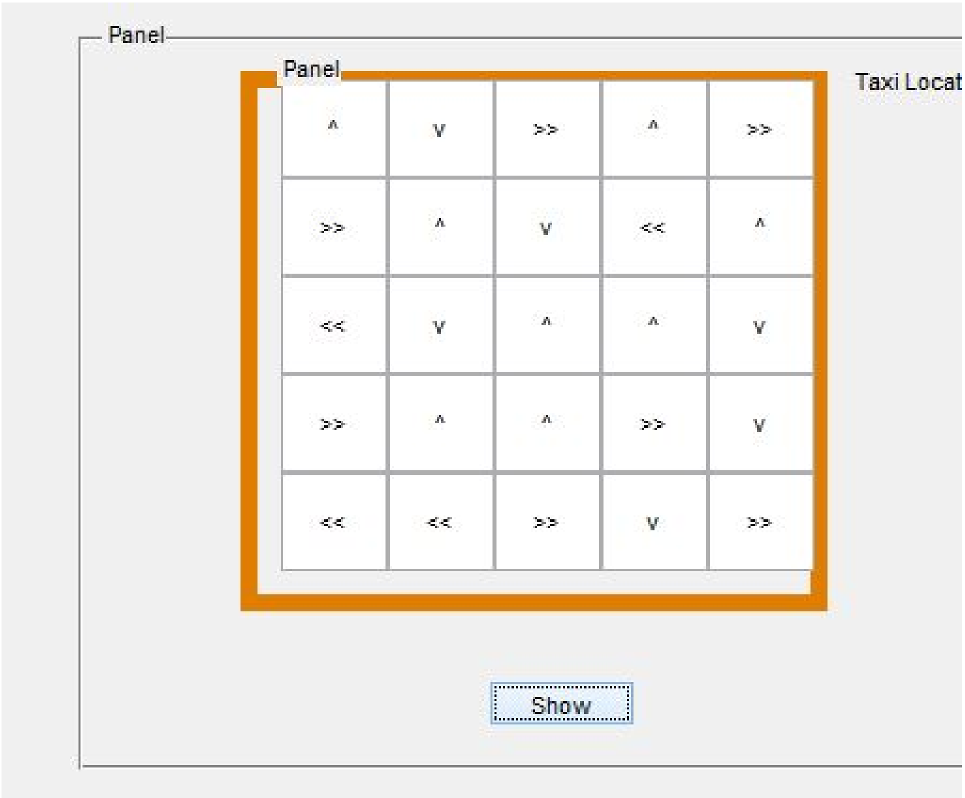


Figure 10: Spatial Plot of policy obtained from Qlearning using Indicator functions on a 10x10 grid when width =0.4

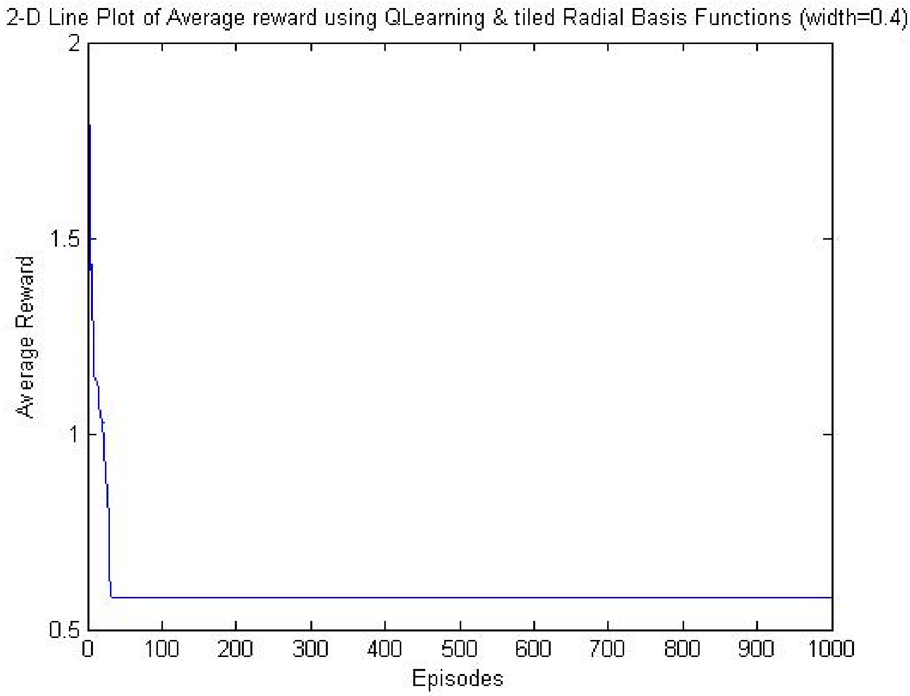


Figure 11: 10x10 grid

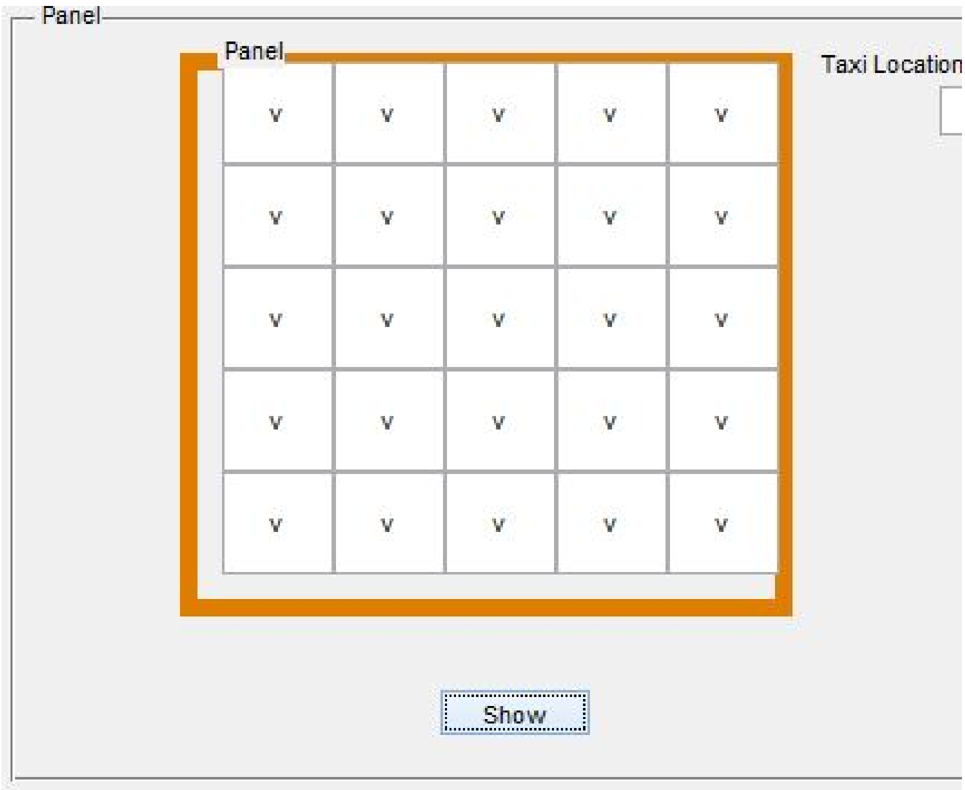


Figure 12: Spatial Plot of policy obtained from Qlearning using Indicator functions on a 10x10 grid when width =10

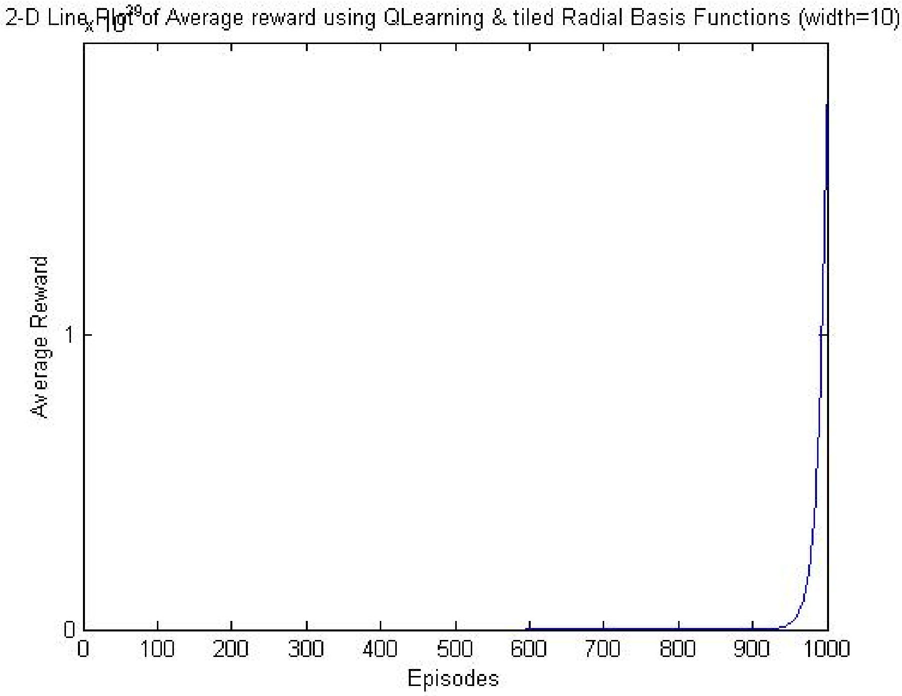


Figure 13: 10x10 grid

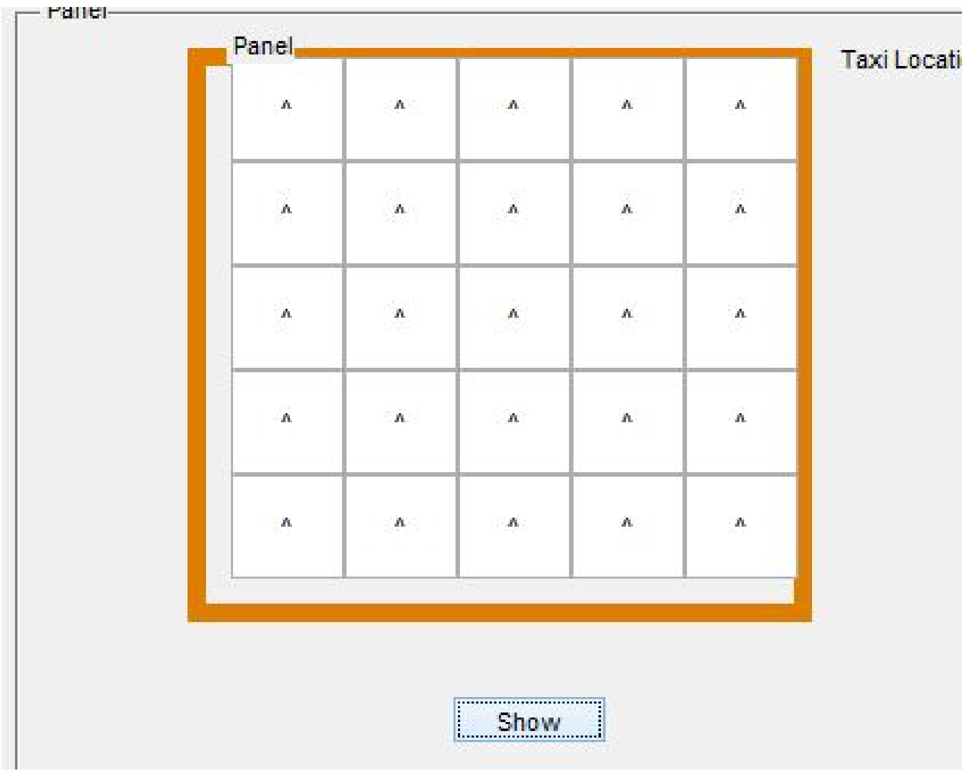


Figure 14: Spatial Plot of policy obtained from Qlearning using Indicator functions on a 10x10 grid when width =0.08

Table 1: Key to spatial plots

Symbol	value
ˆ	Move North(1)
v	Move South(2)
<<	Move East(3)
>>	Move West(4)
Pick	Pick Passenger(5)
Drop	Drop Passenger(6)

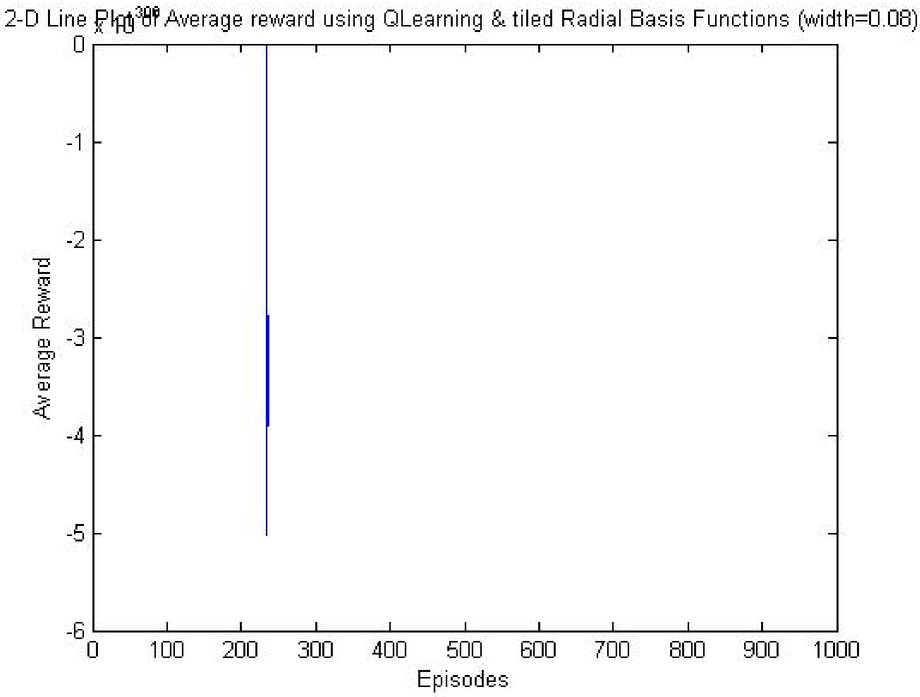


Figure 15: 10x10 grid

2. OTHER PARAMETERS Exploration factor: I noticed that increasing the exploration factor at the initial iterations of the task and reducing the value with time allowed for more exploration and 'data collection'.

Learning rate: I noticed that some values for the learning rate yielded poorer results than others. A very high or very low result may not be favourable.

Figure 16: Spatial Plot state space in 5x5 grid

4 Conclusions

Using function approximation methods greatly reduces the memory requirements of the learning agent. It also allows for generalisation of points close to known points. Using Radial basis functions further allows for measured contributions from neighbouring data points.

Appendix

Listing 1: Code to implement the solution to the problem using QLearning Lookup

```
%Initialisation of a trail
%State variables: taxiLocation {1, ..., 25},


%%
clear;
clc;
initTaxi = randi([1, 25]); %Taxi is uniformly randomly in any of the 25 grid squares
goalLocations = [1,5,21,25];
rewards = 1*ones(25,4); %initialise average reward gotten when each action is taken
inite = 0.7; % probability of exploration
lr = 0.8; % learning rate
discountFactor = 0.95;
counts =  zeros(25,4); %count of actions at each state or gridlocation
episodes = 1000;
cummulativeReward = zeros(episodes,1);
totalReward = zeros(episodes,1);
wait = 10;
initTaxi


%%
for episode=1:1:episodes
    taxiLocation = initTaxi;
    goalReached = false;
    timeLimit = 35;
    e = inite * (.05^(episode-1));
    tr=0;
    while goalReached == false && timeLimit > 0
        reward = 0;
        oldLocation = taxiLocation;
        action = chooseArm(e, rewards(oldLocation,:)); %returns the index of the arm cho
        count = counts(oldLocation,action); %gets the number of times this arm has been
        Q = rewards(oldLocation,action); %get the current Q state-action value

        %make a move
        [taxiLocation,successfulMove] = attemptMove(oldLocation,action,5);

        %get reward
        goalReached = ~isempty(find(taxiLocation == goalLocations,1));

        if goalReached
            reward =1;
            %sprintf('Goal Reached, episode: %d',episode)
        end
        nextOptimum = max(rewards(taxiLocation,:));
        Qnew = Q + lr *(reward + ((discountFactor*nextOptimum)- Q));
        rewards(oldLocation,action) = Qnew;
        counts(oldLocation,action)= count + 1;
        tr=tr+reward;
        timeLimit = timeLimit - 1; %reduce time limit
    end
    cummreward = sum(rewards,2);
    cummulativeReward(episode,1) = mean(cummreward);
    totalReward(episode,1)=tr;

    %convergence
    %if episode > 1
```

```
    %    if cummulativeReward(episode,1)− cummulativeReward(episode−1,1) < 0.0001
    %        if wait == 0
    %            sprintf('Final episode: %d',episode)
    %            sprintf('Average cummulative reward: %d',cummulativeReward(episode,1))
    %          break;
    %        else
    %            wait = wait −1;
    %        end
    %    end
    %end
end

[maxrewards,policy] = max(rewards,[],2);

%%
figure
plot(cummulativeReward(1:100,1))

title('2-D Line Plot of Average reward using QLearning Lookup')
xlabel('Episodes')
ylabel('Average Reward')
```

Listing 2: Code to implement exploration-exploitation algorithm.

```
function action = chooseArm(e,rewards)
    % e is the probability for exploration
    randNum = rand(1,1); %generate random number
    if randNum <= e
        %explore
        len = size(rewards,1);
        action = randi([1,len]);
    else
        %exploit
        [themax action]= max(rewards);
        %actionoption = find(max(rewards) == rewards);
        %action = actionoption(randi(numel(actionoption)));
    end
end
```

Listing 3: Code to implement an attempt to move by the taxi

```
function [newLocation,successfulMove] = attemptMove(grdLocation,action,grdsize)
    successfulMove = false;
    %try to go left
    if action == 3 &&  mod(grdLocation,grdsize) == 1
        newLocation = grdLocation;

    %try to go down
    elseif action == 2 && grdLocation <= grdsize
        newLocation = grdLocation;

    %try to go right
    elseif action == 4 && mod(grdLocation,grdsize) == 0
        newLocation = grdLocation;

    %try to go up
    elseif action == 1 && grdLocation > (grdsize-1)*grdsize
        newLocation = grdLocation;
    else
        if action == 1
            newLocation = grdLocation + grdsize;
```

```
        elseif action == 2
            newLocation = grdLocation - grdsize;
        elseif action == 3
            newLocation = grdLocation - 1;
        else
            newLocation = grdLocation + 1;
        end
        successfulMove = true;
    end
end
```

Listing 4: Code to implement the solution to the problem using QLearning with Indicator functions

```
%Initialisation of a trail
%State variables: taxiLocation {1, ..., 25},

%%
clear;
clc;
initTaxi = 13;
goalLocations = [1,5,21,25];
weights = 1*ones(25,4); %weights
inite = 0.7; % probability of exploration
discountFactor = 0.95;
counts =  zeros(25,4); %count of actions at each state or gridlocation
episodes = 1000;
lr = 0.8; % learning rate
cummulativeReward = zeros(episodes,1);
totalReward = zeros(episodes,1);


%%
for episode=1:1:episodes
    taxiLocation = initTaxi;
    goalReached = false;
    timeLimit = 35;
    e = inite * (.05^(episode-1));
    tr=0;

    while goalReached == false && timeLimit > 0
        reward = 0;
        oldLocation = taxiLocation;
        action = chooseArm(e, weights(oldLocation,:)); %returns the index of the arm cho
        count = counts(oldLocation,action); %gets the number of times this arm has been

        Q = indicator(oldLocation)' * weights(:,action); %get the current Q state-action

        %make a move
        [taxiLocation,successfulMove] = attemptMove(oldLocation,action,5);

        %get reward
        goalReached = ~isempty(find(taxiLocation == goalLocations,1));

        if goalReached
            reward =1;
            %sprintf('Goal Reached, episode: %d',episode)
        end

        nextOptimum = max(weights(taxiLocation,:));
        difference = (reward + ((discountFactor*nextOptimum)- Q));
        Q = indicator(oldLocation)' * weights(:,action); %get the current Q state-action
```

```
        Qnew = weights(:,action) + (lr * difference* indicator(oldLocation));
        weights(:,action) = Qnew;
        counts(oldLocation,action)= count + 1;
        tr=tr+reward;
        timeLimit = timeLimit - 1; %reduce time limit
    end
    cummreward = sum(weights,2);
    cummulativeReward(episode,1) = mean(cummreward);
    totalReward(episode,1)=tr;
end

[maxrewards,policy] = max(weights,[],2);

%%
figure
plot(cummulativeReward(1:100,1))

title('2-D Line Plot of Average reward using QLearning & Indicator Functions')
xlabel('Episodes')
ylabel('Average Reward')
```

Listing 5: Code to implement the movement of a taxi when motion is allowed

```
%Initialisation of a trail
%State variables: taxiLocation {1, ..., 25},


%%
clear;
clc;
gridsize = 25;
initTaxi = 280;%randi([1, gridsize*gridsize]); %Taxi is uniformly randomly in any of the
goalLocations = [1,gridsize,(gridsize*(gridsize-1))+1,gridsize*gridsize];
weights = rand(25,4); %weights
sigma = sqrt(1/(2*pi));
inite = 0.7; % probability of exploration
discountFactor = 0.8;
counts =  zeros(25,4); %count of actions at each state or gridlocation
episodes = 1000;
lr = 0.5; % learning rate
cummulativeReward = zeros(episodes,1);
totalReward = zeros(episodes,1);

initTaxi


%%
for episode=0:1:episodes-1
    taxiLocation = initTaxi;
    goalReached = false;
    timeLimit = 700;
    e = inite * (.04^episode);
    tr=0;

    while goalReached == false && timeLimit > 0
        reward = 0;
        oldLocation = taxiLocation;
        fn = radialBasisFunction(oldLocation,gridsize,sigma);
        [maxim oldpos] = max(fn);
        action = chooseArm(e, weights(oldpos,:)); %returns the index of the arm chosen
        count = counts(oldpos,action); %gets the number of times this arm has been used
```

```
        Q = fn' * weights(:,action); %get the current Q state—action value

        %make a move
        [taxiLocation,successfulMove] = attemptMove(oldLocation,action,gridsize);

        %sprintf('From Position %d Move made (%d) Taxi position %d , episode: %d',oldLoc

        %get reward
        goalReached = ~isempty(find(taxiLocation == goalLocations,1));

        if goalReached
            reward =1;
            %sprintf('Goal Reached, episode: %d',episode)
        end

        newfn =radialBasisFunction(taxiLocation,gridsize,sigma);
        [maxim newpos] = max(newfn);
        nextOptimum = max(weights(newpos,:));
        difference = (reward + ((discountFactor*nextOptimum)- Q));
        %adjust the weight that had the highest contribution
        Qnew = weights(:,action) + (lr * difference* fn); %
        weights(:,action) = Qnew;
        counts(oldpos,action)= count + 1;
        tr=tr+reward;
        timeLimit = timeLimit - 1; %reduce time limit
    end
    cummreward = sum(weights,2);
    cummulativeReward(episode+1,1) = mean(cummreward);
    totalReward(episode+1,1)=tr;
end

[maxrewards,policy] = max(weights,[],2);

%%
figure
plot(cummulativeReward(1:episodes,1))

title('2-D Line Plot of Average reward using QLearning & tiled Radial Basis Functions')
xlabel('Episodes')
ylabel('Average Reward')
```