Tic tac toe

Table of Contents

Initialization

```
clear; clc;
```

Parameters

```
initialQValue = 1;
initial_epsilon = 1;
alpha = 1;
max_epochs = 1e5;
verbose = true;
state0 = zeros(3);
player1QTable = InitializeQTable;
player2QTable = InitializeQTable;

epsilon = initial_epsilon;
results = zeros(1,max_epochs);
```

Training

```
for epoch = 1:max_epochs
    if mod(epoch, 10000) == 0
        epsilon = epsilon/10;
    end
    if verbose
        if mod(epoch,max_epochs/50) == 0
            fprintf('Epoch %d: %d%% done.\n', epoch, round(epoch/
max_epochs*100))
        end
    end
    state = state0;
    visitedStateIndices = zeros(1,9);
    moveIndices = zeros(1,9);
    for t = 1:9
        numberOfTurns = t;
        currentStateIndex = stateToNumber(state);
        visitedStateIndices(t) = currentStateIndex;
        [qValues, player1QTable, player2QTable] =
 GetStateQValues(state, player1QTable, player2QTable, t,
 initialQValue);
        [state, move] = MakeMove(currentStateIndex, t, player1QTable,
 player2QTable, epsilon);
        moveIndices(t) = move;
        if t > 4
            player1reward = GetReward(state);
            if player1reward ~= 0
                break
            end
        end
    end
    results(epoch) = GetReward(state);
    [player1QTable, player2QTable] = updateQTables(state,
 player1QTable, player2QTable, moveIndices, visitedStateIndices,
 numberOfTurns, alpha);
end
```

Plotting

```
figure(1)
clf
hold on
windowSize = 1000;
windowStep = 20;
slidingWindow = [];
win = [];
draw = [];
lose = [];
for epoch = windowSize:windowStep:max_epochs
    results_window = results(epoch-windowSize+1:epoch);
    slidingWindow = [slidingWindow, epoch];
    win = [win, numel(find(results_window==1))/(windowSize)];
    draw = [draw, numel(find(results_window == 0))/(windowSize)];
    lose = [lose, numel(find(results_window == -1))/(windowSize)];
end
plot(slidingWindow, win, 'b')
plot(slidingWindow, draw, 'r')
plot(slidingWindow, lose, 'g')
xlabel('Number of iterations')
legend('Player 1 win', 'Draw', 'Player 2 win')
ylabel('Results fraction')
```

Save Q Tables

```
saveQTable(player1QTable, 'player1.csv');
saveQTable(player2QTable, 'player2.csv');
```

Functions

```
function [player1QTable, player2QTable] = updateQTables(state,
 player1QTable, player2QTable, moveIndices, visitedStateIndices,
 numberOfTurns, alpha)
    [player1reward, player2reward] = GetReward(state);

    % Player 1
    player1LastTurn = numberOfTurns-mod(numberOfTurns+1,2);
    t = player1LastTurn;
    move = moveIndices(t);
    visitedStateIndex = visitedStateIndices(t);
    player1QTable(2, visitedStateIndex).entry(move) = ...
        player1QTable(2, visitedStateIndex).entry(move) + alpha*(...
            player1reward - player1QTable(2,
 visitedStateIndex).entry(move));

    for t = player1LastTurn-2:-2:1
        visitedStateIndex = visitedStateIndices(t);
        nextVisitedStateIndex = visitedStateIndices(t+2);
        move = moveIndices(t);

        player1QTable(2, visitedStateIndex).entry(move) = ...
            player1QTable(2, visitedStateIndex).entry(move) +
 alpha*(...
                -player1QTable(2, visitedStateIndex).entry(move) + ...
                max(max(player1QTable(2,
 nextVisitedStateIndex).entry)));
    end

    % Player 2
    player2LastTurn = numberOfTurns-mod(numberOfTurns,2);
    t = player2LastTurn;
    move = moveIndices(t);
    visitedStateIndex = visitedStateIndices(t);
    player2QTable(2, visitedStateIndex).entry(move) = ...
        player2QTable(2, visitedStateIndex).entry(move) + alpha*(...
            player2reward - player2QTable(2,
 visitedStateIndex).entry(move));

    for t = player2LastTurn-2:-2:2
        visitedStateIndex = visitedStateIndices(t);
        nextVisitedStateIndex = visitedStateIndices(t+2);
        move = moveIndices(t);

        player2QTable(2, visitedStateIndex).entry(move) = ...
            player2QTable(2, visitedStateIndex).entry(move) +
 alpha*(...
                -player2QTable(2, visitedStateIndex).entry(move) + ...
                max(max(player2QTable(2,
 nextVisitedStateIndex).entry)));
    end
end
```

```
function [player1reward, player2reward] = GetReward(state)
    for i = 1:3
        if all(state(i, :) == 1) || all(state(:,i) == 1)
            player1reward = 1; player2reward = -1;
            return
        elseif all(state(i,:) == -1) || all(state(:,i) == -1)
            player1reward = -1; player2reward = 1;
            return
        end
    end
    i = [1,2,3];
    if all(diag(state) == 1) || all(diag(flip(state)) == 1)
        player1reward = 1; player2reward = -1;
    elseif all(diag(state) == -1) || all(diag(flip(state) == -1))
        player1reward = -1; player2reward = 1;
    else
        player1reward = 0; player2reward = 0;
    end
end


function [qValues, player1QTable, player2QTable] =
 GetStateQValues(state, player1QTable, player2QTable, t,
 initialQValue);
    if mod(t,2) == 1
        qTable = player1QTable;
    else
        qTable = player2QTable;
    end
    currentStateIndex = stateToNumber(state);
    if(isempty(qTable(2, currentStateIndex).entry))
        qTable(1, currentStateIndex).entry = state;
        qValues = ones(3)*initialQValue;
        qValues(state ~= 0) = NaN;
        qTable(2, currentStateIndex).entry = qValues;
    else
        qValues = qTable(2, currentStateIndex).entry;
    end
    if mod(t,2) == 1
        player1QTable = qTable;
    else
        player2QTable = qTable;
    end
end

function [newState, move] = MakeMove(currentStateIndex, t,
 player1QTable, player2QTable, epsilon)
    if mod(t,2) == 1
        qTable = player1QTable;
    else
        qTable = player2QTable;
    end
    state   = qTable(1, currentStateIndex).entry;
    qValues = qTable(2, currentStateIndex).entry;

    newState = state;
    playerMove = mod(t,2)*2-1;
    r = rand;

    if r < epsilon
        possibleMoves = find(~isnan(qValues));
    else
        possibleMoves = find(qValues == max(max(qValues)));
    end
    move = possibleMoves(randi(numel(possibleMoves)));
    newState(move) = playerMove;
end

function qTable = InitializeQTable
    qTable(2, 19683) = struct('entry',[]);
end

function number = stateToNumber(state)
    state = state+1;
    number = 1;
    for i = 1:9
        number = number+state(10-i)*3^(i-1);
    end
end

function testStateToNumber()
    for i = 0:19682
        stringArray = double(string(num2cell(dec2base(i,3))));
        state = reshape([zeros(1, 9-length(stringArray)),
 stringArray],3,3)-1;
        assert(stateToNumber(state) == i+1);
    end
end

function newQTable = saveQTable(qTable, filename)
    newQTable = cell(2, 19683);
    j = 1;
    for i = 1:19683
        state   = qTable(1, i).entry;
        qValues = qTable(2, i).entry;
        if ~isempty(state)
            newQTable(1,j) = {state};
            newQTable(2,j) = {qValues};
            j = j+1;
        end
    end
    newQTable(:, j:end) = [];
    newQTable2 = zeros(size(newQTable)*3);
    for j = 1:size(newQTable,2)
        for i = 1:2
            newQTable2(i*3-2:i*3,j*3-2:j*3) =
 cell2mat(newQTable(i,j));
```

```
        end
    end
    writematrix(newQTable2, filename);
end
```

Published with MATLAB® R2020b