# Homework 3: Tic tac toe

The goal of this exercise to write a computer program that trains two players to play tic-tac-toe by playing against each other.

Two players learn how to play tic tac toe with Q-learning. Train the two players simultaneously, playing against each other. The goal is to find a Q-table ensuring that the player never loses, see Section 11.5 in the Lecture notes. Use the \varepsilonε-greedy policy, start with a suitable value of $\varepsilon$ , and let $\varepsilon$ tend to zero as training progresses.

Each player has his/her own Q-table. For an efficient implementation, do not initialise your Q-tables for every possible state of the tic tac toe board. Instead only initialise Q-entries when they are enountered for the first time during training. To this end, write a function that checks whether a board configuration occurred previously, or not. If not, add it to your Q-table and initialise it.

You need to upload two csv files. One should be named "player1.csv" and describe the Q-table of the player going first. The other one should be called "player2.csv" for the player going second. The csv files should have the following structure: The first three lines should consist of all the board states known to your Q-table, horizontally appended. A board position is implented as a 3-by-3 block, where an "X" is represented by a "1", an "O" by a "-1" and an empty field by a "0". The next three lines mimic the structure of the boards in the first three lines, except here an entry represents the expected value for a move in that location. Move values for occupied locations should be represented by NaN entries.

The following image visualises the structure on the file "examplecsv.csv", which is also available to download. Note that the specific numerical values in bottom three lines are not meaningful, they are only meant to demonstrate the structure of the file.

Hints: To easily obtain the desired .csv structure in Matlab, maintain the Q-table as a cell array with 2 rows and as many columns as there are known board states. In the first row, save a board state in every cell as a 3x3 matrix and in the second row the corresponding expected future rewards also as a 3x3 matrix with NaN entries on occupied fields. You can then use the function cell2mat followed by csvwrite to generate a .csv file with the right structure. The matlab function isnan can be very useful when debugging errors with the NaN entries. Note that Matlab evaluates arithmetic operations where one component is NaN entirely as NaN.

```
clc
clear
global Q1
global Q2
Q1 = cell(0);
Q2 = cell(0);
alpha = 0.1;
gamma = 1;
epsi = 1;
p1_wins = 0;
p2_wins = 0;
draws = 0;
p1_wins_list = zeros([1,300]);
p2_wins_list= zeros([1,300]);
```

```
draws_list = zeros([1,300]);
% Q1{1,1} = zeros([3,3]);
% Q1{1,2} = eye(3);
% Q1{2,1} = rand(3);
% Q1{2,2} = rand(3);
% Q = cell2mat(Q1);
% csvwrite("prediction.csv", Q)
```

```
tic
for itr = 1:30000
    if ~mod(itr,100)
        epsi = epsi * 0.95;
        p1_wins_list(fix(itr/100)) = p1_wins;
        p2_wins_list(fix(itr/100)) = p2_wins;
        draws_list(fix(itr/100)) = draws;
        p1_wins = 0;
        p2_wins = 0;
        draws = 0;
    end

    if ~mod(itr,500)
        fprintf("iteration: %d", itr)
        toc
        tic
    end
    % borad init
    board = zeros(3);
    isend = 0;
    player = 1;

    ifseen(board, 1);
    [action_board, action_coord_1] = choose_action(board, epsi, player, 1);
    board_prime = board + action_board;
    player = ~player;

    ifseen(board_prime, 2);
    [action_board, action_coord_2] = choose_action(board_prime, epsi, player, 2);
    board_next = board_prime + action_board;

    ifseen(board_next, 1);
    update_Q(board, board_next, action_coord_1, 1, alpha, gamma, 0);
    player = ~player;
    board = board_next;
    while true
        %% P1 round
        board_prime_old = board_prime;
        [action_board, action_coord_1] = choose_action(board, epsi, player, 1);
        board_prime = board + action_board;
        isend = boardcheck(board_prime);
```

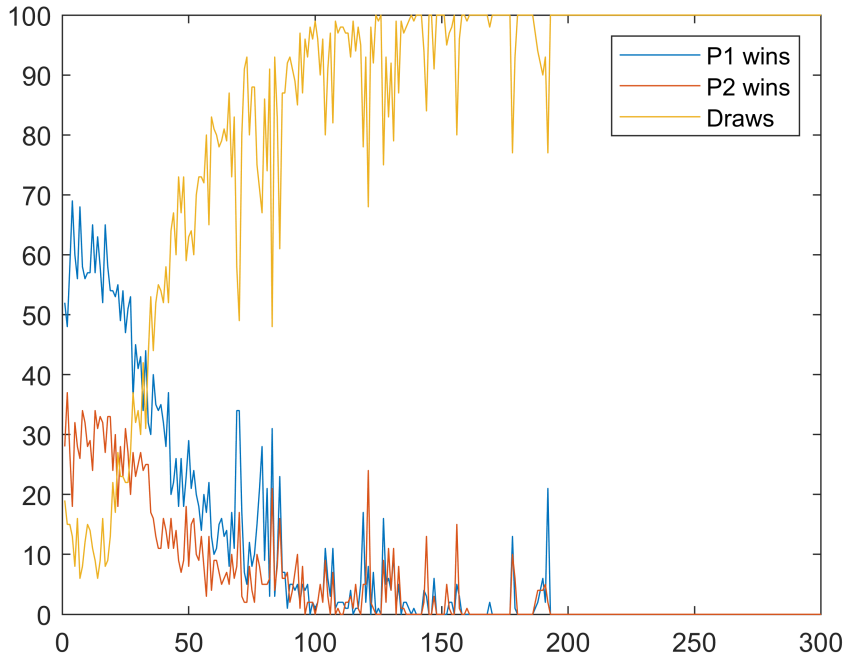
```
        if isend
            [R_1, R_2] = reward_assign(isend, player);
            fupdate_Q(board, action_coord_1, 1, alpha, gamma, R_1);
            fupdate_Q(board_prime_old, action_coord_2, 2, alpha, gamma, R_2);
            if R_1 == 1
                p1_wins = p1_wins + 1;
            elseif R_1 == 0
                draws = draws + 1;
            end
            break
        end
        ifseen(board_prime, 2);
        update_Q(board_prime_old, board_prime, action_coord_2, 2, alpha, gamma, 0);
        player = ~player;

        %% P2 round
        [action_board, action_coord_2] = choose_action(board_prime, epsi, player, 2);
        board_next = board_prime + action_board;
        isend = boardcheck(board_next);
        if isend
            [R_1, R_2] = reward_assign(isend, player);
            fupdate_Q(board, action_coord_1, 1, alpha, gamma, R_1);
            fupdate_Q(board_prime, action_coord_2, 2, alpha, gamma, R_2);
            if R_2 == 1
                p2_wins = p2_wins + 1;
            elseif R_2 == 0
                draws = draws + 1;
            end
            break
        end
        ifseen(board_next, 1);
        update_Q(board, board_next, action_coord_1, 1, alpha, gamma, 0);
        player = ~player;
        board = board_next;
    end
end
```

```
iteration: 500
Elapsed time is 11.224770 seconds.
iteration: 1000
Elapsed time is 25.981391 seconds.
iteration: 1500
Elapsed time is 39.872678 seconds.
iteration: 2000
Elapsed time is 40.380793 seconds.
iteration: 2500
Elapsed time is 46.417074 seconds.
iteration: 3000
Elapsed time is 46.199487 seconds.
iteration: 3500
Elapsed time is 50.297014 seconds.
iteration: 4000
Elapsed time is 52.316940 seconds.
iteration: 4500
```

```
Elapsed time is 53.992566 seconds.
iteration: 5000
Elapsed time is 60.719589 seconds.
iteration: 5500
Elapsed time is 58.984669 seconds.
iteration: 6000
Elapsed time is 62.605534 seconds.
iteration: 6500
Elapsed time is 65.278543 seconds.
iteration: 7000
Elapsed time is 63.629596 seconds.
iteration: 7500
Elapsed time is 62.940355 seconds.
iteration: 8000
Elapsed time is 62.590132 seconds.
iteration: 8500
Elapsed time is 63.361663 seconds.
iteration: 9000
Elapsed time is 64.290948 seconds.
iteration: 9500
Elapsed time is 66.681239 seconds.
iteration: 10000
Elapsed time is 67.556866 seconds.
iteration: 10500
Elapsed time is 66.123914 seconds.
iteration: 11000
Elapsed time is 67.082534 seconds.
iteration: 11500
Elapsed time is 67.421577 seconds.
iteration: 12000
Elapsed time is 66.309010 seconds.
iteration: 12500
Elapsed time is 67.233920 seconds.
iteration: 13000
Elapsed time is 68.561316 seconds.
iteration: 13500
Elapsed time is 70.956834 seconds.
iteration: 14000
Elapsed time is 69.166533 seconds.
iteration: 14500
Elapsed time is 68.381377 seconds.
iteration: 15000
Elapsed time is 71.326025 seconds.
iteration: 15500
Elapsed time is 69.734293 seconds.
iteration: 16000
Elapsed time is 68.366011 seconds.
iteration: 16500
Elapsed time is 68.321209 seconds.
iteration: 17000
Elapsed time is 69.107384 seconds.
iteration: 17500
Elapsed time is 71.944423 seconds.
iteration: 18000
Elapsed time is 68.306564 seconds.
iteration: 18500
Elapsed time is 68.587723 seconds.
iteration: 19000
Elapsed time is 68.126026 seconds.
iteration: 19500
Elapsed time is 72.146461 seconds.
iteration: 20000
Elapsed time is 75.785543 seconds.
iteration: 20500
```

```
Elapsed time is 72.446900 seconds.
iteration: 21000
Elapsed time is 69.288334 seconds.
iteration: 21500
Elapsed time is 69.577856 seconds.
iteration: 22000
Elapsed time is 69.136593 seconds.
iteration: 22500
Elapsed time is 68.998672 seconds.
iteration: 23000
Elapsed time is 69.255912 seconds.
iteration: 23500
Elapsed time is 69.417889 seconds.
iteration: 24000
Elapsed time is 68.828483 seconds.
iteration: 24500
Elapsed time is 68.683166 seconds.
iteration: 25000
Elapsed time is 69.131624 seconds.
iteration: 25500
Elapsed time is 74.038393 seconds.
iteration: 26000
Elapsed time is 1084.565008 seconds.
iteration: 26500
Elapsed time is 72.249380 seconds.
iteration: 27000
Elapsed time is 72.258912 seconds.
iteration: 27500
Elapsed time is 74.637776 seconds.
iteration: 28000
Elapsed time is 70.498782 seconds.
iteration: 28500
Elapsed time is 69.214607 seconds.
iteration: 29000
Elapsed time is 66.331137 seconds.
iteration: 29500
Elapsed time is 66.900456 seconds.
iteration: 30000
Elapsed time is 66.593470 seconds.
```

```
figure
plot(p1_wins_list)
hold on
plot(p2_wins_list)
plot(draws_list)
legend("P1 wins", "P2 wins", "Draws")
```




```
function Q = ifseen(board, Q)
    global Q1
    global Q2

    if Q == 1
        if isempty(Q1)
            Q1{1,1} = board;
            board(board~=0)=NaN;
            Q1{2,1} = board;
        else
            index = find(cellfun(@(x) isequal(x, board), Q1(1,:)), 1);
            if isempty(index)
                Q1{1,end+1} = board;
                board(board~=0)=NaN;
                Q1{2,end} = board;
            end
        end
    else
        if isempty(Q2)
            Q2{1,1} = board;
            board(board~=0)=NaN;
            Q2{2,1} = board;
        else
            index = find(cellfun(@(x) isequal(x, board), Q2(1,:)), 1);
            if isempty(index)
                Q2{1,end+1} = board;
                board(board~=0)=NaN;
                Q2{2,end} = board;
```

```
            end
        end
    end
end

%% Distribute reward to players
function [R_1, R_2] = reward_assign(isend, player)
    R_1 = 0;
    R_2 = 0;
    if isend == 1 && player == 1
        R_1 = 1;
        R_2 = -1;
    elseif isend == 1 && player == 0
        R_1 = -1;
        R_2 = 1;
    elseif isend == 0.5
        R_1 = 0;
        R_2 = 0;
    end
end

% choose A from S using epsilon-greedy
function [action_board, action_coord] = choose_action(board, epsi, which_player, Q)
    global Q1
    global Q2
    action_board = zeros(3);
    if binornd(1, epsi) == 1
%         disp("randomly choose")
        [row, col] = find(board==0);
        randchoice = randi(length(row));
        action_coord = [row(randchoice); col(randchoice)];
    else
        if Q==1
            index = find(cellfun(@(x) isequal(x, board), Q1(1,:)), 1);
            Q_values = Q1{2, index};
            max_qvalue = max(Q_values(:));
            [x,y] = find(Q_values == max_qvalue);
            if length(x) > 1
                randchoice = randi(length(x));
                action_coord = [x(randchoice); y(randchoice)];
            else
                action_coord = [x;y];
            end
        else
            index = find(cellfun(@(x) isequal(x, board), Q2(1,:)), 1);
            Q_values = Q2{2, index};
            max_qvalue = max(Q_values(:));
            [x,y] = find(Q_values == max_qvalue);
            if length(x) > 1
                randchoice = randi(length(x));
```

```
                action_coord = [x(randchoice); y(randchoice)];
            else
                action_coord = [x;y];
            end
        end
    end
    if which_player == 1
        action_board(action_coord(1),action_coord(2)) = 1;
    else
        action_board(action_coord(1),action_coord(2)) = -1;
    end
end

%% Q update when game continues
function update_Q(board_old, board_new, action_coord, Q, alpha, gamma, reward)
    global Q1
    global Q2
    if Q == 1
        index_old = find(cellfun(@(x) isequal(x, board_old), Q1(1,:)), 1);
        index_new = find(cellfun(@(x) isequal(x, board_new), Q1(1,:)), 1);
        max_Q = max(Q1{2, index_new}(:));
        Q1{2, index_old}(action_coord(1), action_coord(2)) = Q1{2, index_old}(action_coord(1),
                                                              alpha * (reward + gamma * max_Q
    else
        index_old = find(cellfun(@(x) isequal(x, board_old), Q2(1,:)), 1);
        index_new = find(cellfun(@(x) isequal(x, board_new), Q2(1,:)), 1);
        max_Q = max(Q2{2, index_new}(:));
        Q2{2, index_old}(action_coord(1), action_coord(2)) = Q2{2, index_old}(action_coord(1),
                                                              alpha * (reward + gamma * max_Q
    end
end

%% Q update when game over
function fupdate_Q(board_old, action_coord, Q, alpha, gamma, reward)
    global Q1
    global Q2
    if Q == 1
        index_old = find(cellfun(@(x) isequal(x, board_old), Q1(1,:)), 1);
        Q1{2, index_old}(action_coord(1), action_coord(2)) = Q1{2, index_old}(action_coord(1),
                                                                alpha * (reward - Q1{2, ind
    else
        index_old = find(cellfun(@(x) isequal(x, board_old), Q2(1,:)), 1);
        Q2{2, index_old}(action_coord(1), action_coord(2)) = Q2{2, index_old}(action_coord(1),
                                                                alpha * (reward - Q2{2, ind
    end
end

%% Check if game over
function isend = boardcheck(board)
    isend = 0;
```

```
    board_flip = fliplr(board);
    row_sum = abs(sum(board,1));
    col_sum = abs(sum(board,2));
    if abs(sum(diag(board)))==3 || abs(sum(diag(board_flip))) == 3 || any(col_sum(:)==3) || any
        isend = 1;
        return
    end
    if all(board,"all")
        isend = 0.5;
        return
    end
end
```