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 ε , and let ε 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]);
```

```
tic
for itr = 1:30000
    if ~mod(itr,1000)
        epsi = epsi * 0.95;
        if fix(itr/3000)<1
            epsi = 0.95;
        end
        p1_wins_list(fix(itr/100)) = p1_wins/100;
        p2_wins_list(fix(itr/100)) = p2_wins/100;
        draws_list(fix(itr/100)) = draws/100;
        p1_wins = 0;
        p2_wins = 0;
        draws = 0;
    end

    if ~mod(itr,500)
        fprintf("iteration: %d", itr)
        disp(size(Q1,2))
        disp(size(Q2,2))
        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
```

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

```
Q1_mat = cell2mat(Q1);
Q2_mat = cell2mat(Q2);
csvwrite("player1.csv", Q1_mat)
csvwrite("player2.csv", Q2_mat)
```

```
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}(:));
        Q_value = Q1{2, index_old}(action_coord(1), action_coord(2));
        Q1{2, index_old}(action_coord(1), action_coord(2)) = Q_value + alpha * (reward + gamma
    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}(:));
        Q_value = Q2{2, index_old}(action_coord(1), action_coord(2));
        Q2{2, index_old}(action_coord(1), action_coord(2)) = Q_value + alpha * (reward + gamma
    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);
        Q_value = Q1{2, index_old}(action_coord(1), action_coord(2));
        Q1{2, index_old}(action_coord(1), action_coord(2)) = Q_value + alpha * (reward - Q_valu
    else
        index_old = find(cellfun(@(x) isequal(x, board_old), Q2(1,:)), 1);
        Q_value = Q2{2, index_old}(action_coord(1), action_coord(2));
        Q2{2, index_old}(action_coord(1), action_coord(2)) = Q_value + alpha * (reward - Q_valu
    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
```