project 4: minmax searches for game agents

Given what we discussed in lectures 05 and 06, the following programming tasks are rather mechanical (i.e. implementations might be tedious but should be straight forward). The only tasks requiring more in-depth thought or creativity are task 4.4 and 4.5 and you are very much encouraged to tackle at least task 4.5.

task 4.1: minmax for *tic tac toe*

In lecture 05, we discussed the minmax algorithm and how it allows for decision making in turn based games. Implement a function that allows a *tic tac toe* game agent to make moves according to the minmax values of the possible successors of the current game state. Host a tournament between two agents where one plays at random and the other according to minmax decisions. Gather the win/loss/draw statistics and ponder what you observe.
Note: even though the size of the *tic tac toe* game tree pales in comparison to those of other games, it may take a while for a game to finish if one agent plays based on minmax decisions. Maybe it'll take longer than you are willing to wait; don't be alarmed.

task 4.2: minmax with α, β-pruning for *tic tac toe*

In lecture 06, we discussed α, β-pruning as a possible remedy for the unacceptably long runtime of complete minmax searches. Hence, implement a function for α, β-pruned minmax searches for *tic tac toe*. Host another tournament between a random player and a player using minmax with α, β-pruning.

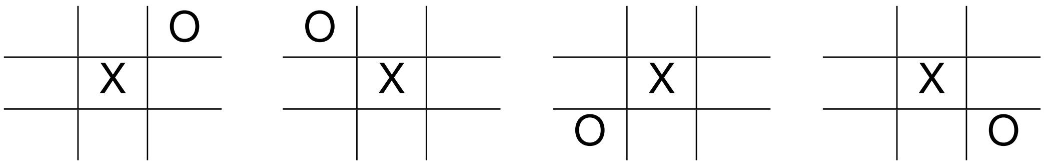
task 4.3: depth-restricted minmax for *tic tac toe*

In lecture 06, we also discussed the idea of depth-restricted searches which require the use of evaluation functions or heuristics for estimating the mmv of non-terminal states or internal nodes of the game tree. In the lecture, we had a brief look at a possible evaluation function for *tic tac toe*.

Use it to realize depth-restricted minmax searches. Have you implementation be flexible enough to allow for different depths or look-ahead. Use this depth-restricted search algorithm to compute the moves of one of the players while the other player still moves at random. Have both players play a tournament of many games and gather the win/loss/draw statistics. What happens if you modify the depth parameter of your depth-restricted search algorithm?

task 4.4: minmax with transposition tables for *tic tac toe*

In class, we also observed that many *tic tac toe* game states reoccur in several branches of the game tree. W could thus create look-up tables to store the mmv of states encountered during minmax searches so as to reuse those values once we reencounter a state in another branch of the tree. Hence (try to) adjust your code for depth-restricted minmax searches such that it makes use of look-up tables. Does this improve runtimes?
Note: there is a lot of symmetry on the tic tact toe board. For instance, the following game states are but rotated versions of each each other



All successors of these states will therefore be rotated versions of each other, too. Moreover, in addition to rotation symmetric states, there also are states that are reflections of each other. Can you think of a way of identifying symmetric states during minmax searches in order to further reduce runtimes?

task 4.5: depth-restricted minmax for *connect four*

Extend your code for *connect four* on a 6×7 board from task 1.3 towards more intelligent game play.
Think about an appropriate evaluation function that evaluates the merits of non-terminal nodes in a corresponding search tree and use this function to realize a depth-restricted search algorithm. Use this depth-restricted search algorithm to compute the moves of one of the players while the

other player still moves at random. Have both players play a tournament of many games and gather the win/loss/draw statistics. What happens if you modify the depth parameter of your depth-restricted search algorithm?
Note: later, we will up the ante and consider connect four on larger boards than 6×7. It may thus be a good idea to already think about evaluation functions that do not depend on the fact that the original version of *connect four* is played on a 6×7 board ...