Tic-tac-toe AI for (3,3,3) or (4,3,3)

Luque Martínez, Victor
Smart Electronics
ETSETB
Barcelona, Spain
Luque.viictor@gmail.com

***Abstract*—This document explains the game engine for a Tic-tac-toe AI working on (3,3,3) and (4,3,3) games, together with the algorithm implemented in this engine.**

***Keywords*—Tic-tac-toe, heuristics, min-max search, weighted min-max**

I. Introduction

This game engine has been developed from the previous engine developed by Sergio Bermejo (Tic-tac-toe-engine – 2). It has been completed to work as a normal (3,3,3) Tic-tac-toe and then modified to include a (4,3,3) board that the user can select before the start of the game, so it is possible to play both (3,3,3) and (4,3,3) game modes. This engine also comes with three different difficulty levels for the AI. The most basic level uses the function *my_random_move* where it selects the machine's move in a completely random way. The next level of AI uses the function *my_deterministic_move* where it selects the machine's move randomly except when there is a winning move or a blocking move. Finally, the most complex level of AI uses the function *my_move* where it selects the machine's move based on a weighted min-max algorithm.

II. Use of the program

A. *Selecting a Game Mode*

First thing to do after executing tic-tac-toe-engine.exe file will be to select a game mode. Two different game modes exist, where the user can choose to play in a 3x3 board or in a 4x3 board, being in both of them the same win rule (put three 'x' in a row). The user has to input '3' if he wants to play with the 3x3 board or '4' if he wants to play with the 4x3 board. Then he will be asked for the starting turn, where he can select the machine by introducing '1' or himself by introducing '2'.

B. *Difficulty of the opponent*

Three different kind of opponent difficulty can be used to adjust the playstyle of the AI. When we check the code, we need to modify the function move to make it use one of the three different functions for the AI move. If we select the function my_random_move then the AI will choose its move randomly thanks to the use of the function rand().

The second difficulty is achieved by using the function my_deterministic_move, which is a superior version of the previous one. This one still chooses its move randomly, but when he has a winning move it will prioritize that one, it will prioritize also a move to block the opponent's win (in this case the user).

The last difficulty is achieved by using the function my_move, in this one the AI created a copy of the board and explores all possible options of movement and assigns a value to each one depending if the final node is a win, a loss or a draw. It adds those values to each path and then choses the path with the greatest value, using a WMM algorithm.

III. AI Algorithm (WMM)

The algorithm used in this game engine was the Weighted Max-Min (WMM). This algorithm overcomes some of the limitations of the classical Max-Min. In this case, the algorithm doesn't need to backtrack all the tree after finding the node values, it simply adds on to the value for the same branch, so it is accumulating the difference of winning positions between the player and the machine, scaling those by the level of analysis of the algorithm. In the end, it chooses the branch with the highest value.

The level of the analysis has a great importance in this algorithm because it's not the same being able to win in the following turn, then having multiple options to win in five turns view, so it's scaled by a exponential value of level. Doing this we obtain an AI which focuses more on the short-term wins or loses, so it will always win if it has the chance, or block an opponent win if its possible, rather than making a move predicting some win in the future turns.

To achieve all this, our function *my_move* first copies the board to a new one, then for every one of the free available positions in the board, it makes a move and checks if there is a winner in the game, in case there is no winner yet, it calls the same function again until it reaches an end node with a winner (or a draw). When it reaches the end node, it adds to the value the new calculated one depending if it's a machine win (positive value) or a user win (negative value). If it's a draw nothing will be added.

We can also adjust the playstyle of the AI by changing the values to be added. If K_1 is the value added when the machine wins and K_2 is the one added when the player wins, then if $K_1>K_2$ the AI will be playing aggressive. Aggressive playstyle means that it will prioritize its win over the players win. If $K_1<K_2$ the AI will be playing defensive. Defensive playstyle means that it will prioritize blocking the player's win rather than its own win.