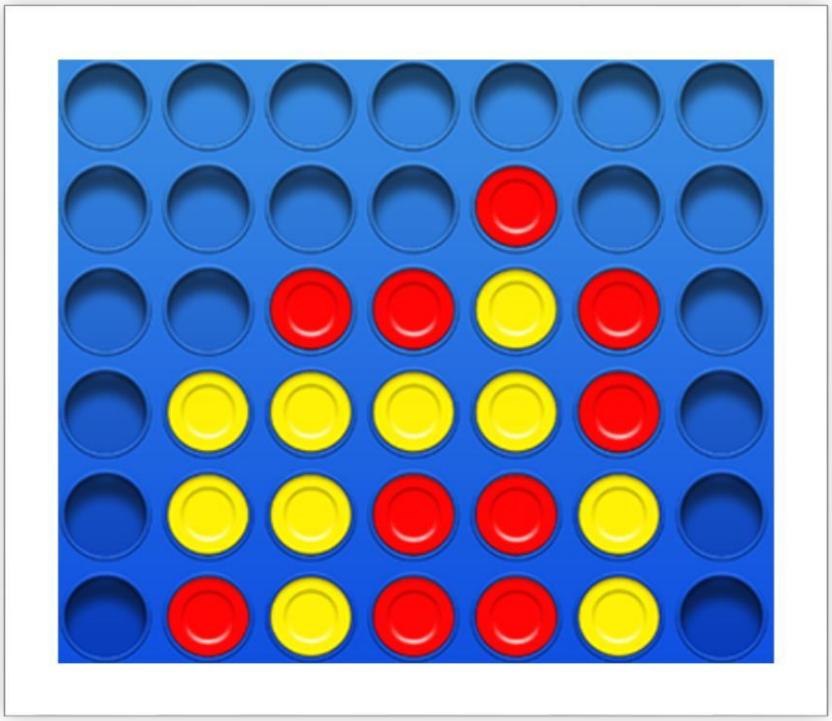

# 4-in a row

## FINAL CHALLENGE

Irving Salmerón Pérez | Artificial Intelligence | 24 de noviembre de 2015

# INTRODUCTION

Four in a row is a two player game that comprises strategy and decision taking. The game is composed of a 6x7 grid board and two types of tiles: red and blue normally, one for each player. To start each player choses one tile or color and a player is chosen to start, it may be randomly or by decision of the players. Turns are taken one per player putting one tile per turn on any of the 7 columns and the goal of the game is to play four tiles of the same color together, it can be either horizontal, vertical or diagonally. When there are 4 tiles of the same color in any of the mentioned positions the game is over and the player who owns the 4 tiles wins. A draw happens when no one has 4 tiles together and the board is full, this is 42 tiles have been played and there is no winner yet.

# JUSTIFICATION

Originally chess was chosen as the game to be implemented, but for time and complexity reasons it was changed for a less difficult game, which brought us to four-in a row which is a somehow chess look-like game as it involves decision making, game theory and state evaluation but with less move possibilities per turn and simpler piece movement options. However the game remains very interesting even it has been solved, computationally speaking, to foresight the whole game so the computer can win for sure as it could happen for example with tic-tac toe. There is research that proves that the first player to move can always win if correctly played but too many resources would be needed to foresight and correctly play to accomplish this.

Foresight is done with the minimax algorithm which works by building a decision tree built of states of the board. A board state is composed of a 6x7 grid containing the pieces that were played by then and depending on the chosen depth the tree is built.

In the case of our implementation the depth is defined by the difficulty level chosen at the beginning of the game. The depth determines how many moves ahead are evaluated so in the case of level one the tree is built one turn ahead, this is one move of each player. The root node of the tree is always the actual state of the board to evaluate in order to determine the best move the computer is going to perform, the leaf nodes are when the depth is reached, the board is full and/or a winning move is found, the last evaluation is called alpha-beta pruning where alpha is the maximum value a move can have and beta is the minimum value of the opponent player.

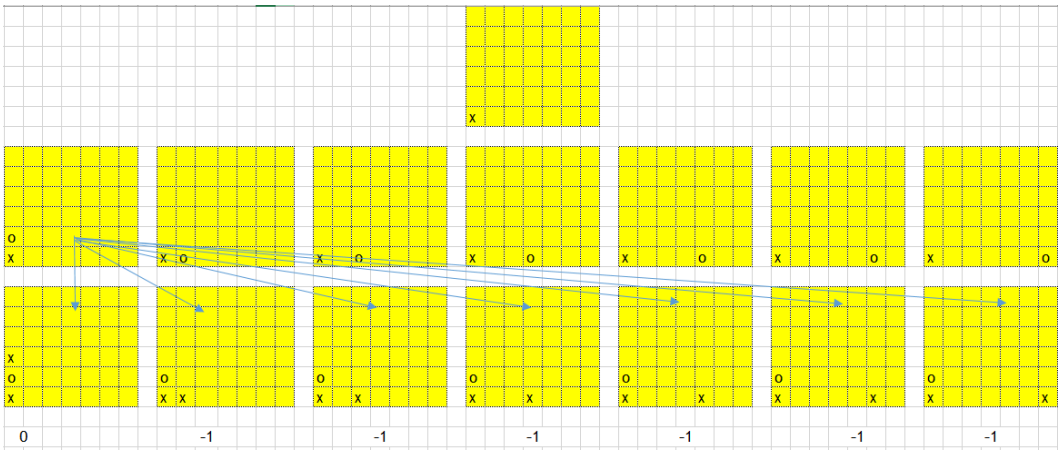


*Figure 1 Board states building with depth 1*

## HOW TO:

In order to run the application python 2.7 needs to be installed, the syntax to run the application is as follows:

- Python play.py

This brings the request to determine the difficulty of the computer from 1-4 and the game starts.

Human is always the X tile and computer is always the O tile, names for human player are assigned from a list of 2: Kasparov and Karpov which are chess players. For the computer is the same with: DeepBlue and Rybka.

## ANALYSIS

In this implementation randomly chosen moves are used, there are two other ways reported in [1] which work with different objectives: block or force a win.

As explained the minimax algorithm is used, which builds a decision tree, but in order to make this algorithm more efficient there is a couple of optimizations that could be done, like alpha-beta pruning which tries to find the maximum values for the computer (maximizer) and the opponent (minimizer) tries to get the lowest value for the maximizer, something like the rational thinking where one tries to not let the opponent win.

Here I have implemented something like an accustomed version of this optimization, where the algorithm prunes the branch if a maximum value is found otherwise it explores the whole branch, the pruning happens when a losing or winning board state is found or the analyzed move leads to a full board.

The implementation resulted in a game with 4 possible difficulties and the possibilities to play human-human, human-computer and computer-computer. I implemented this way for testing purposes as the human playing is very simple, just putting the tile in a valid space.

## PROBLEMS FOUND

During my implementation there were many issues as to understand how the minimax algorithm work, the evaluation of the states, the implementation, the tree construction and the function that stated the move score.

For the algorithm I found a bit hard to understand how to iterate if in turns or the two players turn but I decided to build it in two players turn so that every iteration starts in the same situation and circumstances. For the evaluation of the states I started with [1,0,-1] values for winning, losing or nothing happening but understanding the alpha-beta pruning algorithm and understanding some other work from [3] I realized I could evaluate the 4, 3 and 2-in a row and sum them every iteration per column. This brought me a new challenge as in depth > 1 the tree accumulates higher weights for other nodes rather than the winning ones since it iterates with the same position for every level of depth. First it seemed to be fixed rearranging the scores for every evaluation and winning states but when deep turns came, it still failed to win and decided to move in other columns to accumulate more points since if the winner move comes in depth 2 evaluation it scores with highest and returns accumulating higher values over iterations than the max value.

To solve this I checked the results of every iteration of the built tree and determined that the scores for non-winning boards where as high as the winning ones, so I increased enough the winning score and checked if the value is greater, then make it equal to the highest determined value.

## FUTURE RESEARCH AND CONCLUSION

There is still much to do with my algorithm as it is not completely optimized and there are several methods to try and test if they improve performance and results in thinking words. One way I could think is to improve the evaluation function, the heuristic, which is not as efficient as one would want. Also implement correctly the alpha beta pruning or improve it some way. The search of the possible states could be improved and optimized by linear programming and/or tabu search to select the possible states already visited and look for the best results or genetic algorithms as showed in [2] where the algorithm could find the best solution per state from a population and maybe offering a more efficient operation.

Another improvement is not to check for 3 and 2-in a row where no 4-in a row could be accomplish as it happens on the corners where the max length of the line is 2 or 3 but still my algorithm evaluates it.

In conclusion there is still much to do with this game problem that even it has been solved to determine the winner since the beginning it still offers educational and research work.

## REFERENCES

[1] AI for a Connect 4 game, Michele Shock, December 6, 2007, University of Michigan.

[2] A minimax control design for nonlinear systems based on genetic programming, Joe Imae, Nobuyuki ohtsuki, Yoshiteru Kikuki, Tomoaki Kobayashi,IEEE International Symposium, pp 2-4, Osaka Prefecture university, Japan, 2004.

[3] Teaching a computer to play connect four using the minimax algorithm, Johannes, 2014, taken from: https://johannes89.wordpress.com/2014/02/09/teaching-a-computer-to-play-connect-four-using-the-minimax-algorithm/