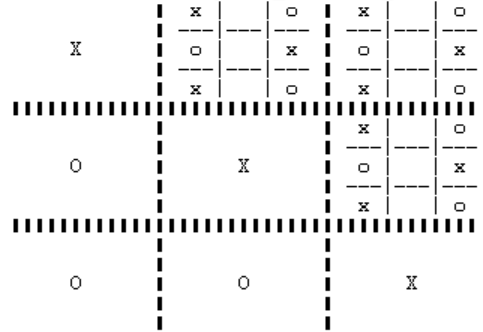
Traditional Project: Alpha-beta project from Bratko book (pag. 368)



Students:

Francesco Zangrillo (0001031021),

Simone Mele (0001037557)

Table of Contents

[**Introduction**](#_d16fxfke2vf3) **1**

[**1. Ultimate Tic Tac Toe**](#_30j0zll) **1**

[**2. Minimax with AlphaBeta cuts**](#_ml9covpr7jtq) **2**

[**3. Evaluation**](#_c6edjryul2ek) **4**

[**User manual**](#_k6edg8j6zrv7) **5**

# Introduction

In this project we implemented the Minimax algorithm with alpha-beta cuts for playing two-person, perfect information games. For this purpose, we implemented a variant of the Tic Tac Toe game: Ultimate Tic Tac Toe.

The project is structured in four decoupled modules:

1. **Game:** allows to choose the game (simple or ultimate tic tac toe), the player type (ai or human) and to manage game turns and interactions.
2. **Simple/Ultimate Tic Tac Toe:** define the rules of the game
3. **AlphaBeta:** node representation and algorithm implementation.
4. **Evaluation:** score configuration and heuristic implementation.

Moreover, predicates have been added for the graphical representation of the boards.

In order to achieve effective decoupling between the AlphaBeta algorithm and the two games, predicates with the same arity were designed.

# 1. Ultimate Tic Tac Toe

Ultimate tic-tac-toe is a board game composed of nine tic-tac-toe boards arranged in a 3 × 3 grid. Players take turns playing in the smaller tic-tac-toe boards until one of them wins in the larger tic-tac-toe board.

Immagine che contiene testo

Descrizione generata automaticamente

The game starts with X playing wherever they want in any of the 81 empty spots. If a move is played so as to win a local board according to normal tic-tac-toe rules, then the entire local board is marked as a victory for the player in the global board.

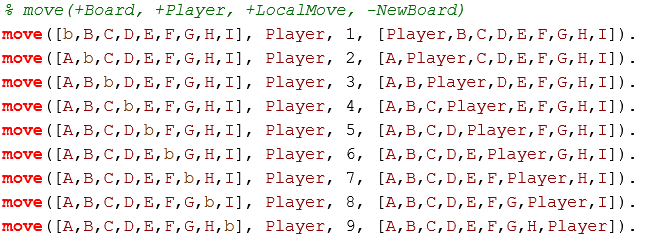
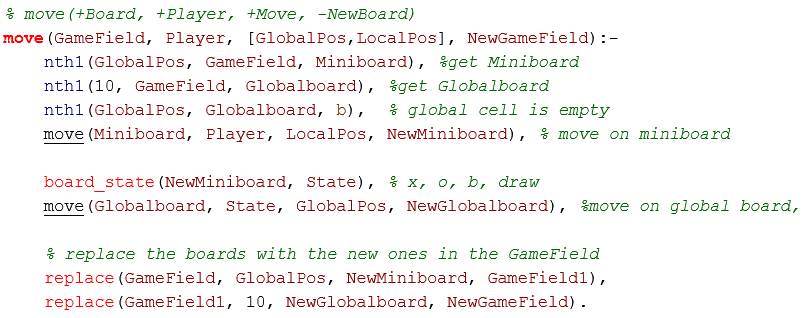
Each small 3 × 3 tic-tac-toe board is referred to as a miniboard, and the larger 3 × 3 board is referred to as the global board. Therefore, we can represent the game field as an array of ten boards:

*GameField = [B1, …, B9, GlobalBoard]*

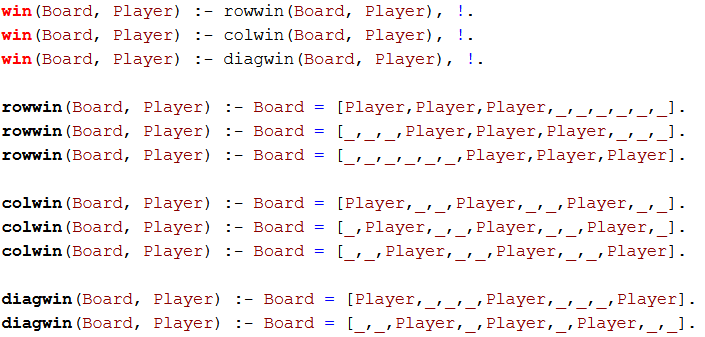
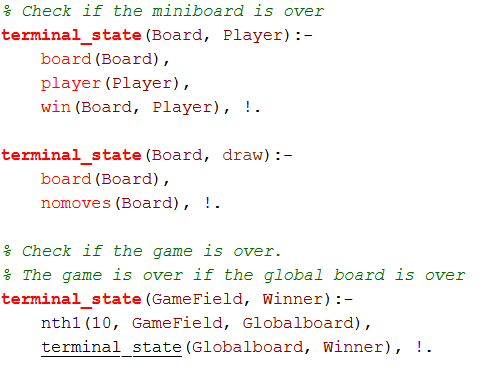
where each board is itself an array of nine cells:  *Board = [A1, …, A9]*.

A cell can have the following values: x, o, b (*blank*), draw. The last one is used only in the global board cells, to represent a draw in the corresponding miniboards.

A move on the game field is represented as a tuple: *[Global board cell, Miniboard cell]*.  
A move *y* on the miniboard can be performed if *Miniboard[y] = b*.  
A move *[x,y]* on the game field can be performed if *GlobalBoard[x] = b* and *y* is a valid move on the selected miniboard *#*x (i.e. *GameField[x]*).



Game play ends when either a player wins the global board or there are no legal moves remaining, in which case the game is a draw (see ***terminal\_state***).



# 

# 

# 

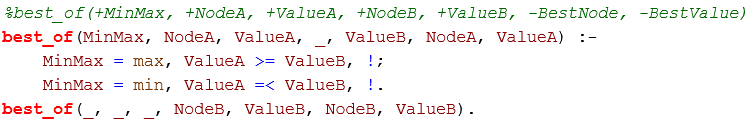
***The win predicate check if a single board is won. The winner of the game is the winner of the global board.***

# 2. Minimax with AlphaBeta cuts

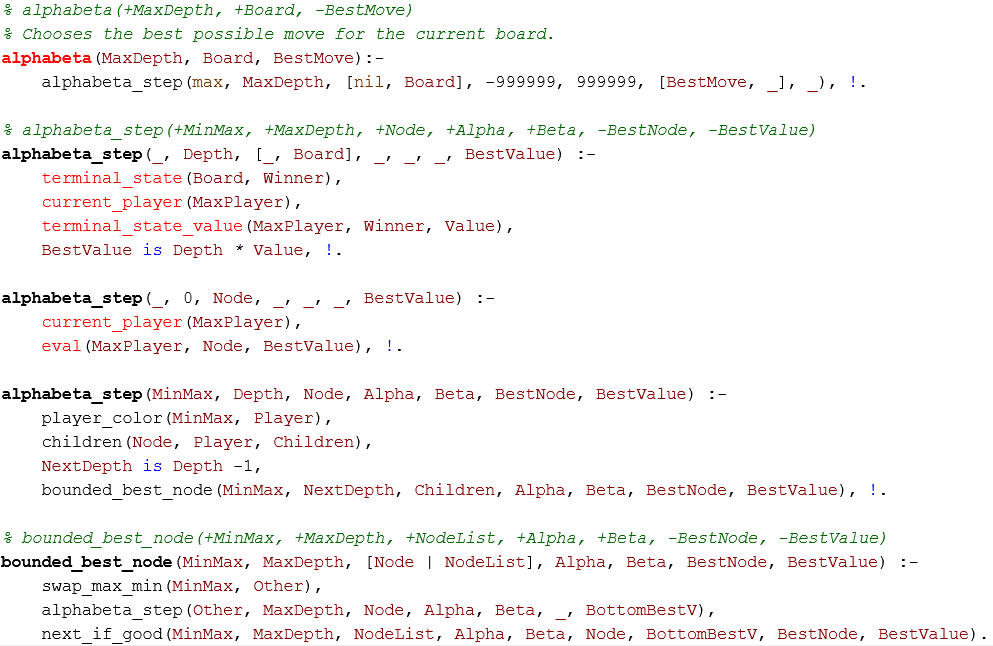
The Minimax principle is a standard technique on which computer game playing is based on.

A game tree is only searched up to a certain depth, typically a few moves, and then the tip nodes of the search tree are evaluated by some evaluation function. The estimated value then propagates up the search tree according to the minimax principle.

The higher the value the higher the player's chances are to win, and the lower the value the higher the opponent's chances are to win. The two players will be called respectively MAX and MIN: MAX will choose a move that maximizes the value; on the contrary, MIN will choose a move that minimizes the value (see **best\_of**)..



In order to reduce the search space, the algorithm holds two values, alpha and beta, representing respectively the minimum score that MAX ensures and the maximum score that MIN ensures.

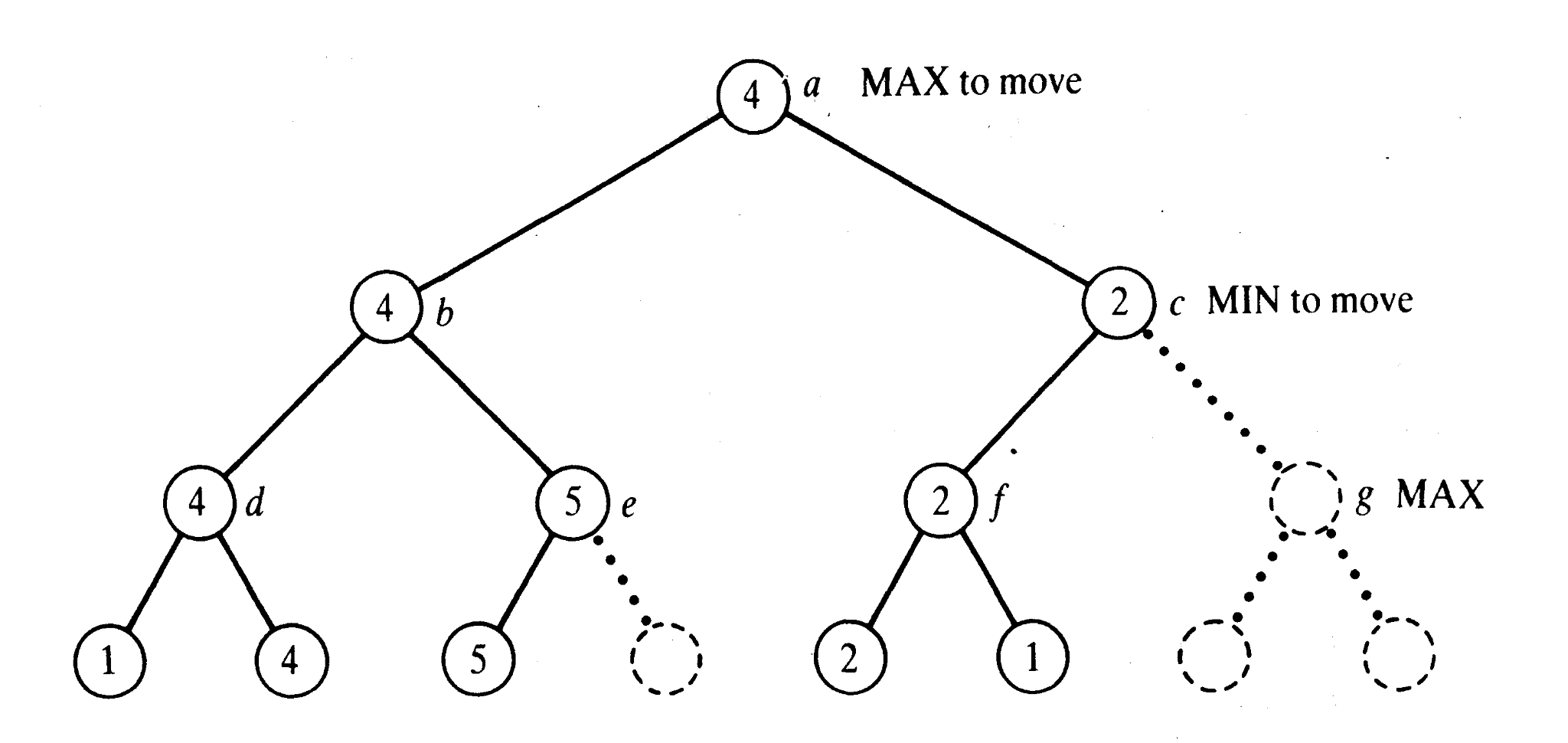


***The node is evaluated if the algorithm reaches either a terminal state or the max depth; otherwise the descendants are explored.  
The exploration is done by the bounded\_best\_node***

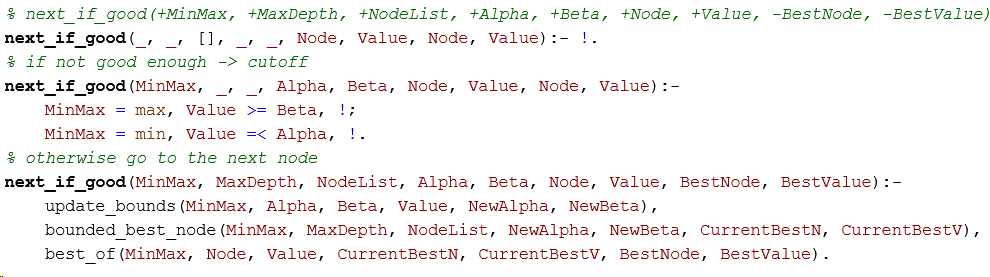
Whenever the temporary score of a MIN (MAX) node becomes lesser (greater) than the minimum (maximum) score of MAX (MIN), the MAX (MIN) player should not consider further descendants of this node, as they will never be reached in the actual play.

The cutoff is applied on descendants by the algorithm if the following statement holds:

or



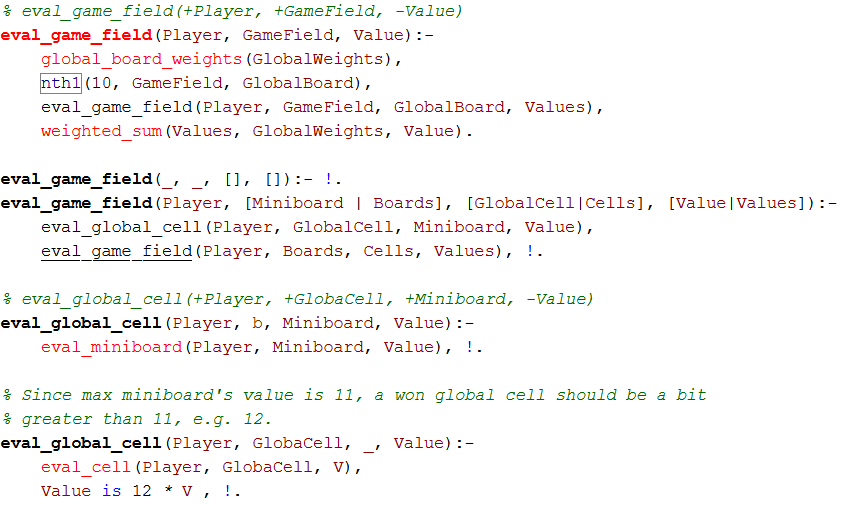
The ***next\_if\_good*** predicate has been implemented for this purpose. The exploration stops if the cut off conditions hold, otherwise it explores the next node and takes the best one (according to the minimax principle).

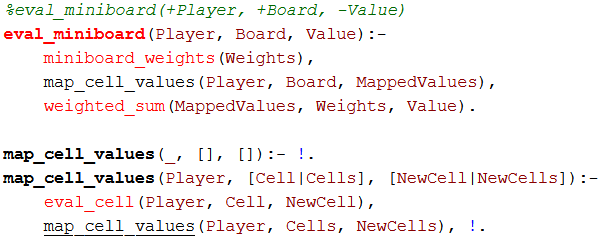


# 3. Evaluation

The aim of the evaluation function is to determine the quality of the state, in the sense of the winning probability. The outcome is known if the state represents a closed game, otherwise it should be estimated by an heuristic function.

The idea is to assign a score to each miniboard and calculate the game field score as a weighted sum (see ***eval\_game\_field***). The score of each miniboard is itself calculated as a weighted sum of the values assigned to each cell, which are 1 if the cell is occupied by the max , -1 otherwise (see ***eval\_miniboard***).  
Once a local board is won by a player or it is filled completely, a fixed score is assigned to the miniboard (see ***eval\_global\_cell***): it has to be large enough to prefer a local win to any other move within the miniboard, but small enough to make it possible to play on another miniboard.





The cells in the miniboards and the global board are weighted according to their winning probability. Therefore, the central one has a greater weight than the corners which, at the same time, have a greater weight than the remaining positions.

***miniboard\_weights****([ 3, 1, 3, 1, 5, 1, 3, 1, 3]).* ***global\_board\_weights****([ 3, 1, 3, 1, 5, 1, 3, 1, 3]).*

# 

# User manual

Start the game with the following command:

***start\_game(<Game>, <player\_x\_type>, <player\_o\_type>).***

***start\_game(tic\_tac\_toe|ultimate, ai|human, ai|human).***

e.g. ***start\_game(ultimate, ai, human).***