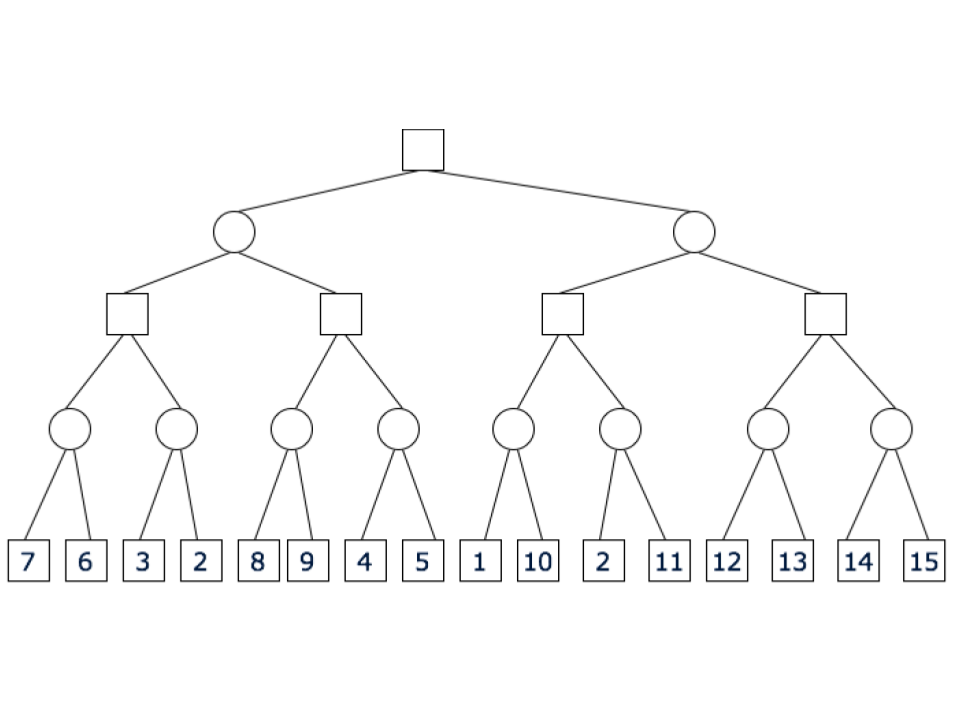
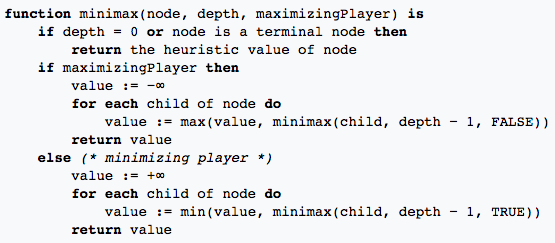
This report describes the implementation and intuition of negamax alpha beta with transposition table lookups. In order to get to the implementation of this algorithm, several intermediate steps were conducted (read: algorithms were implemented). This report first starts with the implementation of a minimax alpha beta pruning and improves to minimax alpha beta pruning with transposition table lookup. Next this reports progresses to negamax with alpha beta pruning and, lastly and finally concludes with negamax with alpha beta pruning and transposition table lookup. These steps are implemented to understand the value and strength of implementing negamax. Using the progression of these algorithms it can be seen where the strength of negamax with alpha beta pruning and transposition table lookup lies.

Minimax

Minimax is a depth first algorithm in which the starting player (max) always chooses the move with the highest possibility to win, i.e. aims to maximise its own score. The second player (min) aims to minimize the score of the max player and therefore always selects the move that provides the lowest score to the max player.





The general search flow (see~/ref{appendix} of each algorithm is conducted in a depth first manner. Meaning, that for each node (read: move) the algorithm generates a pair of moves based on the coordinates of the hex board. A pair consist of two coordinates; one coordinate for the white stone and one coordinate for the black stone. In total there are 3660 permutation pairs of moves, from which the algorithm randomly selects one of its children. Amongst those children the algorithm selects the child that yields the highest value. This child with the highest value is considered to be the best next move. All the implemented algorithms follow these first same steps. The difference lies in how the highest value is determined. The following sections elaborate more on this determination of the highest value.

## Minimax with alpha beta pruning

## In the first variation of minimax an alpha beta pruning is applied. This variation is a search algorithm that aims to reduce the size of a search tree by decreasing the number of nodes that are evaluated. Alpha beta pruning is generally used in adversarial search algorithms where there are two players. Each game has a max and a min player. The max player seeks to maximise his score whilst the min player aims to minimise the score. The search stops when there is at least one possibility found that proves a move to be worse than a previously examined move. When alpha beta pruning is applied to a minimax tree, the same move is return as a standard minimax algorithm would. The difference is that the alpha beta pruning returns this move with less nodes investigated. It can be seen in the pseudo code in figure~\ref{fig:pseudoAB} that, for each child the max or min value is selected depending on whether or not the next move is for the min or max player, respectively. In case of the max player, if the upper bound of an investigated child node appears to be smaller than the lower of a parent node, the remaining child nodes are not further investigated and a pruning occurs. In case of a min player, if the lower bound of an investigated child node appears to be higher than the upperbound of a parent node, the remaining child nodes are not further investigated and a pruning occurs.

## ../../../../Downloads/alphabetatree.png

## ../../../../Desktop/Screen%20Shot%202018-10-28%20at%2007.44.10.p

## Minimax with alpha beta pruning and transposition table lookup

## Negamax with Alpha beta pruning

In order for the reader to understand the gist of this report it is important to provide the basic understanding of the negamax algorithm. In essence, negamax is a variation on the minimax algorithm\*\*\*(add image). In negamax both players aim to maximise their own score without considering the impact on the opponent. In theory, it could be seen as there being no min players. Finally, when looking at alpha beta pruning the focus lies on the benefit of reducing the size of the Representation of the state space of a game (game tree) in order to optimise the search for the best moves.

## ../../../../Downloads/negamaxAB.png../../../../Downloads/negamax.png

## ../../../../Desktop/Screen%20Shot%202018-10-28%20at%2007.43.13.p

## Negamax with alpha beta pruning and transposition table lookup

## ../../../../Desktop/Screen%20Shot%202018-10-28%20at%2007.42.47.p

## How to run the program.

## In order to run the program there are three options.

## Run jupyter notebook Omega program, locally

## Go to terminal and navigate to the folder containing omega.ipynb

## When jupyter notebook is opened, click on omega.ipynb

## Click in the top menu on cell and press ‘Run all’.

## This runs the program and the game can be played.

## The user selects which AI to play against (advised is option 3: NegamaxAB TT, which is the fastest)

## The user then selects opponent type. The word human, decides if the user begins first (human vs AI) or second (AI vs human) or if the user wants to see a game simulation.

## The game ends when no moves are available and, thus, one of the players has won.

## Alternatively (if not having jupyter notebook installed), Run jupyter notebook Omega program, online.

## Go to:

## <https://notebooks.azure.com/redencio/libraries/omega2/html/omega.ipynb>

## (Sign in to run)

## Following step 2 and 3 as described above.

## Run python program.

## In a terminal, navigate to the location of the omega\_game.py

## Type python omega.py and click ENTER

## This runs the program and the game can be played.

## The user selects which AI to play against (advised is option 3: NegamaxAB TT, which is the fastest)

## The user then selects:

## Opponent type. Here, The word human, decides if the user (human) begins first (human vs AI) or second (AI vs human) or if the user wants to see a game simulation.

## The user also selects the search depth, which depends on how deep you want the engine to search for a best move. Generally, the higher the depth the longer it takes for the AI to find a best move. Higher than 4 does not necessarily improve the quality of the move. It does however, take more time to reach this good move and increases the chance of getting in a GHI situation. There is not general solution to this problem for depth first search, as it does not occur too often.

## The game ends when no moves are available and, thus, one of the players has won.

## For each run the game prints out the current state of the board game, starting with an empty board, the player to move, coordinates of which move this player has made and the board game (game state) after the move. For the AI, the time it took to find a best move is included and is called ‘thinking time’.