**3.5 AI Design And Implementation**

The aim of our program is to play interactively with the user through a solution of a given life and death problem. In order to do that, the program needs to be capable of solving the problem by itself. Accomplishing this required an artificial intelligence, which can make humanlike moves, to be created.

Typical life and death problem involves a board position and an objective to be achieved. The aim of the artificial intelligence is to find a sequence of moves in the given situation that satisfy this objective. This gives the program the ability to respond properly to any move made by the user (or by the program itself in AI vs AI game mode). Therefore, developing such artificial intelligence was of major importance. There are several ways it could be implemented by and one of them relies on using a tree search algorithm like Minimax or Alpha-Beta.

A tree search algorithm considers each possible action at a given moment alongside with each of the subsequent possible actions and so on until a state is found that satisfy a given goal. This allows for brute-forcing of life and death problems in attempt to find a winning sequence of moves. When the search reaches a state where the objective is completed or where there is no chance to complete it anymore the corresponding result is returned and the move with highest score is picked. Since all possible moves are considered, the larger the number of possible moves, the slower the response time of the AI. Therefore, to get a decent result in reasonable time the program should not deal with problems involving considering too many legal moves.

**Minimax:**

Definition: In games like Go where two players are involved the search tree algorithms become more complicated. They have to take into account the assumption that the opponent also chooses the move that gets the best result. That is the case with the Minimax search, where there are two types of nodes – a max and a min node. The max node is where the player tries to maximize his score and the min one is where the opponent attempts to minimize his. The leaf nodes are the terminal positions of each sequence and contain a value illustrating the result of the particular sequence. Starting from there and going up the tree the max nodes get the maximum value of its children and the min nodes – the minimum of its children nodes. At the end the root stores the value of the best move at that moment.

**Implementation:**

Our implementation of the Minimax follows the stated description, but the actual tree is not constructed. Instead, each move is created recursively. The type of the AI (either Minimax or Alpha-Beta) is chosen from the GUI by the user. If the Minimax is selected it is instantiated and gets a stone colour as well as the objective for the current problem.

On every move that the computer has to make it gets the current board and runs a recursive function for every legal move it finds with the help of the Legal move checker. The next step of the recursion is to consider every legal response that the opponent could perform and so on until a terminal state is found where either the computer wins, loses or just can not improve the situation by placing stones and has to pass. The values of the outcomes are then passed back and the results of all the initial moves are compared. The move with the highest result is chosen as a reply and its board coordinates are given to the game engine to actually perform the move.

A significant improvement was made during the implementation of the algorithm. Initially in the design on every recursive move generated a separate function had to create all the possible boards, which was found inefficient. It was replaced by a loop which iterates over the board and when a legal move is found the recursive function is called for that move.

**Alpha-Beta:**

Alpha-Beta search is almost similar to the Minimax. In fact they are guaranteed to always give the same result, but Alpha-Beta has better performance, since some of the branches are not investigated. The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially both players start with their worst possible score - alpha is negative infinity and beta is positive infinity. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta<=alpha). In that case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored. This property of the algorithm decreases significantly the time needed to find a good move when brute-forcing life and death problems.

**Implementation**

Our implementation of the Alphabeta does not differ much to the one of the Minimax. Again the actual tree is not constructed. The values of alpha and beta are maintained in order to break a particular recursion if the situation described in the previous paragraph occurs – beta becomes less or equal to alpha. The connection of the Alpha-Beta with the other components in the program is the same as for the Minimax.

**TODO:** add pictures of them both showing the same initial move to a problem and compare the number of moves that each algorithm considered to show Alpha-Beta is more efficient.

**TODO**: add conclusion