**1. Introduction**

This is a group project with the aim to create an interactive program that plays out and attempts to solve life and death problems within the board game, Go. The finished problem solver contains several elements. It features an artificial intelligence which selects future moves and a game engine which stores current game boards. Also a graphical user interface is a key component, allowing an individual to interact with the board, create problems and play against the game's artificial intelligence or another human. The finalised problem solver created allows a user to improve their ability and tactics to enable better play through of similar problems within a real-life game of Go.

When Go is analysed, it can be shown to be a very interesting game. It is a perfect information game that makes use of pure calculation during the making of moves and there is no psychology element during game play. This means that a computer simulation of the game is possible. Go in itself is also a very complex game, with its number of possible games being 10^761. In comparison, chess has 10^120 possible games, highlighting the vast search space of Go. Whilst a brute force chess program, which exhaustively searches for moves, can play at a championship level - the creation of a similar program in Go would take many years. A practiced human Go player may be able to look upon a game in progress and predict numerous future moves, allowing them to select moves that are beneficial to their current strategy. To create a program to achieve a similar level of skill such as this whilst playing Go requires a huge amount of time as well as a trade-off between fast running times and a more intelligent AI. This group project had 5 team members and 6 months to create a Go problem solver including an extensive artificial intelligence designed to be similar to a human player's thought process.

There are several phases within the game of Go and like other board games, many types of recurring problems which can be identified through specific patterns. This project's focus is on the life and death problems that occur during a game of Go and the creation a program which can successfully solve these problems whilst playing against a human. The game itself is played between two players using black and white stones respectively. These stones are placed on intersections within a board of grid lines and the main aim of the game is for a player's stones to enclose a larger area of the board than their opponent's. A player can also remove their opponent's stones from the board by surrounding them with their own stones, which is called capturing. Within the many rules of Go, such as capturing, life and death is a prominent concept. A group of a player's stones can either be classified as alive or dead. If this group of stones is mostly surrounded by opponent's stones but cannot be captured and removed from the board even when the opponent moves first, it is alive. For a group to be alive, it must contain at least two "eyes," being a section of board only surrounded by friendly stones. Whilst, if the group cannot avoid capture even when the player owning the group of stones moves first, it is classed as dead. The dead state normally occurs when a group has none or only one eye within it. Another state of stones that can occur is unsettled, where the outcome and eventual life or death of a group of stones depends on which player moves first. Normally, objectives accompany life and death problems - such as white to kill a group of black stones or black to escape capture from a group of white stones.

Life and death problems were selected as the project's focus for numerous reasons. One such reason is that the search space is smaller for these problems in comparison with a full game allowing the AI to be created and run more efficiently as less searching is required during move selection. Several methods can be used to provide move selection such as visual intuition or a randomized algorithm, e.g. Monte Carlo [*reference here*]. In this project the paired methods of board evaluation and move look ahead through Minimax [*reference here*] and AlphaBeta [*reference here*] were used, providing the most efficient basis of life and death problem solving. Life and death problems also allow the program to have a specific objective, as previously described, enabling specialised heuristics to be implemented which led directly on from common problem objectives. Following on from this, life and death problems also feature repeated common stone patterns which will normally influence the outcome of the problem. Hence, this allows heuristics to be implemented with these distinct stone positions and their usual outcomes kept in mind. To create a program that is able to mimic the game intelligence used during life and death problems, careful planning and several algorithms were used, including tree-searching techniques and numerous heuristics.

The first step in creating a Go program capable of playing against a human and solving life and death problems, is implementing a brute force strategy during move selection within a game. To enable this in the Go problem solver, a legal move checker utilizing a detailed algorithm must be created. This checker allows the program to filter out all illegal moves, minimizing the search space for the best possible move whilst the checker also removes any captured stones from the board. From there, minimax tree searching is utilized to decide on the game engine's next move through minimizing the possible loss for a worst case scenario - where scenarios are certain selections of future moves. The minimax algorithm plays out all possible legal moves and evaluates them through the user-given objective. If one of the first available moves successfully completes the objective, it is returned. Otherwise, minimax will play out the best moves for the AI and the opponent sequentially. This sequential play through creates branches within the search tree until no legal moves remain. Hence, through exhaustive search of all moves, minimax returns the next possible best move according to the objective supplied - allowing for successful brute force play through of life and death problems.

In order to make minimax more efficient, the alpha-beta algorithm can also be implemented in accordance with the minimax search tree. In essence, alpha-beta will return the same next best move for the artificial intelligence as the minimax but it works more effectively by removing branches from the original minimax search-tree that would not influence the final best possible move returned. The algorithm does this by stopping the evaluation of a branch of moves when it is found that these moves are worse than previously considered moves, which therefore creates a more optimal subtree - instead of the exhaustive search tree produced through the sole usage of the minimax algorithm, as described previously.

Whilst minimax successfully allows for brute forcing of problems and alpha-beta decreases the search space yet further, even using these two tree-search algorithms in conjunction with each other can still lead to long running times for more complicated problems. There is also no usage of strategy implemented through the utilization of minimax and alpha-beta for move selection. The implementation of heuristics and evaluation functions is fundamental to improving a program's artificial intelligence through adding algorithmic strategy. Heuristics are implemented within the Go problem solver to improve its move selection during play through, instead of solely taking a brute force approach. The use of heuristics is essentially akin to implementing the techniques and rules that humans use during play within the problem solver. By giving a program heuristics, there is an attempt to provide the judgement that humans possess by coding several game-related rules and regularly occurring board situations within it. Through the use of pattern matching and recognition algorithms, the system should be enabled to recognize when these heuristic situations occur and move accordingly - allowing for quicker and more human-like move selection.

The project's final program structure contains several key elements, including the graphical user interface and an artificial intelligence comprising of a legal move checker, tree searching algorithms and extensive heuristics. Each of these components has accompanying tests and documentation to prove its success. The program allows for life and death problem creation as well as play through, either against the AI or another human. The AI itself makes use of the minimax tree searching algorithm combined with the alpha-beta search algorithm to allow for precise move selection. Implemented heuristics based upon Go strategies improve these move selections further during life and death problems within the program. With a fully-developed AI, the project's end program has the heuristic capabilities to play against humans with higher skills levels and also the ability to reach specific objectives of harder to solve life and death problems within efficient running times.