University of Glasgow | School of Computing Science

Team I: Go Problem Solver With Alpha-Beta Tree Search

Eilidh Anderson
Jamie Dale
Scott Hood
Kiril Hristov
Niklas Zwingenberger

Level 3 Project — 27 March 2015

Abstract

The abstract goes here.

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Name: _____ Signature: _____

Contents

Chapter 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. A graphical user interface is another 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 within 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.

Chapter 2

Background

2.1 Go Background

Go is a two player board game that originated in China more than 2,500 years ago and is still mostly played in its original form. Players have ranks from 30-1 Kyu, and 1-9Dan where Dan ranks are for grand masters. It is a territorial game. Simply put the aim of Go is to surround more territory than your opponent. The board, usually marked with a grid of 19 lines by 19 lines, may be thought of as a piece of land to be shared between the two players. One player has black stones and the other white stones. Each player takes a turn to place a stone; stones are placed on the intersections of lines and not in the squares themselves. The game continues until either both players pass or there are no more legal moves for each player. Some of the basic rules are detailed bellow:

- The board is empty at the beginning of the game unless the players agree to a handicap (If one player has a handicap of 3 then that player will start with 3 stones on the board at the start of the game).
- Once placed on the board, stones may not be moved, but stones may be captured. This is done by surrounding an opposing stone or group of stones.
- A player may pass his turn at any time.
- A stone or solidly connected group of stones of one colour is captured and removed from the board when all the intersections directly adjacent to it are occupied by the enemy(The group has no liberties).
- A player's territory consists of all the points the player has either occupied or surrounded.
- The game is won by gaining the most points, which are determined by:
    1. The number of pieces captured.
    2. The amount of territory held at the end.

2.1.1 Go Terms Relevant For Life And Death Problems

Liberty: A liberty is an empty intersection that is adjacent to a stone or to a connected chain of stones.

Eye: Eyes can be described as internal liberties of a group of stones that, like external liberties, prevent the groups capture but are much harder for an opponent to fill in. Eyes are very significant for life and death problems as the existence or non-existence of eyes in a group determine whether that group is alive or dead. A group with no eyes or one eye will die unless its holder can develop them into two eyes. A group with two or more eyes will live because it is impossible to remove all liberties of the group in one move by the attacker. In figure 2.1.3 all internal liberties with a red circle can be described as eyes.

INSERT EYE PICTURE

Atari: Atari is a term used in Go for a situation where a stone or group of stones only has one liberty and can be captured on the next move unless the defending player places a stone on the liberty thus giving the group more liberties and at least temporarily taking the group out of Atari. In figure 2.1.1 above if black places a stone at any of the liberties with a red circle then black will capture the white stone. In figure 2.1.2 if white places a stone at either the liberty at a or the liberty with a red circle then the black stones will be captured.

INSERT ATARI PICTURES

Hane: A Hane is a move that wraps around the opposing players stone or group of stones and doesnt touch any of the players stones or a diagonal move played in contact with an enemy stone. There are many positions where a hane is considered a good move [*find a position where a hane would be a good move*]. In figure 2.1.4 the stone labelled with a one is a hane as it wraps around the black stone and isnt adjacent to another white stone.

INSERT HANE PICTURE

Ko: Ko describes a situation where two alternating single stone captures would repeat the original board position. These alternating captures could repeat indefinitely creating an infinite loop in the game. The Ko rule stops this infinite loop from occurring: If one player captures the Ko, the opponent is prohibited from recapturing the Ko immediately. In figure 2.1.5 above black can capture the white stone with the red circle by a play at a. The resulting position is shown on the right. Without a Ko rule, in this position White could recapture the black stone with the red circle by playing at b, which would return the board to the position on the left and then Black could also recapture creating an infinite loop. So, if in left position of the diagram above Black captures at a, White may not play at b for his first move after the black capture. Instead White has to play elsewhere. After that Black can choose either to win the Ko by playing at b in the right position in the diagram, or to play elsewhere as well. Playing elsewhere however would allow White to take the Ko back, since the recapture restriction is only valid for the next turn.

INSERT KO PICTURE

Miai: Miai can be defined as a pair of empty points that have the same value. For example if black plays at A, white can play at B and suffer no disadvantage from the exchange. Equivalence is an important aspect of miai, in the sense that the two options allow the same objective to be achieved

more or less. For example, miai in a life and death context might mean the existence of two different moves resulting in the same outcome of a group living or being killed. In figure 2.1.6 above both point a and point b have the same value in terms of the life and death of the group if black plays at a then white plays at b and vice versa. If white places a stone at either a or b he creates a second eye for the group and thus keeps it alive.

INSERT MIAI PICTURE

Seki: The term Seki translated into English means mutual life, this situation occurs when two live groups share liberties which neither of them can fill without dying so neither player will. In figure 2.1.7 above the white group of stones and black group of stones with red circles share two liberties a and b. If either player plays into one of these points the opposing player with play into the other and capture the opposing players stone so neither player will. [*maybe include another seki example where the group has eyes*] .

INSERT SEKI PICTURE

2.2 Go Problems

2.2.1 Why Life And Death?

The search space for Go's game tree is both wider and deeper than that of chess. It has been estimated to be as big as 10^{170} compared to 10^{50} for chess. It is possible for the search space for a life and death problem to be larger than all of chess. The universe just has 10^{82} atoms in it at most. With the timescale and number of people on this project it would not be feasible for us to create an AI to play a full sized game of Go. Therefore it has been decided to concentrate our efforts into solving some life and death problems which is a more realistic yet still challenging goal. Furthermore, the project initially focused on a small subset of life and death problems before moving on to more computationally challenging problems once the initial problems were being solved with a reasonable success rate.

2.2.2 Common Life And Death Problems

Most life and death problems be grouped into a shape or a technique used to solve the problem given the limited time scale of the project, the number of different type of patterns the program can solve is constrained to a small subset of the common problems. However with that said one small positional change of a stone can a big difference to the complexity of the problem.

2.2.3 Unsettled Three

This shape is concerned with when a groups interior eye shape consists of three spaces in either the form of a bent three space or a straight three. Here the key number is 3 if the space is any less than three then the group is dead, four spaces and the group is alive. In figure 2.2.1 above if white plays

at either of the red circles the group lives because it creates two eyes for that group if black plays at either of the red circles it kills white.

INSERT FIGURE

2.2.4 Six Die, Eight Live

The groups of stones in this section consist of rows of stones on the second line in from the edge of the board. Therefore the groups eye space lies on the edge of the board. When the group is away from the corner the rule is six die, eight live.

Chapter 3

Design And Implementation

3.1 Requirements Gathering

The original product specification given by the formal product specification was to create *an interactive program, with GUI, that presents a go problem on a board, and interacts with the user as they explore the solution space of the problem. The program uses tree search algorithms to find good moves*. From this, some ambiguities were identified in the requirements such as what tasks, goals, and objectives would be explored in the program. So a requirements gathering interview was arranged with our project supervisor, Dr. John ODonnell, to help specify the requirements further.

For this interview, any ambiguities in the project requirements were identified and questions intended to reduce them were formulated. Questions were also created that would identify more requirements, so that a more specific requirements document could be created at the end of this process.

3.1.1 Interview Questions

Nature Of Problem

What are the types of objectives the program will be asked to solve?

- It is hard for the program to determine the nature and goal of a problem as they are laid out in the problem book. The program would have to choose its own boundary for the problem and determine which pieces are irrelevant to the overall solution of the problem. This is too ambitious, so a boundary should be defined along with a nominated group of stones that should either be captured or saved.

What difficulty of problems (30 kyu - 9 dan) should the program be able to solve?

- It should be able to solve problems correctly at as strong a level as possible. At a minimum, it should be able to solve simple problems using brute force. The program should be tested

with problems of an estimated strength against a competent human player to find the kyu ranking of the program. 20 - 25 kyu is around the right ranking that the program should be able to achieve. If the program ends up better than this, problems of better ranking could be used to improve it further.

How complex (number of moves) will the solved problems be expected to be?

- The number of moves to solve is a useful metric for calculating complexity when we use a brute force algorithm, however for more complicated heuristics it becomes less useful. A directory of collections of test cases should be created so that many tests cases are able to be ran over the program and calculate the complexity. However calibration is difficult as humans and computers find different things difficult. Humans can perceive good moves and can deliberately structure the tree deeper rather than wider by forcing the other player to make specific moves to save their pieces.

Solution

How long should the AI have to respond?

- The program should have no time limit to solve, but it should be reasonable. ie. making a move in a few seconds or minutes compared to days and weeks. The interface should have a way for the user to define how long the AI has to take a turn. Thirty minute turns are too long in practice. To control time, a difficulty chooser could be implemented that would set time limits for turns or set depth limit for move searches. This should be a rough guide and not a set limit ie. if the time limit is set to 30 seconds, the AI can take from 10 - 60 seconds. As long as the times are reasonably similar.

Whom should the AI be able to play against?

- It should be able to play against itself or against humans.

What minimum success rate should the AI have?

- The program should be able to solve 28 kyu problems with certainty. It may be unable to solve 15 kyu problems reliably but may solve them on occasion.

How do we know the AI has succeeded?

- Maybe a main group could be elected to either be captured or be saved depending on problem. If the group is captured, or is unable to be captured then the program could be alerted that the problem has been solved. Pitting the AI against itself is not a good way to test the successfulness of our program, so the program should be tested with humans who know how to solve the problems so they can pick out problems in the program. There are two avenues that the testing process should pursue, the first being Small data set; in depth evaluation, where good Go players are asked to make good moves against the AI on a problem and then

play again but make bad moves. Then they should be asked if they think the AI is good in the program. Ideally, there should be 5 (minimum) to 10 (ideal) testers for this step. The other testing avenue is Large data set: low depth evaluation, where a person, aided with the answers to the problems, is asked to test the first move the AI makes of many problems (several hundred). If the AI makes the correct first move, it can be assumed that the program is likely able to find the correct solution.

Graphical User Interface

Is a textual user interface required in addition to a graphical user interface?

- Maybe start the program in TextUI, then have a command to start GUI but the ability to be able to switch between TextUI and GUI no needed. A TextUI is important for the portability of the program, as it can be hard to get GUIs to work on different systems.

What information should be shown from the AI?

- The ability to go back through the moves showing: What moves were considered; What the heuristic evaluated the value of the moves as; Why it did not take moves that would be considered as good; Why it took moves that would be considered as bad. Also, make it possible to query the AI, to ask it what moves it is considering or show the value the heuristic gives for a specific position. This would help to find the bad decisions in AI so that fixes are more easily targeted.

Also, after a discussion with Dr. John ODonnell, it was decided that the program would be created with the purpose of solving Life and Death problems as solving the entire game would be far too ambitious for a project of this length.

3.2 Requirements Specification And Organization

From the initial product specification and the requirements and refinements gathered in the interview, the identified ambiguities were removed to create a more descriptive requirements specification: The resulting program of this project should be able to solve Life-and-Death Go problems. This should be done either independently by an AI and/or with user interaction via a GUI that shows the board and allows legal moves to be made. The AI should be able to be given an objective such as defend group A or take group B for either colour and be able act accordingly. The problem difficulty for the AI to solve should be 15 Kyu (on a range from 30 Kyu - 9 Dan) and it should solve the problem correctly with a success rate of at least 80

Using this new specification, the planning of the program began. First, the GameEngine, the GUI, and the AI were identified as the three main components of the program. As the GUI and AI could not work without the underlying GameEngine, the GameEngine was the first milestone. The GameEngine was split up between all members of the team, where each person then had a section to implement. This first milestone was completed in November when a text based UI was also created so that it was possible to interact with the program before the GUI was implemented.

After the GameEngine was implemented, the team was split into two: Eilidh, Jamie, and Scott began working on the GUI; and Kiril and Niklas began working on the AI. The AI was identified to us by Dr. John ODonnell as being the bulk of the work in this project, so the GUI was planned to be finished in January, so that the GUI team would be able to join Kiril and Niklas in working on the AI from January until the end of this project in March.

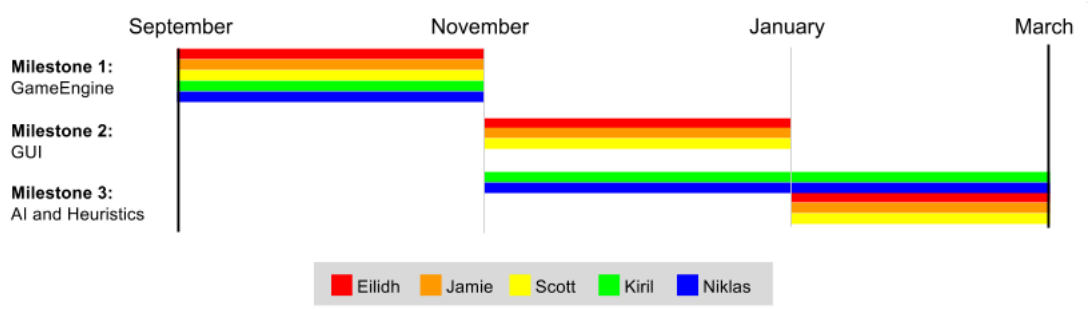


Figure 3.1: Gannt chart showing organization timeline.

3.3 Game Engine Design And Implementation

3.4 Graphical User Interface Design And Implementation

To create a fully-fledged Go Problem solver allowing the user to interact with life and death problems, a graphical user interface had to be designed and implemented to allow for communication between the user, GameEngine and artificial intelligence. The finished GUI was created using Java Swing [reference here] and the Graphics package [reference here]. The user interface provides a wide range of functionalities through graphical board representation and the implementation of interactions such as placing stones and saving board states. Hence the final design of the GUI successfully allows the user to create and play life and death problems as specified within the requirements.

3.4.1 Initial Design

During the preliminary design process of the GUI, a paper prototype was created as a basis for implementation. As can be seen in Figure 3.1, the paper prototype showed the original layout of the user view of the GUI which consisted of several elements. These elements included a representation of the board, featuring white and black stones where the human and AI would both play moves. Another component that can be seen are the radio buttons that were drawn out, which would allow the human and computer's stone colours to be chosen. Lastly, an undo button and a reset button were also included in the paper prototype. The undo button would remove the latest move on the board, whilst the reset button would adjust the board back to its original loaded state. The original design also included a toolbar featuring a single button, labelled "Menu". Pressing the menu button reveals the options seen in Figure 3.2 for loading and saving the board, as well as saving a log which would be a series of played moves. Whilst, the paper prototype was a strong basis for the beginning of implementation it was missing a wide range of functionalities, such as bounds and player modes,

which were included within the final implementation. Some redundant options within the original paper prototype's file menu were also removed during implementation, such as the save log.

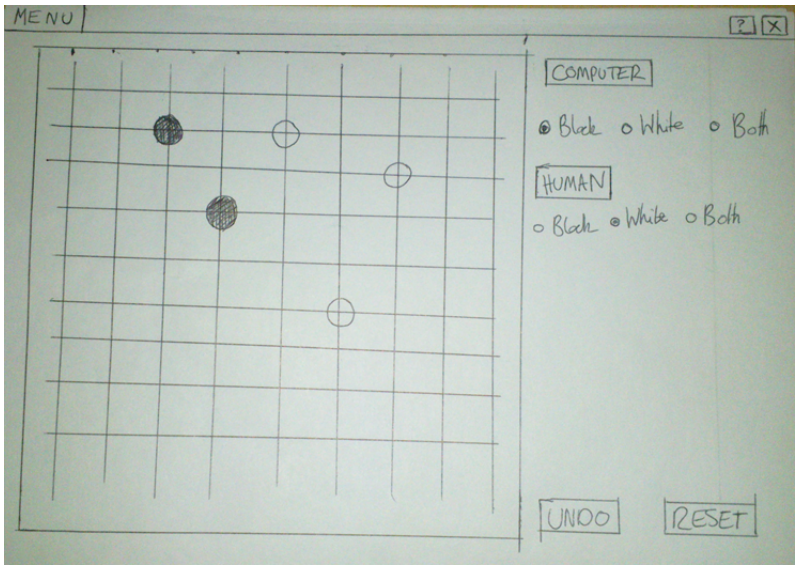


Figure 3.2: Original paper prototype GUI view.

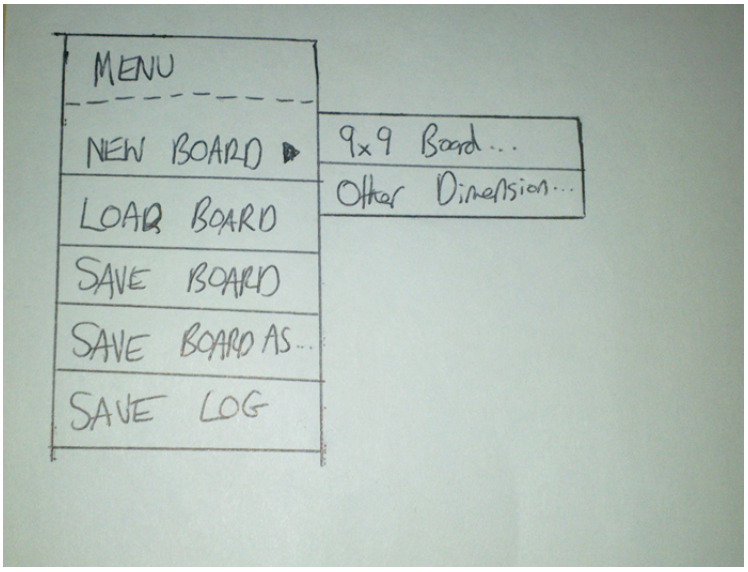


Figure 3.3: Original paper prototype file menu.

3.4.2 Visual Implementation

The first step of implementing the GUI was the graphical board representation, which can be seen as the main visual feature of the problem solver. As described previously (see *Section 3.3*), the GameEngine uses a 2D array to represent the board within a Board object. This 2D board array had to be translated to a graphical representation to enable the GUI to communicate directly with the GameEngine and AI as well as the user. As the GUI was created using Java Swing [reference

here], the Graphics package [reference here] was utilized to provide a visual of the board. To depict the board on screen, the current number of board lines (height/width) are taken from the Board object specified within the GameEngine. A series of brown rectangles are then drawn and filled in to represent the board using the number of specified board lines. These rectangles overlap to form the appearance of a grid and again using the line number provided, coordinates are drawn aligned to each of the rectangles. The user can control whether these coordinates are shown or not and they can be a useful aid during play. This described method of drawing out the board means that the GUI is flexible to representing any size of board specified by the user, as long as it is a square. Following on from the successful drawing of the board layout itself, the GameEngine's 2D array board representation is scanned and when a stone was found - being Integer 1 for black and Integer 2 for white - the appropriate counter is drawn on the board. The counters are painted in a top-left across and down fashion, comprising of a fully coloured circle with a given edge around them. Hence, a full board containing a grid, counters and available coordinates is drawn as seen in Figure 3 using the 2D Board object array provided to the GUI.

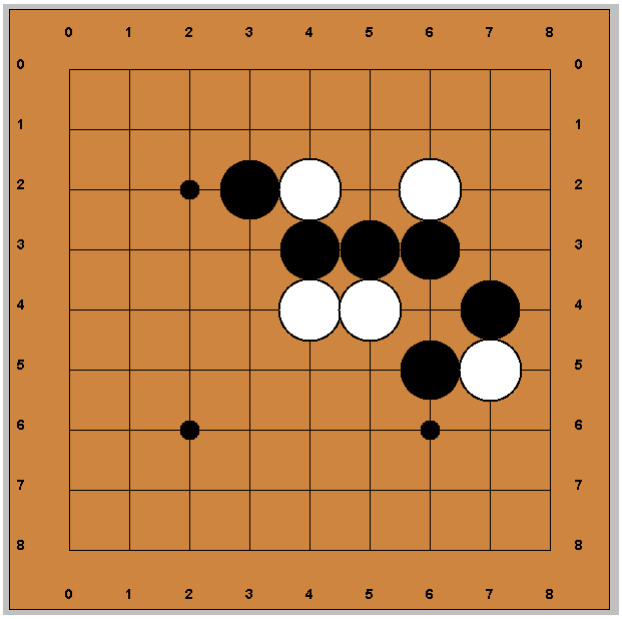


Figure 3.4: Full board (grid, counters and coordinates).

Several other visual enhancements were made following the original drawing of the board, allowing for a higher level of feedback and a more user-friendly system. These included the use of transparent stones upon the board when the user's mouse hovers over an intersection, being where a stone can be placed upon the board. These transparent stones are colour-coded and are either black, white or red - which signifies an illegal move through utilization of the legal move checker, an important part of the GameEngine as described previously (see Section 3.3). The see-through stones are depicted in Figure 4 and were also implemented using a 2D array, likewise to the current board representation. This 2D array refreshes itself each time a mouse move is detected. After this refresh, a stone is placed within the transparent stone's 2D array at the current location of the mouse. Checking whether the current stone location is legal or not occurs after the updating of the 2D array and once the check occurs, the appropriately coloured see-through stone is displayed.

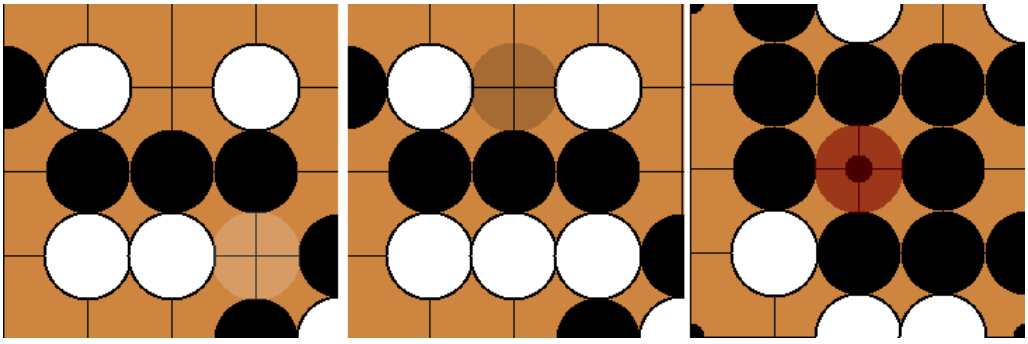


Figure 3.5: Transparent stone: White, black, illegal move.

Yet another visual enhancement within the GUI can be seen in Figure 5, being the depiction of the artificial intelligence's search space as bounds. These bounds are depicted graphically and represent the board positions that the AI will consider moving to. The intersections that are available to the AI are coloured brown, whilst the rest of the board is greyed out. This graphical feedback allows the user to easily see the multiple positions which the computer will possibly move to and this feedback can be toggled on or off according to the user's preference. Originally the bounds took only the shape of a rectangle or square between two specified user points, however through a process of improvement the finished GUI allows the user to select a flexible set of bounds simply by clicking on the intersections to be included within the AI's search space. This change not only provided better visual feedback but was also more efficient for running the artificial intelligence whilst in-game.

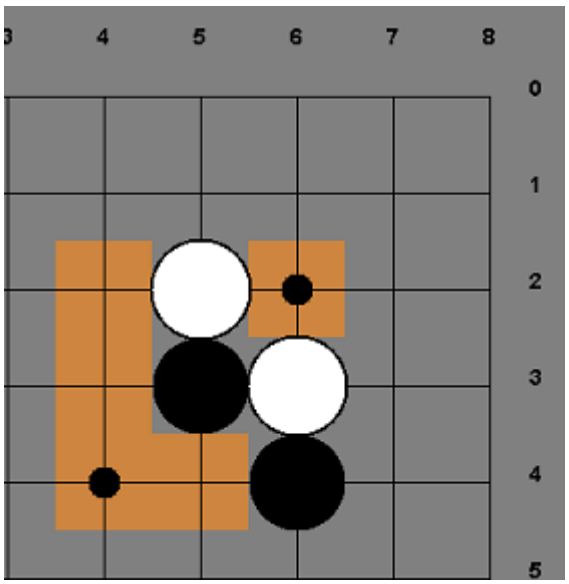
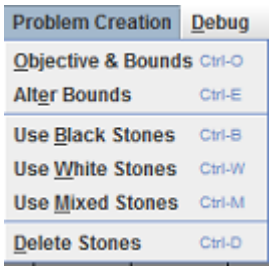


Figure 3.6: Artificial intelligence's graphically represented bounds.

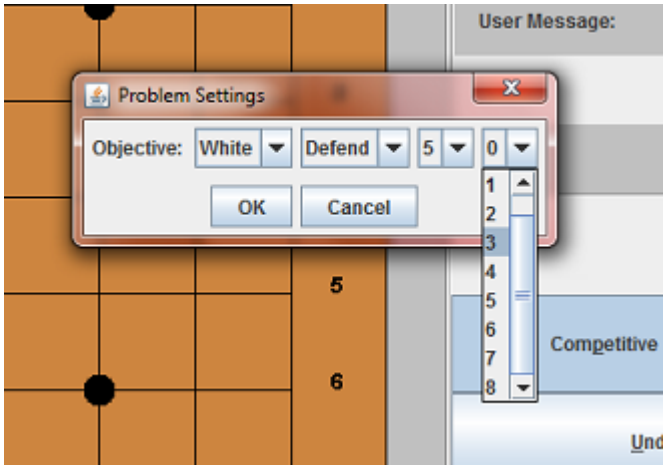
3.4.3 Player Modes

Following the implementation of the board representation itself, as well as several other visual features, a major design change from the paper prototype took place. The design decision was made

to implement two separate user playing modes, being "*Creation Mode*" and "*Competitive Mode*." The availability of user modes allows for easy distinguishing between the two main features of the Go problem solver - being allowing a player to create a problem or playing out a problem. The creation of a problem provides the user with various options whilst playing out a problem allows the user to choose between playing against the AI or another human. The separation of the two modes also rids the GUI of the possibility of having overlap of certain features between modes, which could create numerous errors - for example, deletion of stones is only available in creation mode. Each mode also has their own dedicated menu upon the toolbar, named "*Problem Creation*" and "*Competitive Play*." When the GUI is initially started, it is opened in problem creation mode and displays a blank 9x9 Go board - ready to begin letting the user to design life and death problems. The problem creation menu in the toolbar, as seen in Figure 6, allows access to various features during the creation of problems within the GUI. These include options during the placement of stones, such as using only a specific colour of stone (white/black) or being able to delete stones by pressing on them. Problem creation mode also allows the user to alter the AI's objective and bounds. The objective is specified in a pop up box, as displayed in Figure 7, whilst the bounds can be altered by the user selecting bound selection and then specifying their bounds by selecting specific intersections. This collection of features easily allows for problems to be made quickly and can then be saved by the "*Save Problem*" option in the file menu, ready to be played in competitive mode.



(a) Problem creation menu.



(b) Objective dialog choice box.

Figure 3.7: Creation mode user options.

Competitive mode meanwhilst is intentionally selected by the user after the initial opening of the GUI. This mode allows for the play through of a problem in several approaches, including human versus human, human versus AI (either Minimax or AlphaBeta) and AI versus AI. Upon selection of the "*Competitive Mode*" button a pop up box appears allowing for player choice as seen in Figure 8. If the bounds and objective are not specified before this selection, the GUI will prompt the user to choose them before allowing competitive mode to be entered. Once a game is in play, the problem creation menu is greyed out and the competitive play menu is available to use. This menu includes such features as swapping player colours, forcing the AI to move and also allowing the user to change the AI type currently being used within play. An example of an AI move can be seen in Figure 9 and Figure 10, where labels on the side of the board specify the AI type and its move selection which allow the user to easily learn from its position selection, as seen in Figure 11. *WRITE ABOUT HEURISTIC CHOOSER HERE* Hence the options available in competitive

mode allow for a flexible and user friendly environment whilst playing through a problem with the user interface.

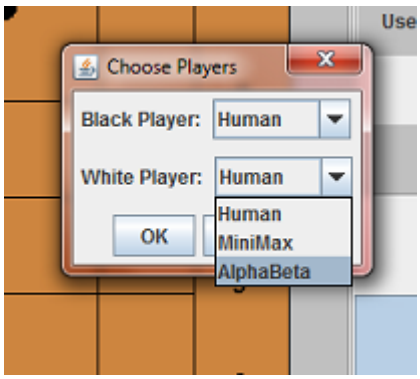


Figure 3.8: Player dialog choice box.

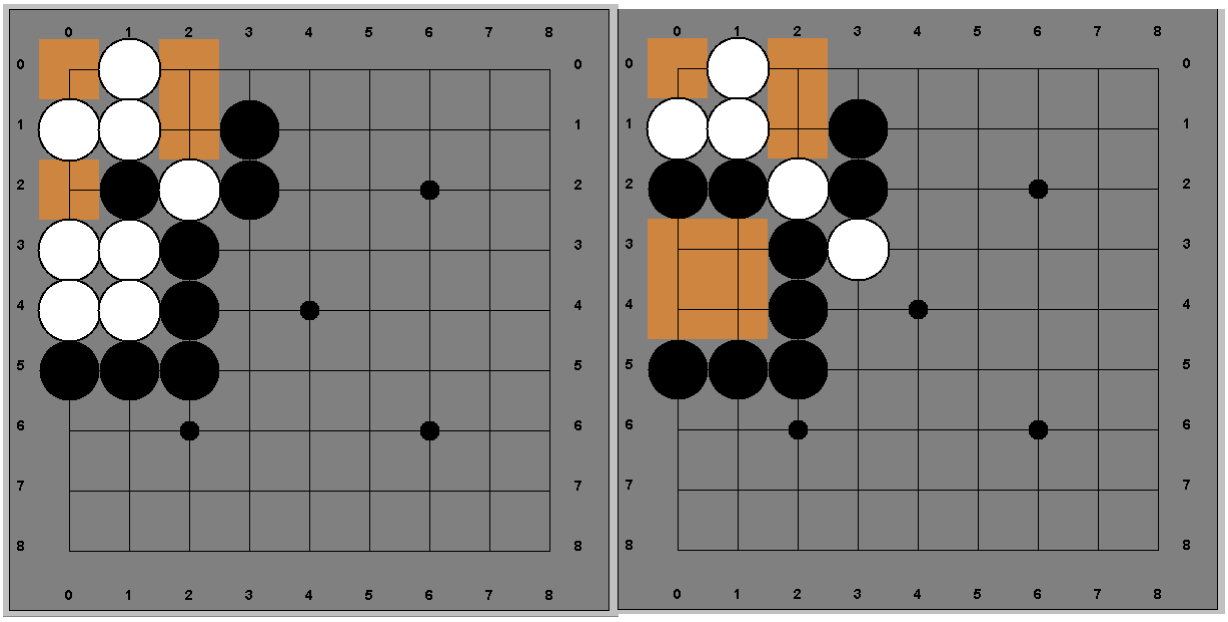


Figure 3.9: Example of AI moving (successfully capturing black stones).

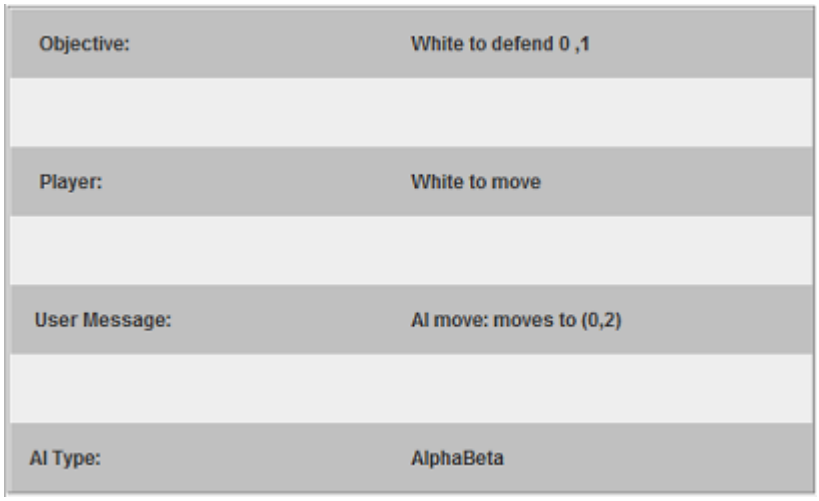


Figure 3.10: AI feedback following its move.

Insert heuristic chooser picture

Several features of the GUI are included in both creation and competitive modes. These features include the "*Show Coordinates*" button which displays the coordinate numbers along all sides of the board, allowing easy stone placement. The user can also load or save problems at any point as well as use debug features. Debug features include displaying the log, being the history of all actions within the GUI. Also, the usage of the help menu is available in both modes which has options including the display of keyboard shortcuts. For example making the AI move against itself in AI versus AI mode can be quickly done using *ALT + I* and there are several other keyboard shortcuts which enable even quicker use of the GUI. Other useful functionalities provided in both modes include the undo and reset button, allowing for the quick backing up of mistakes or reset to the original board, which is practical at all times for the user.

3.4.4 Finalised Graphical User Interface

Through the evidence given prior, it is clear that a user interface was completed successfully allowing for easy interaction during problem creation and competitive play. The entire finished view of the GUI can be seen in Figure 12. The final implementation of the interface allows for the quick making and saving of problems before the user is able to play them in a variety of ways and against several different entities, including the AI and other human players. The GUI also has a wide range of useful features, such as making the AI's functions including search space and heuristics available to the user, as well as having a clean look and layout.

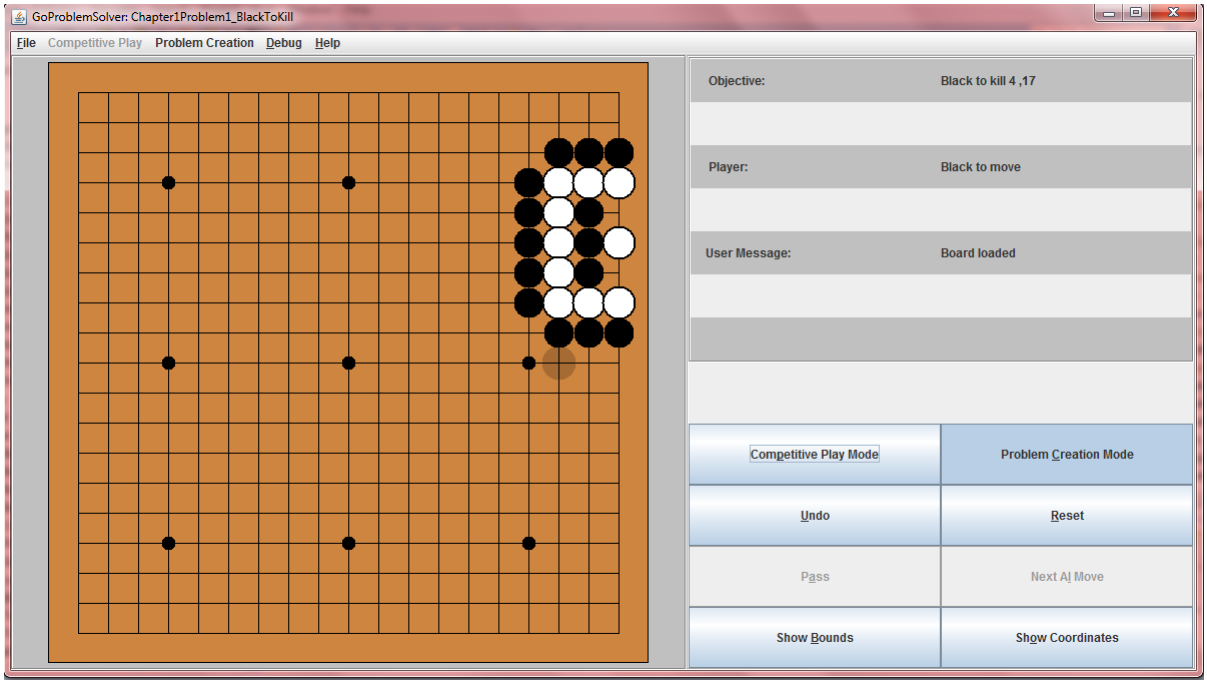


Figure 3.11: Full view of the final GUI.

3.5 Artificial Intelligence Design And Implementation

3.6 Heuristics Design And Implementation

3.7 Integration And Implementation Reflection

Chapter 4

Evaluation

4.1 Solvable Problems

4.2 Subject Testing

4.3 Other Testing

4.4 Testing Reflection

Chapter 5

Conclusion

ADDING CITES HERE JUST NOW SO BIBILOGRAPHY DISPLAYS [3, 5, 4, 7, 2, 6, 1]

Bibliography

[1] Bernd Brugmann. *Monte Carlo Go.* Max-Planck-Institute of Physics, 1993.

[2] James Davies. *Life And Death. Volume Four: Elementary Go Series.* Kiseido Publishing Company, 2nd edition, 2014.

[3] Oracle. *Java 2D Graphics and Imaging*. http://docs.oracle.com/javase/6/docs/technotes/guides/2d/, 2011.

[4] Oracle. *Graphics (Java Platform SE 7)*. http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html, 2014.

[5] Oracle. *Swing APIs and Developer Guides*. http://docs.oracle.com/javase/8/docs/technotes/guides/swing/l, 2015.

[6] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 3rd edition, 2010.

[7] Kano Yoshinori. *Graded Go Problems For Beginners. Volume Two: 25 Kyu To 20 Kyu Elementary Problems.* Nihon Ki-in, 1985.

Chapter 6

Appendix

School of Computing Science
University of Glasgow

Ethics checklist for 3rd year, 4th year, MSci, MRes, and taught MSc projects

This form is only applicable for projects that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please submit an ethics approval form to the Department Ethics Committee.

If your evaluation does comply with all the points below, please sign this form and submit it with your project.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and PDAs is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Project title _____

Student's Name _____

Student's Registration Number _____

Student's Signature _____

Supervisor's Signature _____

Date _____