Enabling Computer to Play Go Against a Human Player

- Naomi Christie
- 200724012
- Matthew Huntbach
- Computing & Information Systems

Abstract -- This project creates a computer opponent for the game of Go using minimax and alpha-beta pruning and is implemented using the Python programming language, the Django web framework and a PostgreSQL database. The game itself is a variation of Go aimed at facilitating the learning of basic moves in the game for beginners.

I. Introduction

i. Background

The ancient Chinese two-player board game Go has attracted media attention in recent years following Google's success in writing Artificial Intelligence capable of beating the world's best human Go player (1). While there are many resources to play Go online both against other humans and against computers (2), there are fewer resources featuring computer opponents aimed at assisting beginners to understand the basic rules. Gomoku is a game played on a Go board in which the aim is to place five stones in a row, horizontally, vertically or diagonally before one's opponent manages to get five in a row, but it loses much of the rest of the game logic, including captures (3). The Robogo project implements a new variation of Gomoku in which all the rules of the game of Go are maintained aside from the scoring method, which is simplified down to: the winner is the first to get five stones in a row vertically or horizontally (not diagonally). The human player will play against a computer allowing them to learn how moves and captures work on a Go board without needing to find a knowledgeable human opponent. While there are resources to play Gomoku against computers online (4, 5) on investigation I could not find this unusual variation in which capture rules are upheld available and therefore it represents a new offering to the Go playing world.

ii. Terminology

This section of the report will cover some terms common in the game of Go.

- Stone: a player's piece, can be black or white
- Group: a collection of stones which are touching on the board
- Capture: when a stone or group of stones is surrounded on all sides they become the prisoners of the opponent
- Ko rule: the board must not return to an identical state during gameplay. This rule prevents the game from stalling
- Intersection: stones are played not within the lines of the grid, but on the cross-shapes which are made by the lines, these are known as intersections
- Liberty: a stone or group of stones which are yet to be captured have free spaces around them horizontally and vertically, these are known as liberties. Once a group has zero liberties it is captured
- Jump: a move in the game which doesn't connect to a stone, but is some number of places away

- Connecting move: a move in the game which links directly to another stone either horizontally or vertically

II. Literature Review

i. Language and framework

The decision to write Robogo in Python was based on two things - firstly the popularity of the language; 58% of respondents to the StackOverflow 2022 developer survey said they had 'done extensive development work' in Python over the last year or 'want to work in' Python over the next year (6), and secondly its status as a back-end language, therefore suitable for the most significant feature of this project which was allowing a computer to play the game.

Flask and Django were both considered for the project. Flask is a lightweight web framework best suited for building APIs, whereas Django is geared towards fuller projects featuring some element of user interface alongside the back-end (7, 8). In addition Django allows for easy integration with databases through its models, providing a 'single, definitive source of information' (9) about the data in the project.

A range of online Django tutorials (10-12) were used as models for how to build up the project, with the greatest reliance on on Django's own official starter project (11).

Various front-end point-and-click graphical options were investigated. Initially the pygame library for Python was considered (13, 14), but eventually was ruled out as it would have been complex to integrate this with a Django back-end in the available time. Using css, html and javascript to render the board was also considered (15, 16), but again on further investigation this implementation would have required a significant amount of effort to be devoted to integration into the Django ecosystem, and can easily become unwieldy (17). Eventually the decision was made to rely only on Django's in-built templates and views, and to render the Go board using unicode characters. There are various conventions for preparing ascii Go diagrams (18), and the eventual design used these as an inspiration, but adapted from these to make the board easier to read as moves were made. Using Django forms (19) was chosen over point and click, again in order to reduce the amount of time spent on front-end coding.

ii. Minimax and Alpha-Beta Pruning

The main resources for understanding minimax and alpha-beta pruning were Patrick H. Winston's 2010 lecture on Search: Games, Minimax and Alpha-Beta (20), and Rashmi Jain's Blog post on Minimax and Alpha-Beta Pruning (21).

Additionally Lane Wagner's blog post on writing binary trees in Python was used as a model for how to build a tree by using a Python class to create the node object (22).

III. Robogo User Guide The user should make sure they have Docker (25) installed on their machine. Then open a terminal at the robogo directory and type in the following command: `./start_game.sh`

Note: if you get an error message saying `permission denied: ./start_game.sh` then please run the following command: `chmod u+x start_game.sh` and then try `./start_game.sh` again.

This should open the game in a web browser. The user should see a page showing a simple 9x9 Go board where the intersections are labelled as plus signs, and the coordinates, 0-8, are labelled along the top and the right hand side.

[INSERT IMAGE HERE SHOWING BOARD]

A form at the top of the screen will allow the user to state which game they are playing (based on their IP address), which colour player they're playing (human user should always play the black player), and which X and Y coordinates they want to play at. When they give coordinates for their move it will appear on the board, and white's response will be displayed. The human user can then input the coordinates of their next move, and so on. The game detects when a user has five stones in a row and then declare that this player as the winner.

Errors: If the user inputs coordinates which are out of range they will forfeit that move and white will play instead. In later iterations this will be replaced with warnings to the user and the opportunity to input valid coordinates.

Capture logic and the Ko rule are yet to be implemented in this iteration of the software.

IV. Robogo program

i. Overview

Robogo is implemented using the Django web framework for the Python language and a PostgreSQL database. Docker containerisation has been employed in order to make it easier to run the code from different computers or servers by automating the process of setting up dependencies. Online resources (23, 24) were referred to in order to set up Docker for this project. The Django framework uses Model, View, Template structure. The Model represents both the classes in the program and the items in the database. The View handles the logic in order to prepare information for the Template, which is the means of viewing the relevant information from a web browser. Django was chosen in order to simplify the coupling of concepts in the code to the database, create some ready-made structure to the code and to ease the rendering of information as a web page. The code to allow the computer to select the best move is stored in `games/minimax.py` and `games/go_minimax_joiner.py` and implements the minimax algorithm.

ii. Minimax with Alpha-Beta Pruning

The minimax algorithm allows the computer to optimise its choice of next move by looking ahead a number of steps in the game by creating a game tree made up of potential moves for each point in the game. The algorithm then assesses the utility of the moves at the given future point in the game. For ease this future point in the game will be referred to as the terminal state, although in reality is will normally be a few moves on in the game and not the end state of the game and greater discussion on the reasons behind that will come later. From this terminal state, the algorithm works back up the tree to establish the most optimal move for each player at each point by inheriting the utility from the terminal state. For a two-player game such as Go the computer will always choose the move with the minimum utility to its opponent and presume that its opponent will select their move for maximum utility, and this alternating minimising and maximising of utility is where the term minimax derives. Alpha beta pruning can be layered over minimax to reduce the amount of computing time required to calculate the optimal move. It works by navigating the game tree to the terminal state of one branch and establishing the utility of the optimal move on that branch, now when the next branch is assessed, as soon as the utility breaches the threshold value set by the first branch we can ignore all paths below that level and avoid these calculations.

In this project minimax is implemented using a base class in `games/minimax.py` called `MinimaxNode`. This class creates a node in the game tree, which in turn has a member variable called `leaves` which is a

list of MinimaxNodes. The `MinimaxNode` is agnostic to which game is being played, and although it's used in Robogo in order to play this variation on five-in-a-row Go, it could equally be used for any adversarial two-player pure logic game, such as Chess. `MinimaxNode` has a method called `get_optimal_move` which can be used on the penultimate nodes in order to determine which terminal node would be selected. The method loops over the leaves of the penulimate node, and selects the one with the best score for the player, so if it's the minimiser this is the score which is worst for the opponent, and if it's the maximiser it's the score which is best for itself.

Alpha-Beta pruning is not yet implemented on this project although some initial investigation has been made and an `evaluate` method has been written based entirely on the pseudocode presented in Jain's blog post (21).

In order for Minimax to work it's necessary to be able to determine the utility of the terminal nodes, which is where the logic of the given game becomes relevant. In order to separate out the five-in-a-row Go logic from the pure Minimax code a new class called `GoNode` was created in `games/go_minimax_joiner.py`. `GoNode` inherits all the variables and methods from `MinimaxNode` and in addition has the ability to examine any given board state. In order to determine the utility of the board state for a terminal node `GoNode` looks for all stones grouped in horizontal or vertical lines, and keeps a running score of the highest number of stones per player and then returns the highest group length as that player's score.

During development the list of potential moves was kept artificially short in order to preserve computing resource, in particular before the alpha-beta pruning was applied. Another benefit to restricting move options is creating a beginner-friendly computer player who takes a simplistic strategy similar to that that human beginners tend to adopt. This beginner strategy involves either directly blocking one's opponenet, or alternatively connecting to one's own stones directly. More sophisticated gameplay tends to involve moves which don't always connect to one's own stones or the opponent's stones.

This initial iteration of Minimax goes only one layer deep, i.e. treats the current board state as the penultimate move before assessing utility. As it only goes one layer deep and also deliberately only makes connecting moves it's currently very easy to beat the computer.

iii. Database

The database software used for this project is PostgreSQL. The database consists of two tables: Games and Moves. Each user can play one game per device as their game is tied to their IP address. Each game can have zero to many moves, and if the game is deleted all of its moves also get deleted from the database.

iv. Front End

The front-end is implemented using Django's templating functionality which in turn uses Jinja scripting. The board is rendered using "+" to represent empty intersections, "●" to represent black stones and "○" to represent white stones.

V. Testing and Error Handling

i. Test driven development

This section of the report will discuss the concept of test driven development (TDD) and how it was used during the writing of the code. Please see `games/tests` for the tests written on the code in this project, further explanation to follow.

ii. Error-handling

This section of the report will discuss the importance of error handling and how it's used in the software. At present there is very little error handling - only one try/except block. As the project develops more error handling will be included.

V. Conclusion

In order to meet the original aims of this project the Minimax algorithm needs to go a number of layers down the game tree in order to make more sophisticated choices for the next move. Alpha-beta pruning also needs to be deployed in order to preserve computing effort in order to calculate possibilities further down the tree. The restriction on moves to just connecting moves needs to be lifted and replaced with jump moves in order to offer intermediate gameplay, and no restrictions for advanced gameplay. In order to implement minimax with alpha-beta pruning recursive methods for nodes should be used, as illustrated in Jain's blog (21). The full capture and Ko rules were not implemented in this iteration and would also need to be coded into the program in order to meet the original goals. One challenge is identifying groups of stones as this is required in order to determine capture logic. A potential solution to finding stone groups would be to add a method to the `GoNode` which would walk the board state and when it finds a stone look for all connecting stones and add that to a stone group object stored in the node.

References:

- [1] https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol
- [2] https://senseis.xmp.net/?GoServers#toc6
- [3] https://en.wikipedia.org/wiki/Gomoku
- [4] https://gomokuonline.com/
- [5] https://gomoku.yjyao.com/
- [6] https://survey.stackoverflow.co/2022/#most-popular-technologies-language-learn
- [7] https://www.guru99.com/flask-vs-django.html#:~:text=KEY%20DIFFERENCES%3A,for%20easy%20and%20simple%20projects.
- [8] https://stackabuse.com/flask-vs-django/
- [9] https://docs.djangoproject.com/en/4.0/topics/db/models/
- [10] https://codyparker.com/django-channels-with-react/8/
- [11] https://docs.djangoproject.com/en/4.0/intro/tutorial01/
- [12] https://www.linkedin.com/learning/django-essential-training/what-is-django?autoplay=true&contextUrn=urn%3Ali%3AlyndaLearningPath%3A5d546c44498e876bef6651ba&u=84146594
- [13] https://towardsdatascience.com/python-miniproject-making-the-game-of-go-from-scratch-in-pygame-d94f406d4944
- [14] https://github.com/thomashikaru/gogame
- [15] https://levelup.gitconnected.com/how-to-make-a-go-board-with-css-ac4cba7d0b72
- [16] https://codepen.io/viethoang012/pen/ygzZaB

- [17] https://www.saaspegasus.com/guides/modern-javascript-for-django-developers/client-server-architectures/
- [18] http://www.weddslist.com/goban/
- [19] https://docs.djangoproject.com/en/4.0/topics/forms/
- [20] https://www.youtube.com/watch?v=STjW3eH0Cik
- [21] https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/
- [22] https://blog.boot.dev/computer-science/binary-search-tree-in-python/
- [23] https://blog.logrocket.com/dockerizing-django-app/
- [24] https://docs.docker.com/samples/django/
- [25] https://docs.docker.com/get-docker/

Note: the final project report write up will feature Vancouver-style referencing as outlined in the following resources:

- https://www.imperial.ac.uk/admin-services/library/learning-support/reference-management/vancouver-style/citing/
- https://guides.lib.monash.edu/citing-referencing/vancouver#:~:text=Vancouver%20is%20a%20numbered%20referencing,the%20corresponding%20in%2Dtext%20references.