Chess Engine AI

Author: Leon Szabo (201515134)

Project Supervisor: Flávia Alves

Contents

1 Project Description

The popular board game chess has recently gained a surge in popularity[1] with new players looking to improve their skills. As a result, chess engines are needed more than ever so that players can improve their skills by analysing misplays in games in an offline setting. That is especially the case for players at the highest level, where a game analysis tool can significantly decrease the time required to look through the possible move variations and find which moves with slight game variations are better than looking through the possible moves without an AI system.

The chess engine AI we plan to create aims to provide a GUI with a competitive algorithm that will allow players to challenge the AI. We will write the back-end and front-end in Go, using the Fyne[2] package for the GUI. As for the chess engine's search and evaluation, we will use alpha/beta algorithms.

2 Aims & Objectives

2.1 Aims:

- Program a chess engine
- Create a stylistic and simple GUI
- Add player vs AI compatibility

2.2 Objectives:

- Develop a chessboard representation
- Develop a chess search and evaluation algorithm
- Implement automated testing through GitHub
- Evaluate performance of chess engine
- Add intractability to GUI

3 Key literature & Background Reading

There have been many chess engine creations over the years, and with every year, resources have expanded, and knowledge about chess and AI has furthered.

The first part of creating a chess engine is the board representation, a way to characterise how a computer sees the board and where the pieces are. There are two categories that representation falls into, piece-centric and square-centric[3]. Square-centric styles tend to be simple; they also tend to be slower and more inefficient. Therefore, piece-centric styles are preferred as they offer a better

way to evaluate chess positions giving a considerable performance benefit over square-centric models. Bitboards fall in the piece-centric class[4].

After board representation, there is the search and evaluation. Depth-First search is a commonly used category of search algorithms; other algorithms, like Breadth-First, try to explore the whole domain of possible moves; in a game like chess, an estimated 10^{43}[5] possible positions is not feasible for an algorithm to process. So instead, Depth-first aims to explore each branch before backtracking; we will use alpha-beta, a commonly used expandable Depth-first search algorithm[6].

4 Development & Implementation Summary

The development environment we will be using is Visual Studio Code[7], as I have used it throughout my course and have become comfortable with it. Another reason for its usage is that it natively supports various languages, including Go[8], the programming language we will be using. We are using Go because the language is fast, simple, and maintainable[9]; I have also become fond of the language as it has many features that make it easy to write and maintain, such as the built-in testing library[10].

1. Create Chess board representation and rules:
Board representation is needed to evaluate moves, detect attacks, and hold positions. In essence, it is the backbone of the engine. Therefore, having a good board representation is essential. So we plan to *create a bitboard* and *use bit operations*[4] for the rules of Chess.
2. Program Chess AI:
There are two main types of chess AI: hand-crafted evaluation (HCE) and multi-layer neural networks[11]. We will be *making an HCE algorithm using minimax and alpha-beta pruning.*
3. Create a stylish and simple GUI:
All chess engines incorporate similar placements of GUI elements[12][13][14]; the chess board on the left, taking up the majority of the screen; and the point value of the winning side on the top right. For the standard, *we will be using the same placements.* For the style, *we will use a minimalistic GUI.*
4. Add player vs AI compatibility:
Before adding this compatibility, we must have already finished the AI vs AI compatibility. Adding AI vs player compatibility will require *being able to move the chess pieces* and *adding a helper line* to show where the pieces clicked on can move, among other features.
5. Evaluate the performance of the AI:
After the chess engine's creation, it will be *tested against chess engines of increasing difficulty* to get game outcomes in a parsable format. A ranking algorithm such as Ordo[15] can then *work out its rating against other chess engines* by receiving a PGN[16] as input and calculating a rank based on the results of what the PGN contains.

5 Data Sources

The only data source we plan to use is from the CCRL website[17]; it is a list of chess engines and their calculated ratings by the CCRL testing group; from this list, we will only be using the free or open-source engines. There is more detail on how we plan to use the engines in section 6.2.

6 Testing & Evaluation

6.1 Testing:

Most of the testing will be automated rather than manually tested, as having a quick way to check if certain functionalities of the chess engine work is better than manually checking them each time we make an edit. In addition, Go, the programming language we will be using, has a built-in testing package, making it easy to incorporate while programming to test semantic errors. Finally, Visual Studio code will catch most syntax errors.

We will be implementing unit testing, integration testing, and GUI testing. Unit testing is essential to ensure all back-end functions work as intended isolated. Integration testing is just as necessary as functions working in isolation will not mean they work together. Go's testing package can be used to automate both of these testing methods; we will be able to program alongside testing with Github actions.

GUI testing is much harder to automate; depending on the scale of the GUI we implement, it may be automated or manually tested. If we decide to use automation, we will use AutoHotKey[18] to simulate user actions.

6.2 Evaluation:

On top of testing the code, we will need to evaluate the strength of the chess engine to see if it performs well. There are two routes to take with evaluation; we could introduce human participants that could play against the AI or introduce another chess engine to play against our AI.

Introducing human participants will bring complications, such as how good a chess player thinks they are against how well they play in reality. There may also not be players good enough to play against the AI and adequately determine its strength; collecting and compiling data may also prove to be time-consuming. That is why we decided to take the AI vs AI route; pitting our AI against already-established engines that have an evaluated rank will significantly increase the precision of our evaluation.

We will start our evaluation by fetching a list of free or open-source engines and their ratings from the CCRL website[17]. Then pit them against each other using Cute Chess's[19] AI vs AI capability to generate PGNs, which can be parsed through Ordo[15] to generate a ranking.

7 Ethical Considerations

Currently, there is nothing that would require us to act ethically. However, we have read the ethical considerations and will follow them if we make changes.

8 BSC Project Criteria

An ability to apply practical and analytical skills gained during the degree programme:
I have learned a lot over the degree programme, with many practical programming skills coming from nearly all courses in years one and two. Analytical skills have also come from programming, as problem-solving is fundamental to programming.
Innovation and/or creativity:
As no program is the same, Innovation will come from programming the back end, especially the AI aspect of the chess engine; we will need to innovate for the chess engine to perform well. In addition, we need to be creative if we want to experiment with AI, and as the chess engine has GUI implementation, design choices will require creativity.
Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution:
There are many resources for chess engines; we will research and apply analytical skills to summarise it into something manageable. As mentioned in section 6.2, we will be using Ordo[15] and Cute Chess[19] for the evaluation.
"That your project meets a real need in a wider context.":
As mentioned in section 1, players use chess engines for analysis; engines significantly decrease the time needed to calculate possible board variations.

There is also a need to make competing software to increase chess engine quality. Without competition, there would be a lack of strong chess AI.
"An ability to self-manage a significant piece of work.":
Following the Gantt chart in section 10 will allow us to see where we are in the project and track our progress to see if we are falling behind. I have learned how to manage time by keeping to deadlines; this skill will allow us to segment work and produce results on dates throughout the project.
"Critical self-evaluation of the process.":
When we finish major parts of the project, such as the bitboard creation, auto testing, and the GUI. We will see what has been successful or unsuccessful and improve what we have built.

9 UI/UX Mockup

As mentioned in section 4.3, chess engines, for the most part, incorporate similar GUI placements; if we decide to defer from the standard, it would have to be for a good reason. Changing it might confuse chess players who are used to the

standard placements. That is why we decided to stick to the standard with a reasonably simple GUI, as seen in figure 1

The chess board is on the left side, taking up most of the space; this is where the player can interact with the pieces; The figures chess board is from Lichess[13]. Our GUI will have a similar style but might use different chess piece designs depending on what we think looks better. The right side shows the chess piece move history in algebraic notation[20]. As our GUI is straightforward now, if we have extra time to work on optional features such as game look back, we can quickly implement them into the GUI as there is plenty of white space.

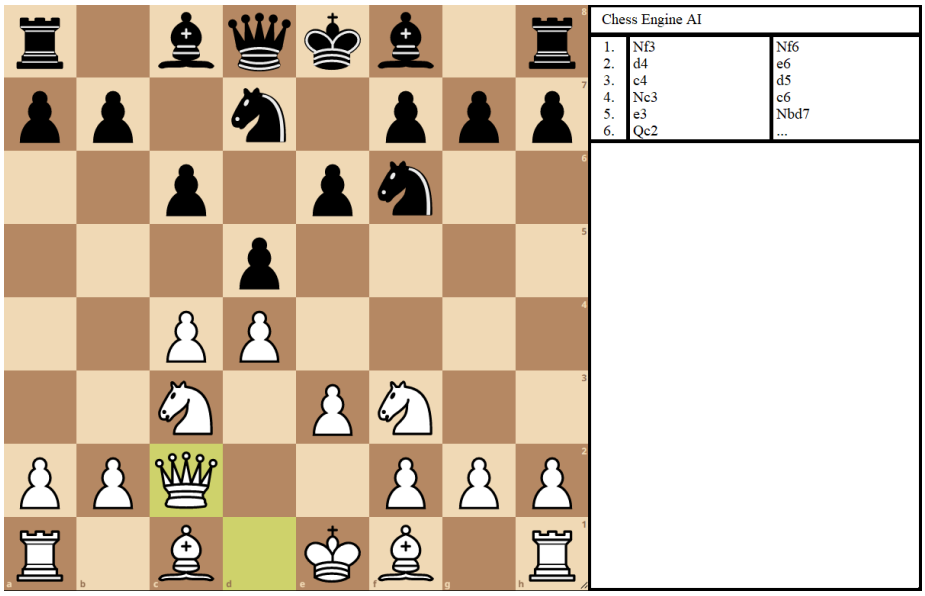
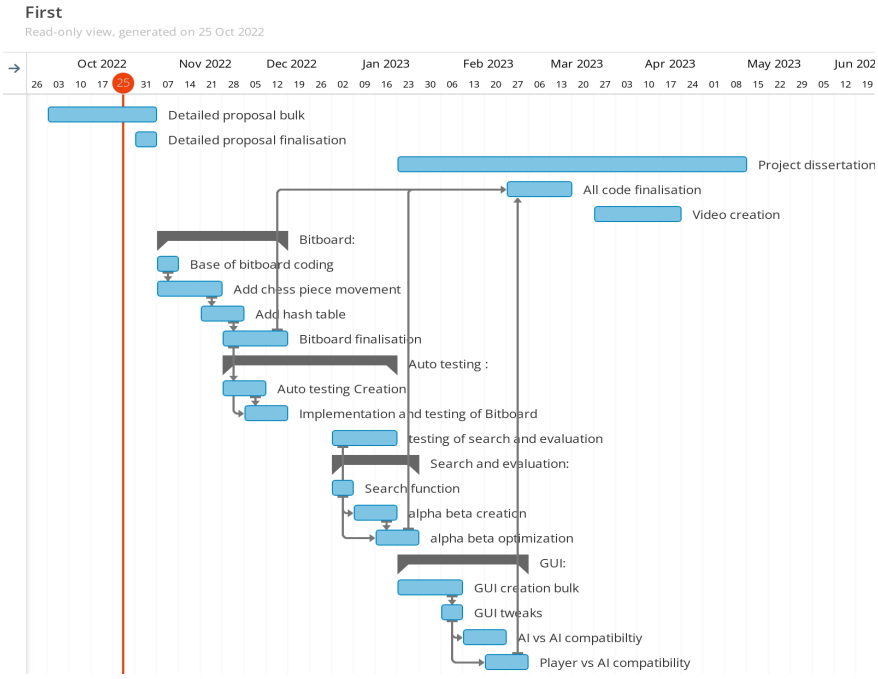


Figure 1: Example GUI for chess engine, adapted from Lichess[13]

10 Project Plan

We have laid out the project plan as a Gantt chart created in Instagantt[21]; there are three main focuses, the bitboard, auto-testing, and GUI creation; we will start writing the honours halfway through the course when there is enough material to start.



11 Risks and Contingency plans

Plans			
Risks	Contingencies	Likelihood	Impact
Producing broken code	Regularly test code	high. There will be a lot of code, we are bound to make mistakes	low. we plan to remove all errors through testing
Low productivity	Have regular check-ins	medium. We will have to balance working on this and other things	medium. This will impact us if we miss a week or two and fall behind

Risks	Contingencies	Likelihood	Impact
Forgetting to save	Make regular backups of work	low. our IDE implements autosaving so saving is still quite frequent	low. auto saves are frequent so there won't be that much loss
Drive failure	Make regular off-site backups on Github	low. Drives do not usually fail	high. Could be detrimental if we have not made backups
Inaccurate deadlines from Gantt chart	Produce working code at the minimum	low. In nearly all cases overestimations have been made	high. It could delay us by weeks if estimations aren't accurate

References

[1] Y. Litwin. "The rise of chess into the mainstream," The science survey. (Mar. 13, 2021), [Online]. Available: https://thesciencesurvey.com/sports/2021/03/13/the-rise-of-chess-into-the-mainstream/ (visited on 10/06/2022).

[2] Fyne. (Jun. 30, 2022), [Online]. Available: https://developer.fyne.io/ (visited on 10/04/2022).

[3] "Board representation," Chess programming wiki. (Jan. 28, 2020), [Online]. Available: https://www.chessprogramming.org/Board_Representation (visited on 10/13/2022).

[4] T. Riegle. "Chess program board representations," Crafty Chess. (Sep. 21, 2016), [Online]. Available: https://craftychess.com/hyatt/boardrep.html (visited on 10/13/2022).

[5] C. E. Shannon, "Xxii. programming a computer for playing chess," *Bell Telephone Laboratories*, vol. 41, no. 314, pp. 2–5, Nov. 1949.

[6] "Search," Chess programming wiki. (Feb. 4, 2022), [Online]. Available: https://www.chessprogramming.org/Search (visited on 10/13/2022).

[7] Visual Studio Code. (Nov. 3, 2021), [Online]. Available: https://code.visualstudio.com/ (visited on 10/04/2022).

[8] "Programming lanuages," Visual Studio Code. (Nov. 3, 2021), [Online]. Available: https://code.visualstudio.com/docs/languages/overview (visited on 10/12/2022).

[9] "Why go? 8 engineers discuss golang's advantages and how they use it.," *Built in*, Jul. 11, 2021. [Online]. Available: https://builtin.com/software-engineering-perspectives/golang-advantages (visited on 10/12/2022).

[10] Golang. (Aug. 2, 2022), [Online]. Available: https://go.dev/ (visited on 10/12/2022).

[11] "Evaluation," Chess programming wiki. (Sep. 9, 2021), [Online]. Available: https://www.chessprogramming.org/Evaluation (visited on 10/12/2022).

[12] chess.com. (Nov. 2, 2022), [Online]. Available: https://www.chess.com/ (visited on 10/12/2022).

[13] Lichess. (Jul. 4, 2022), [Online]. Available: https://lichess.org/ (visited on 10/12/2022).

[14] chess24. (Feb. 28, 2022), [Online]. Available: https://new.chess24.com/ (visited on 10/12/2022).

[15] M. Ballicora, Ordo. (Apr. 19, 2016), [Online]. Available: https://sites.google.com/site/gaviotachessengine/ordo (visited on 10/13/2022).

[16] "Portable game notation," Wikipedia. (Aug. 2, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Portable_Game_Notation (visited on 10/13/2022).

[17] G. Banks, R. Banks, S. Bird, K. Kryukov, and C. Smith, CCRL. (Oct. 29, 2022), [Online]. Available: https://www.computerchess.org.uk/ccrl/ (visited on 10/23/2022).

[18] AutoHotKey. (Oct. 30, 2022), [Online]. Available: https://www.autohotkey.com/ (visited on 10/24/2022).

[19] I. Pihlajisto and A. Jonsson, Cute Chess. (Aug. 9, 2020), [Online]. Available: https://cutechess.com/ (visited on 10/24/2022).

[20] "Algebraic notation (chess)," Wikipedia. (Oct. 31, 2022), [Online]. Available: https://en.wikipedia.org/wiki/Algebraic_notation_(chess) (visited on 11/01/2022).

[21] Instagantt. (Oct. 31, 2022), [Online]. Available: https://instagantt.com/ (visited on 11/02/2022).