(Vecteezy, n.d.)

Herm0ni Chess Bot
Functional Specification 2024-2025

Student Name: Seán Rourke

Student Number: C00251168

Supervisor: Joseph Kehoe

Contents

1. Introduction 3
2. Problem and Proposition 3
   2.1. Problem Statement 3
   2.2. Value Proposition 3
   2.3. Project Scope 3
   2.4. Project Objectives 4
3. System Overview 4
   3.1. System Context Diagram 4
   3.2. Assumptions and Dependencies 4
4. User Characteristics 5
   4.1. User Summary 5
5. Requirements 5
   5.1. Functionality 6
      5.1.1. Core Features 6
      5.1.2. Additional Features 7
   5.2. Usability 7
   5.3. Reliability 7
   5.4. Performance 8
   5.5. Security 8
6. Use Case Diagram 8
7. Brief Use Cases 9
   7.1. Accept Challenge Use Case 9
   7.2. Start Game Use Case 10
   7.3. End Game Use Case 10
   7.4. Set Clock Use Case 10
   7.5. Update Clock Use Case 10
   7.6. Calculate Move Use Case 11
   7.7. Make Move use Case 11
   7.8. Get Move Use Case 11
   7.9. Update Position Use Case 11
8. Detailed Use Cases 12
   8.1. Calculate Move Use Case 12
   8.2. Update Position Use Case 13

References 14

1. Introduction

Herm0ni is an artificial intelligence (AI) chess bot designed to calculate the position of a chess game, the next best move to make, and make that move without the need of human input. This functional specification will go into detail about the key features and objectives of this chess bot.

2. Problem and Proposition

2.1. Problem Statement

The structured environment of complex situations makes chess a great environment for the application of algorithms. The use of AI in chess is not a new concept, with the first program capable of playing a full game of chess being published by Alan Turing in 1951. In 1997, the chess engine DeepBlue defeated world champion Garry Kasparov in a six-game match (Krishnamurthy, 2022). Since this moment, the skill gap between top level chess engines and players has continued to grow. Top level chess engines even compete in their own designated tournaments. Chess bots can also be useful for learning, by being able to play games against an estimated Elo rating without the pressure of a timer or the risk of losing Elo. However, with so much time and effort being put into these top-level engines, the use of AI for learning at a beginning or intermediate level has not seen the same level of effort. Chess.com offers multiple lower-level bots for players to play against, but without paying for a subscription, the number of bots below 1500 Elo rating is a mere nine. This causes the jump in difficulty from one bot to the next to be too steep. Having more of these lower rated bots would allow for more gradual progress and stop players from having to choose between a bot that is too easy and a bot that is too hard.

2.2. Value Proposition

The Herm0ni bot is an attempt to resolve the issue outlined previously, by offering a playable bot of multiple levels of difficulty. The idea of this bot is to fill in some of the large Elo gaps caused by the lack of many lower rated bots currently. Herm0ni will offer various levels of difficulty to provide players of various skill levels with a fair level of competition.

2.3. Project Scope

The Herm0ni bot will be available a bot account on lichess.org, playable by searching the name of the bot. Different bot accounts can be created to allow for various levels of difficulty. As it is on the lichess site, the bot will not display any evaluation or feedback on the site itself. The bot will be playable by any player account on lichess but searching for the bot as an opponent.

2.4. Project Objectives

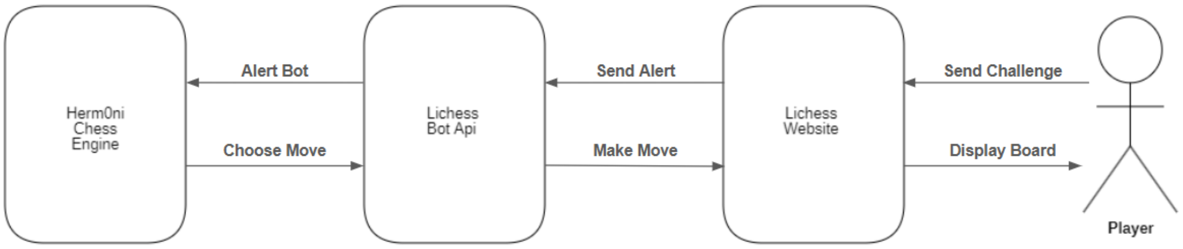
The primary objective of this project is to create a fully automated chess engine that can evaluate a position to effectively determine the next move at a reasonably high skill level. The engine will have a bot account on lichess.org that is playable by searching for its name in the opponent field.

A secondary objective is to create multiple different bot accounts with differing skill levels to allow users to play against a bot that is most suitable to them.

A tertiary objective of this project is personal development. Creating complex algorithms to analyse a chess position and find solutions quick enough to be able to play under time constraints will be a challenging task that I believe will enhance my programming skills.

3. System Overview

3.1. System Context Diagram



This context diagram shows the communication flow from the user to the engine through the lichess.org website and the lichess bot API.

3.2. Assumptions and Dependencies

Assumptions:

- Users have a lichess.org account that they can use to challenge the bot.
- User knows the rules of chess.

Dependencies:

- Wi-Fi connection to use the website.
- Lichess.org is operational.
- Lichess API token.

4. User Characteristics

4.1. User Summary

Users wanting to improve

Beginner to intermediate chess players who are looking for an evenly matched bot that they can play against without the stress of losing Elo. For example, Bob has been playing chess for six months and wants to continue improving but gets nervous about losing Elo rating when playing other users online, causing him to make mistakes. Playing against a bot of his skill level will allow him to still improve, but without the negative impact of a loss, helping to alleviate his nerves.

Users interested in bot vs bot games

Users, possibly other developers, who are interested in seeing chess matches between two different engines. For example, John is a software developer who has recently made a chess engine of his own. He wishes to see how his engine performs against various other engines. John can have his bot play against the Herm0ni bot to help determine the strength of his own bot and what areas it excels or is weak in. For example, if John's bot consistently loses on time, perhaps his algorithms are inefficient.

5. Requirements

This section of the specification will explain all functional and non-functional requirements of the Herm0ni chess bot.

5.1. Functionality

5.1.1. Core Features

Start Game

The Herm0ni bot must be able to recognise that there is a request to start a game being sent to it and can accept this request.

Evaluate Position

Using bitboards and a heuristic algorithm, the current position of a game can be evaluated based on things such as material count, pawn structure, control of key squares and king safety. A bitboard is a data structure representing the chess board, 1 bits are used to represent the existence of a piece on the board in a certain position. The bitboards for each piece can be compared and combined to establish the position after each move (Bijl & Tiet, 2021). The heuristic algorithm will determine the evaluation of a position based on factors such as material count, material position, king safety, etc.

Move Generation

Using a minimax algorithm and alpha-beta pruning, the bot must determine what move to make next. This algorithm works by recursively looking at potential next moves and evaluating the position after that move is made. If a worse position is found after one move, the value moves after that initial move are not considered (Bijl & Tiet, 2021). The bot must always make a legal move.

Game Logic and Rules

The bot must be aware of a game ending due to checkmate, timeout, resignation or draw conditions.

Time Control

The bot must be aware of the time controls set on a game. The most popular time control format is ten minutes with no bonus time (Chess.com, n.d.). The bot can use the amount of time it has remaining to determine how long to spend calculating its next move, so for short games such as one minute or three minutes, both popular formats, the bot should spend less time per move than it does in ten-minute games.

Implementation of Lichess Bot API

The Herm0ni bot will be available as an opponent on lichess.org. Lichess.org is a free, open-source chess server. It is one of the most popular chess websites in the world with over five million games played per day (Lichess, n.d.). Lichess offers the ability to create a bot account powered by an engine, facilitated by its bot API. The Lichess bot API is written in Python and through using Universal Chess Interface (UCI) can send commands, process responses and correctly integrate a C++ executable.

UCI is an open communication protocol used for chess engines. It allows them to communicate with other programs such as graphical user interfaces (GUI) (Chessprogramming, 2024).

Through implementing the engine using a lichess.org bot account, any games that the bot plays can be viewed again in the future.

5.1.2. Additional Features

Opening Book Use

An opening book can be used to recognise various openings and play a pre-determined set of moves to follow the opening. An opening book is a database of common opening lines for a chess game that allows a bot to follow common openings without having to calculate each move (Chessprogramming, 2024).

Tablebase Use

A tablebase can be looked at once there are at most seven pieces left on the board to ensure that the best move is played. An endgame tablebase is a database detailing the best moves to end the game once there are only a certain number of pieces remaining. The most robust tablebase is the Syzygy tablebase and contains moves for up to seven remaining pieces (Chessprogramming, 2024).

5.2. Usability

As Herm0ni is a bot account on lichess.org, all UI features are handled by lichess.org. The bot should be able to be used just by searching its name as an opponent and challenging it to a game. There should be minimal delay between the challenge being sent and the bot accepting the challenge to begin the game.

5.3. Reliability

Playing against a bot will not be useful or enjoyable if it doesn’t reliably perform as intended. The bot must reliably accept challenges to games and play said games without frequent bugs, crashes or other issues. For the purposes and scope of this bot, an issue once every fifty games would be acceptable. Herm0ni must also be able to play to the same level consistently.

Playing against a bot will not be effective as a practice tool if it isn’t consistent in its difficulty. Any average Elo specified for the bot should be accurately represented.

5.4. Performance

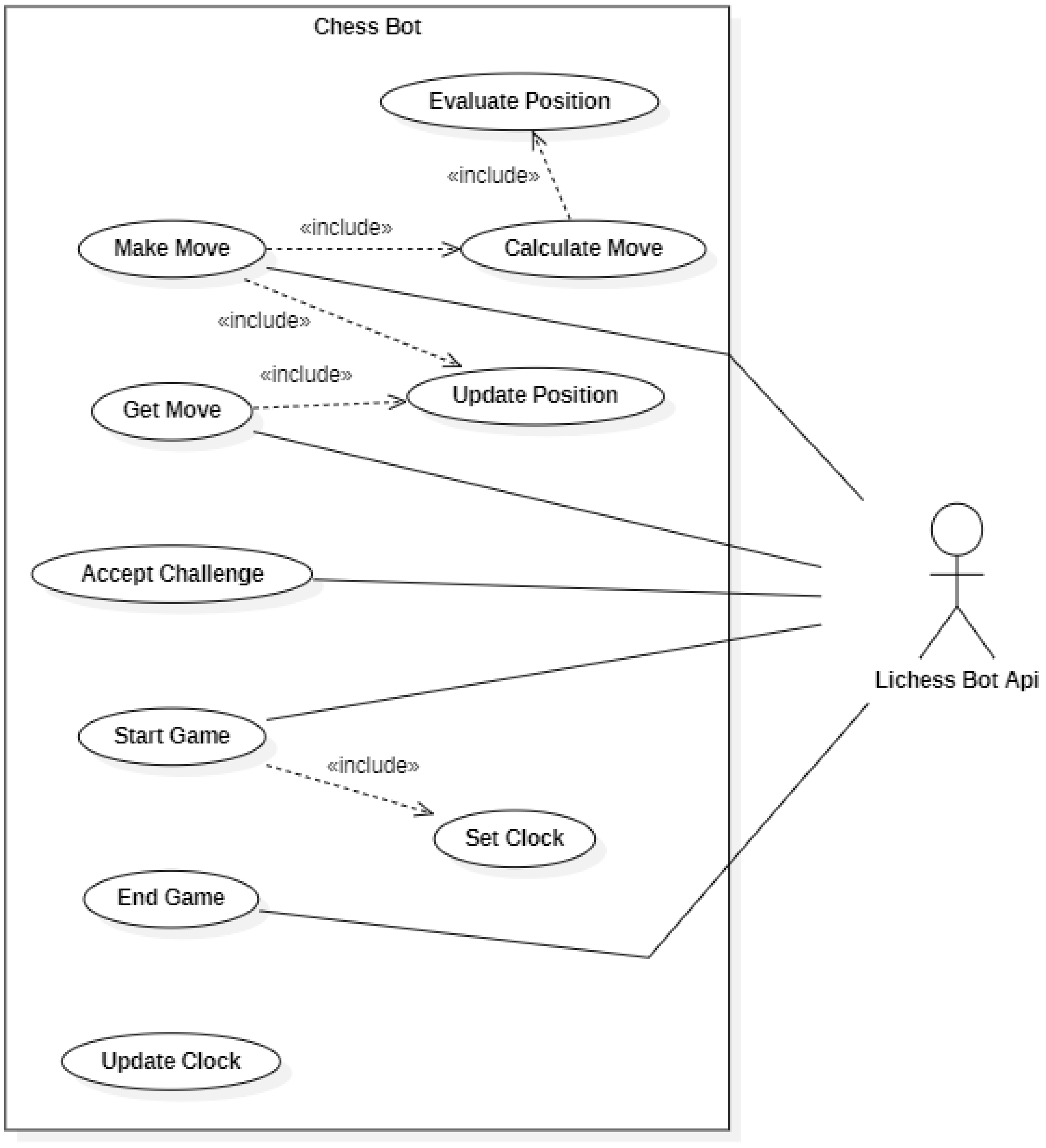
Due to the variety of time constraint formats available for chess games, Herm0ni must be capable of performing its algorithms in a short amount of time, as to avoid losing on time and to not have the user waiting too long for a move. The amount of time spent per move can be determined using the time controls for the game so longer can be spent per move in longer time formats.

5.5. Security

There are multiple security factors to consider with the creation of this bot.

- The Lichess bot API authentication token must be kept secret and stored securely.
- Only one request can be made at a time to respect the API’s rate limits.
- Inputs from the API must be validated to avoid malicious payloads.
- The bot must adhere to all fair play policies outlined on lichess.org.

6. Use Case Diagram



7. Brief Use Cases

7.1. Accept Challenge Use Case

Use Case Name	*Accept Challenge* ***UniqueID*** *UC001*
Primary Actors	Herm0ni Chess Engine, Lichess Bot API
Description	A user challenges the Herm0ni bot to a game on lichess.org. The request is sent to the engine. The bot accepts the challenge is accepted by the bot. The response is sent to the website.

7.2. Start Game Use Case

Use Case Name	*Start Game*	***UniqueID***	*UC002*
Primary Actors	Herm0ni Chess Engine, Lichess Bot API		
Description	Bitboards are initialised to the starting chess position. Time control format is communicated. Clock is set based on time control format. What colour pieces the bot will use it communicated.		

7.3. End Game Use Case

Use Case Name	*End Game*	***UniqueID***	*UC003*
Primary Actors	Herm0ni Chess Engine		
Description	End of game is reached through any valid means (checkmate, resignation, abandonment, stalemate, timeout, 50 move rule, insufficient material, repetition). Memory being used for the game is freed.		

7.4. Set Clock Use Case

Use Case Name	*Set Clock*	***UniqueID***	*UC004*
Primary Actors	Herm0ni Chess Engine, Lichess Bot API		
Description	Time control format that has been selected by the user is communicated to the bot. Starting time is set to correct amount based on this information. Bonus time amount per move is initialised.		

7.5. Update Clock Use Case

Use Case Name	*Update Clock*	***UniqueID***	*UC005*
Primary Actors	Herm0ni Chess Engine		
Description	The bot's clock is decremented during their turn. Once a move is made, decrementing of the clock is paused until it is the bot's turn again. Bonus time per move is added to player's clock once the move is made.		

7.6. Evaluate Position Use Case

Use Case Name	*Evaluate Position*	***UniqueID***	*UC006*

Primary Actors	Herm0ni Chess Engine, Lichess Bot API
Description	Current position is evaluated using a heuristic algorithm. The evaluation is stored as a number with 0 indicating an equal position, a positive number indicating an advantage for white and a negative number indicating an advantage for black.

7.7. Calculate Move Use Case

Use Case Name	*Calculate Move*	***UniqueID***	*UC007*
Primary Actors	Herm0ni Chess Engine		
Description	The evaluation after the next potential, legal moves is calculated and compared. Best potential next moves are evaluated after a subsequent move, assuming the opponent also makes the best move. Any branch resulting in a worse position is pruned. This is repeated to a depth of three moves or until time allocated for each move is reached.		

7.8. Make Move use Case

Use Case Name	*Make Move*	***UniqueID***	*UC008*
Primary Actors	Herm0ni Chess Engine, Lichess Bot API		
Description	The best move is calculated. The selected move is communicated to the site using UCI. The current position is updated using bitboards. The bot's clock is paused, and time increment is added.		

7.9. Get Move Use Case

Use Case Name	*Get Move*	***UniqueID***	*UC009*
Primary Actors	Herm0ni Chess Engine, Lichess Bot API		
Description	User makes their move on the site. This is communicated to the bot. The current position is updated using bitboards. The bot's clock is resumed.		

7.10. Update Position Use Case

Use Case Name	*Update Position*	***UniqueID***	*UC010*
Primary Actors	Herm0ni Chess Engine		
Description	A move is made by either the user or the bot. The bitboard are compared to see what piece moved and to what square it moved to. The bitboards are updated to represent the new position. Whether either player's king is in check is checked.		

8. Detailed Use Cases

8.1. Evaluate Position Use Case

Use Case Name	*Calculate Move*	***UniqueID***	*UC006*
Description	Current position is evaluated using a heuristic algorithm. The evaluation is stored as a number with 0 indicating an equal position, a positive number indicating an advantage for white and a negative number indicating an advantage for black.		
Actors	Herm0ni Chess Engine		
Precondition	• Lichess.org is operational. • A game is in progress with the Herm0ni bot.		
Trigger	A move is made by the player, or the engine is considering what move to make		
Main Path	1. The bot looks at the current position of the board. 2. The material count for each player is added up. 3. The activity of the pieces for each player is given a numerical score. 4. The king safety of each player is analysed and given a numerical score. 5. The pawn structure of each player is evaluated and given a numerical score. 6. For each score, the score is added to the evaluation number for white and subtracted for black. 7. The final evaluation score is arrived at through the calculations of each separate score.		
Postconditions	The position is given an evaluation number.		
Alternate Flows	**none**		

8.2. Calculate Move Use Case

Use Case Name	*Calculate Move*	***UniqueID***	*UC007*
Description	The bot evaluates the position that possible, legal moves and the responses to those moves can lead to. Any branch of opponent responses that results in a worse position is pruned and not analysed deeper. After either analysing three moves ahead or the time allowed per move runs out, the best move found is selected.		
Actors	Herm0ni Chess Bot		
Precondition	• Lichess.org is operational. • A game is in progress with the Herm0ni bot.		
Trigger	The engine begins calculating at the start of the game and continues throughout until the game ends.		

Main Path	8. The bot looks at the current position of the board. 9. The bot considers a possible, legal next move. 10. The possible, legal responses to that move by the opponent are considered. 11. If a move is found that results in a worse position for the bot, the branch is pruned, and the move is not considered. 12. Steps 2 to 4 are repeated for all possible, legal moves that the bot can make. 13. This is repeated to a depth of three moves, or until the time allocated per move is reached. 14. The best move found is selected to be made.
Postconditions	A move is selected, and the selection is communicated to the site.
Alternate Flows	**4a No Good Move Found** 1. The bot picks the move that results in the lowest negative impact on the evaluation. **4b Forced Move** 1. The bot is forced to make a certain move due to check(s) or pins.

8.3. Update Position Use Case

Use Case Name	*Update Position*	***UniqueID***	*UC010*
Description	A move is made by either the user or the bot. The bitboard are compared to see what piece moved and to what square it moved to. The bitboards are updated to represent the new position. Whether either player’s king is in check is checked.		
Actors	Herm0ni Chess Bot		
Precondition	• Lichess.org is operational. • A game is in progress with the Herm0ni bot.		
Trigger	Either the bot or its opponent makes a move.		
Main Path	1. The bot selects its next move. 2. The bitboards for the piece that is moved it updated to reflect its new position. 3. The bitboard is compared with other piece bitboards to determine if a capture has taken place.		

	4. An XOR operation is performed on the bitboards to determine the new position. 5. Whether either king is in check is checked.
Postconditions	The new position of all pieces and whether a king is in check is determined.
Alternate Flows	**1a Opponent Makes the Move** 1. Opponent's move is received in JSON from the site using UCI. 2. Message is converted from JSON. 3. Bitboards are updated to reflect the new position of all pieces.

References

Bijl, P. and Tiet, A.P., 2021*. Exploring modern chess engine architectures.* Victoria University, Melbourne.

Chess.com, n.d. *Time Controls.* [Online]
Available at: https://www.chess.com/terms/chess-time-controls
[Accessed 23 October 2024].

Chessprogramming, 2024. *Endgame Tablebases.* [Online]
Available at: https://www.chessprogramming.org/Endgame_Tablebases
[Accessed 23 October 2024].

Chessprogramming, 2024. *OpeningBook.* [Online]
Available at: https://www.chessprogramming.org/Opening_Book
[Accessed 23 October 2024].

Chessprogramming, 2024. *UCI.* [Online]
Available at: https://www.chessprogramming.org/UCI
[Accessed 22 October 2024].

Krishnamurthy, B., 2022. *The Evolution of Chess AI.* [Online]
Available at: https://builtin.com/artificial-intelligence/chess-ai
[Accessed 16 11 2024].

Lichess, n.d. *About lichess.org.* [Online]
Available at: https://lichess.org/about
[Accessed 20 October 2024].

Vecteezy, n.d. *Vecteezy.* [Online]
Available at: https://www.vecteezy.com/free-vector/chess-silhouette
[Accessed 26 November 2024].