Software Engineering Stage 6

FuryEngine Report

Atharva Malik

GitHub: chess-engine

Date of Submission: 4 th July 2025, 9:00 AM

1. Identifying and Defining....4
1.1 Problem Statement.... 4
1.2 Project Purpose and Boundaries....4
1.3 Stakeholder Requirements....4
1.4 Functional Requirements.... 5
CLI Version.... 5
Website Version.... 5
1.5 Non-Functional Requirements....6
CLI Version.... 6
Website Version.... 6
1.6 Constraints.... 6
Time.... 6
Budget.... 6
Technical Limitations....7
2. Research and Planning.... 7
2.1 Development Methodology....7
2.2 Tools and Technology.... 7
2.3 Gantt Chart.... 8
2.4 Communication Plan....9
3. System Design.... 10
3.1 Context Diagram....10
3.2 Data Flow Diagrams.... 10
Level 1.... 10
Level 2 - Set Game Position....11
Level 2 - Find Best Move.... 11
3.3 Structure Chart.... 12
3.4 IPO Chart....12
3.5 Data Dictionary.... 13
3.6 UML Class Diagram.... 14
4. Producing and Implementing.... 14
4.1 Development Process.... 14
4.2 Key Features Developed....15
4.3 Screenshots of Interface....15
CLI Version....15
4.4 Sprint Summary.... 16
5. Testing and Evaluation.... 16
5.1 Testing Methods Used.... 16
5.2 Test Cases and Results.... 17
5.3 Evaluation Against Requirements.... 18

5.4 Improvements and Future Work..18
6. Feedback and Reflection...18
6.1 Summary of Client Feedback..18
Siddhant Bhale (Year 9)...18
Natalie Biyo (Year 11)...19
Akshat Khurana (Year 11)..19
Daniel Hay (Year 11)..19
Warren Mao (Year 11)..19
6.2 Personal Reflection..20
7. Appendices..20
7.1 Full Diagrams..20
7.2 Full Test Logs..20
7.3 Exemplar Code Snippets..20
Main Function..20
Bot Class..21
Negamax Algorithm...22
Minimax Algorithm...23
Parallel Search...25
7.4 Secure and Safe Software Implementation..27
7.5 Credits...27

1. Identifying and Defining

1.1 Problem Statement

Modern chess engines have achieved a level of play that consistently surpasses human capabilities, making them unbeatable. Their rapid, optimal move generation, while efficient, results in an 'emotionless' game experience that is demoralising for human players, diminishing their enjoyment and desire to continue playing. Therefore, this project sets out to make a chess engine that is quite strong, so as not to get crushed, but also beatable. An engine which seems to think like a human, taking longer to ponder over more complex positions. In summary, this engine seeks to overcome the current issue of dominant, unengaging AI and foster a more enjoyable and rewarding human-computer chess experience.

1.2 Project Purpose and Boundaries

As explained in the problem statement, the project is trying to make chess more fun for human players when playing against their computer counterparts. The project is setting out to make a chess engine rated about 2700 with various features that are seen in common engines like:

- Support for Universal Chess Interface CLI
- Forsyth-Edwards Notation Support
- Error and Action Logging
- Vast Opening Book
- Online Hosting

1.3 Stakeholder Requirements

The stakeholders are people like me and Adi who love playing chess but do not want to play against other engines as they are either too easy to beat or too "emotionless." The stakeholders require a chess engine that is not only challenging to play but also engaging to play against.

1.4 Functional Requirements

CLI Version

The following functional requirements apply to the CLI version of the chess engine:

Definition	Analysis
Best move ● Time Left ● Move Time	Find the best move with Time Left and Move Time constraints
Forsyth–Edwards Notation support	Support for the FEN notation style of board representation
Raw Eval	Enable the return of a Raw Evaluation of a position.
Opening Book	A ‘book’ with the best opening moves

Website Version

The following functional requirements apply to the website version of the chess engine:

Definition	Analysis
Best move ● Time left ● Move time	Find the best move with Time Left and Move Time constraints
Forsyth–Edwards Notation support	Support for the FEN notation style of board representation
Opening Book	A ‘book’ with the best opening moves
Online Hosting	An interface for hosting the chess engine online on Lichess

1.5 Non-Functional Requirements

CLI Version

The following non-functional requirements apply to the CLI version of the chess engine:

Definition	Analysis
Universal Chess Interface CLI	This is the part which allows users to communicate with the engine.
Reasonable Logging	A log file written in simple English to help people understand any problems they encounter.

Website Version

The following non-functional requirements apply to the website version of the chess engine:

Definition	Analysis
A GUI	An intuitive GUI that enables people to easily interact with the engine.
Reasonable Logging	A log file written in simple English to help people understand any problems they encounter.

1.6 Constraints

Time

The time constraints for this project are outlined in more detail in the Appendices along with the full Gantt Chart. However, the main time constraint is the due date, which is the 4th of July.

Budget

The budget limit on this project depends mainly on hosting the website version as the costs for it can add up fast. Therefore, I want to make a system that allows me to host it for as close to free as possible.

Technical Limitations

This is the main constraint on this project. This project is an entirely new avenue of exploration for me, therefore, due to the lack of experience with lower-level programming, efficient programming, advanced OOPs principles, and CLI interfaces, the end result could be slightly different to the envisioned product. However, due to its inherent complexity, this project will be a great learning experience for me, therefore the team is quite keen to tackle it.

2. Research and Planning

2.1 Development Methodology

I have decided to use an agile model, along with the Object Oriented Paradigm, due to its intrinsic flexibility, continuous improvement, and faster time to market. Each sprint the team would try to implement a new feature:

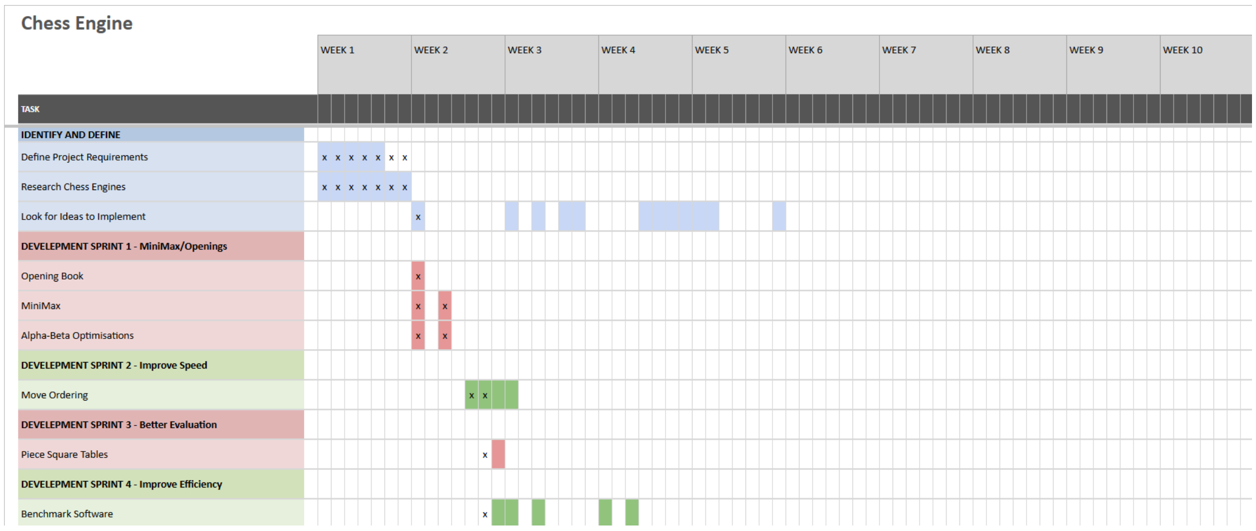
- Plan: Quickly plan on what features I want to implement. Plan out the test cases for the features.
- Design: Conduct online searches and brainstorm ways to implement planned feature
- Develop: Make the solution by writing out the code.
- Test: Test the engine locally with famous test positions and planned test cases.
- Deploy: Deploy the engine on Lichess to let it compete against other bots.
- Review: Review how well/poorly the engine fared.

2.2 Tools and Technology

For the development of this project, various tools and technologies were used, some of which include:

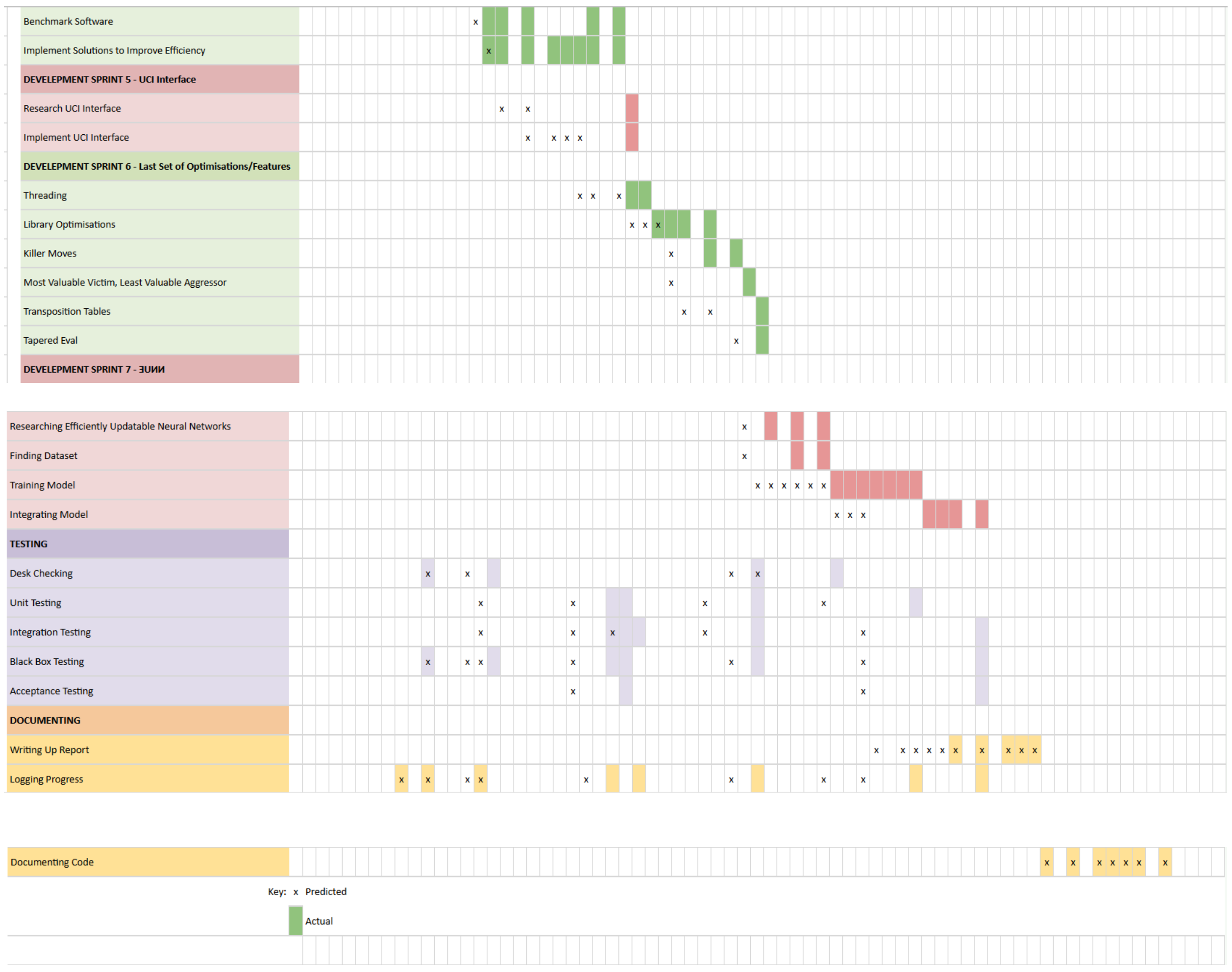
- Python 3.12
- Python-Chess
- C++ 17
- Chess-library
- NNUE-trainer
- Chess Programming Wiki
- G++

2.3 Gantt Chart



Chess Engine

	WEEK 1							WEEK 2							WEEK 3							WEEK 4							WEEK 5							WEEK 6							WEEK 7							WEEK 8							WEEK 9							WEEK 10						
TASK																																																																						
IDENTIFY AND DEFINE																																																																						
Define Project Requirements	x	x	x	x	x	x	x																																																															
Research Chess Engines	x	x	x	x	x	x	x																																																															
Look for Ideas to Implement								x							🟦		🟦		🟦	🟦					🟦	🟦	🟦	🟦	🟦	🟦					🟦																																			
DEVELEPMENT SPRINT 1 - MiniMax/Openings																																																																						
Opening Book								x																																																														
MiniMax								x		x																																																												
Alpha-Beta Optimisations								x		x																																																												
DEVELEPMENT SPRINT 2 - Improve Speed																																																																						
Move Ordering												x	x	🟩	🟩																																																								
DEVELEPMENT SPRINT 3 - Better Evaluation																																																																						
Piece Square Tables													x	🟥																																																								
DEVELEPMENT SPRINT 4 - Improve Efficiency																																																																						
Benchmark Software													x	🟩	🟩		🟩					🟩		🟩																																															



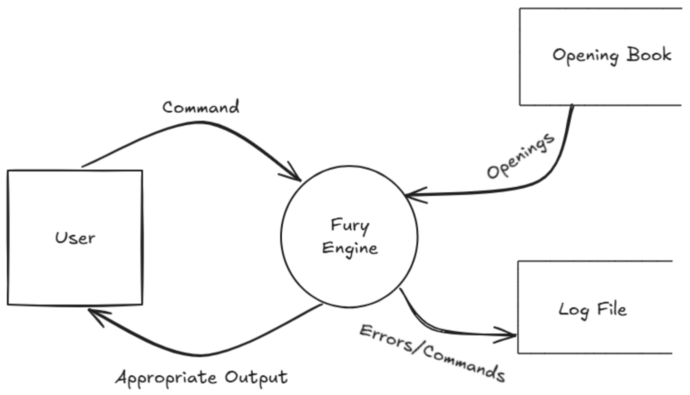
*Note: The full sized Gantt Chart is provided in the Appendices.

2.4 Communication Plan

Good client-developer communication is imperative for the development of a good product. Therefore, I will be engaging the client in face-to-face meetings once a week. Along with this, there will be multiple conversations over communication platforms such as WhatsApp and iMessages. There will be one final face-to-face meeting with the client once the product has been fully developed to perform acceptance testing and get their feedback.

3. System Design

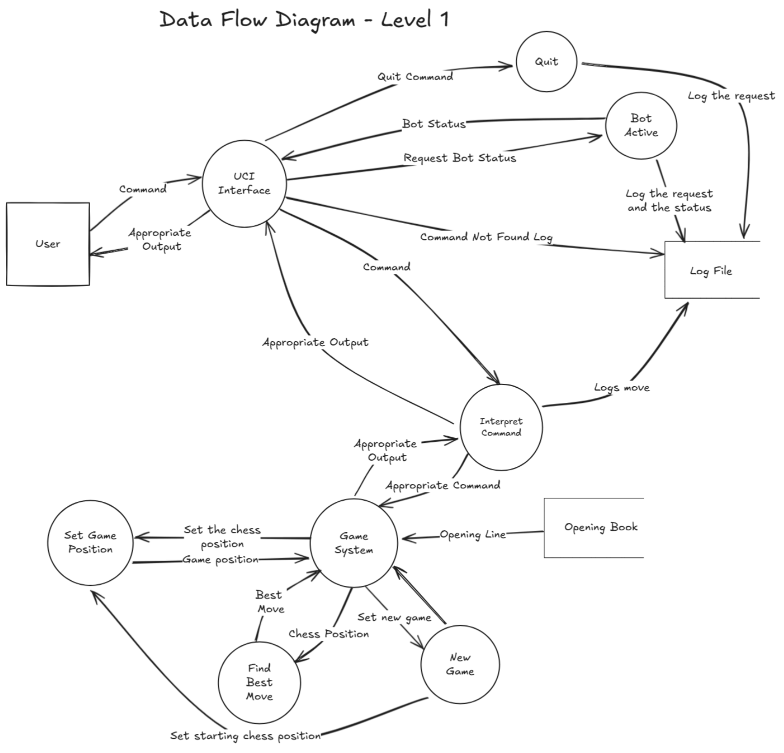
3.1 Context Diagram



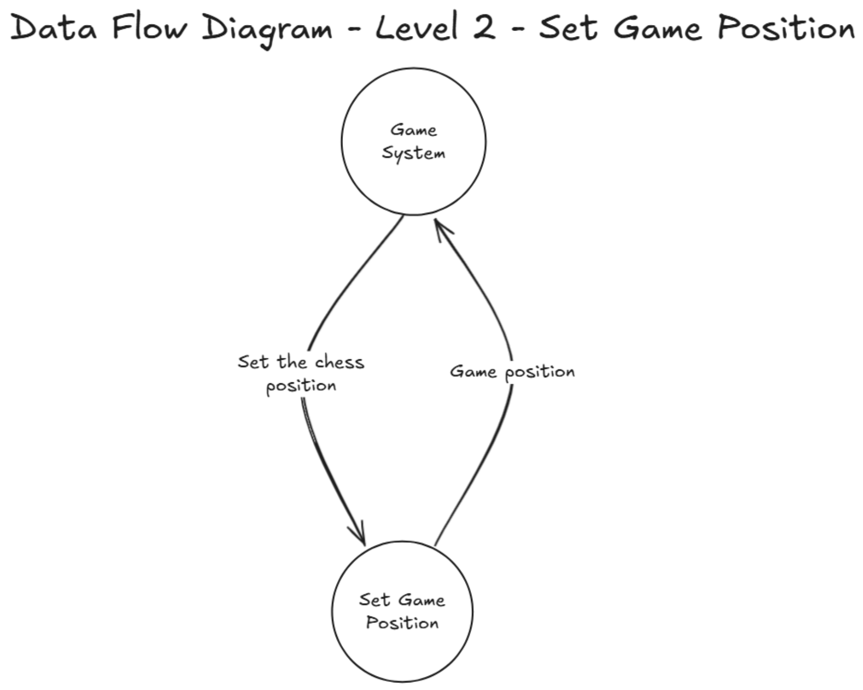
Data Flow Diagram - Level 0 - Context Diagram

3.2 Data Flow Diagrams

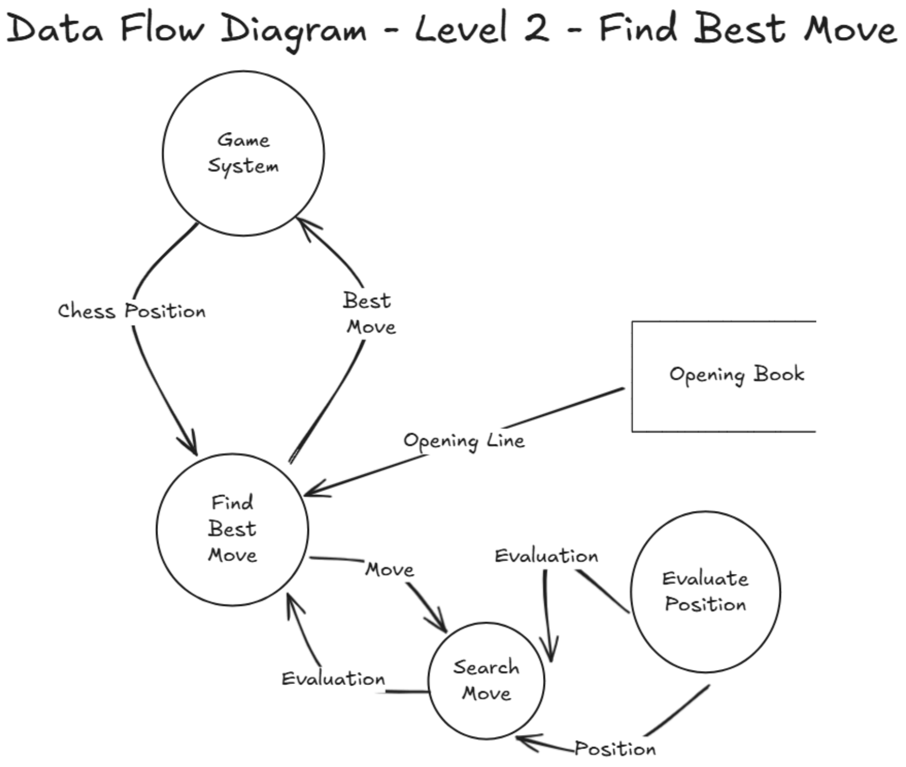
Level 1



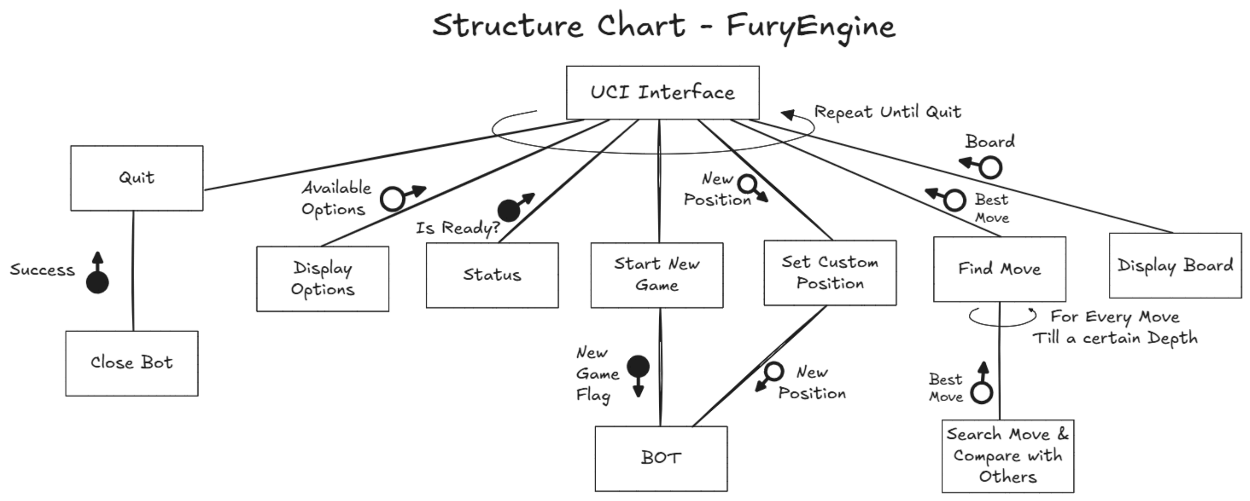
Level 2 - Set Game Position



Level 2 - Find Best Move



3.3 Structure Chart



*Note: The full sized versions of all of these diagrams are provided in the Appendices.

3.4 IPO Chart

Module	Input	Process	Output
Evaluation (static)	• Board State	• Add up all of the material points for white, taking into consideration the piece square values. • Add up all of the material points for black, taking into consideration the piece square values. • Subtract the sum of black material from the sum of white material.	Floating point representation of the position
Inline Search	• Board State • Side to Move • Depth	• Search through all moves in the position. • Make each move. • Repeat these steps the specified number of times to search for the specified depth.	The Best Move
UCI Interface	• Command	• Break down the command, comparing	• “Command not found”

		the first word with the list of available commands. • If no comparison found, return “Command not found” • Otherwise, execute appropriate functions.	OR • Execution of appropriate functions.
Parallel Search	• Board State • Side to Move • Depth	• For each available move, create a separate thread. • For each move, perform inline search in its thread at the depth of specified depth minus one.	The Best Move

3.5 Data Dictionary

This is the data dictionary for the main Bot class.

Field Name	Data Type	Size	Description	Example Value	Constraints/Validation Rules

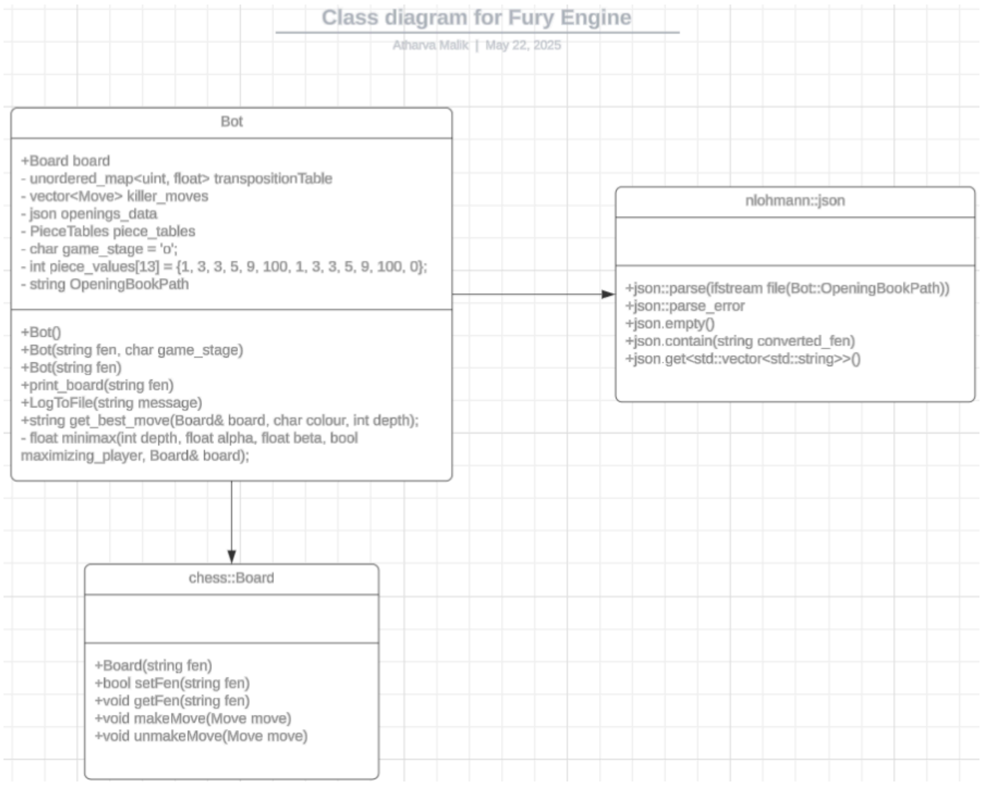
transpositionTable	unordered_map<uint, float>	12 bytes per entry	It is a table that stores evaluations of zobrist hashes	Sample entry:
				{
				<1431999469626031762, 6.52>
				<6454364923581549305, -0.91>
				}
					The keys must be unsigned 64 bit integers and the values must be the evaluation of the hashed position

openings_data	JSON	2.02 MB	A json dictionary storing a starting position and good moves to play from that position	Sample entry:
				{
				"r2qkb1r/pp3ppp/2n1pn2/3p4/3P2b1/2PB1N1P/PP3PP1/RNBQ1RK1 b kq -": [
				"g4h5"
				]
				}
					The key must be a FEN (Forsyth–Edwards Notation) string and the value must be a list of good moves.

game_stage	char	8 bits	A single character representation of the game stage	‘o’	Must be one of: ‘o’, ‘m’, ‘e’ Representing opening, middle game, and end game respectively

piece_values	Array of integers	52 bytes	An array representing the values of the pieces with the index being the piece number	{1, 3, 3, 5, 9, 100, 1, 3, 3, 5, 9, 100, 0}	Must be the values representing the material values of the black and white pieces

3.6 UML Class Diagram


Class diagram for Fury Engine
Atharva Malik | May 22, 2025
Bot
+Board board
- unordered_map<uint, float> transpositionTable
- vector<Move> killer_moves
- json openings_data
- PieceTables piece_tables
- char game_stage = 'o';
- int piece_values[13] = {1, 3, 3, 5, 9, 100, 1, 3, 3, 5, 9, 100, 0};
- string OpeningBookPath
+Bot()
+Bot(string fen, char game_stage)
+Bot(string fen)
+print_board(string fen)
+LogToFile(string message)
+string get_best_move(Board& board, char colour, int depth);
- float minimax(int depth, float alpha, float beta, bool maximizing_player, Board& board);
nlohmann::json
+json::parse(ifstream file(Bot::OpeningBookPath))
+json::parse_error
+json.empty()
+json.contain(string converted_fen)
+json.get<std::vector<std::string>>()
chess::Board
+Board(string fen)
+bool setFen(string fen)
+void getFen(string fen)
+void makeMove(Move move)
+void unmakeMove(Move move)

*Note: The full sized version of this diagram is provided in the Appendices.

4. Producing and Implementing

4.1 Development Process

To build this project, as mentioned above, I decided to settle on the agile model of development to allow for rapid prototyping and high flexibility. To start with, a basic engine was made which used the MiniMax Alpha-Beta algorithm to provide reasonable moves. Following this, tests were performed on the engine, and at each subsequent sprint, the identified bugs were fixed and a new feature was added to improve the bot.

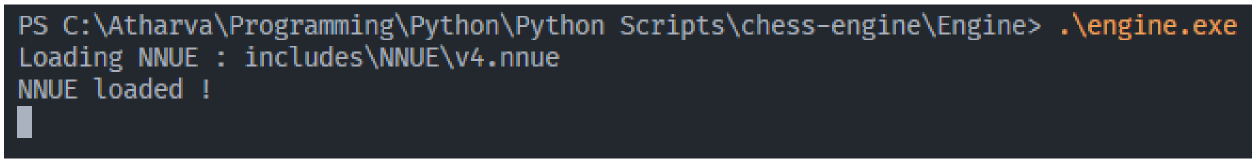
4.2 Key Features Developed

Feature	Description	Justification
UCI Interface	The UCI Interface is how the end user communicates with the engine.	This is an essential feature as it provides an interface of communication for the CLI version of the engine. It allows the users to directly probe the engine and also allows the engine to be compatible with various GUIs due to the popularity of the interface.
Threading	Every move of a position is analysed in a separate thread.	This allows the engine to parallelly process all of the possible moves, thus speeding it up enormously.
Negamax	An algorithm used to analyse a sequence of positions up to a given depth.	This is an essential feature as it allows the engine to assign a mathematical value to separate moves and hence compare them. Although I had started with MiniMax Alpha Beta, Negamax soon became the favoured choice due to its brevity and simplicity.
ƎUИИ	A custom neural network specialising in evaluating positions.	The NNUE (often called ƎUИИ) is a custom built neural network designed to be update and produce results efficiently. The benefits of using a neural network over a hand-crafted evaluation function are immense as neural networks better understand the underlying complexities in chess positions, leading to more accurate evaluations, and due to efficient updates, faster evaluations. This allows the engine to play at a better level.

4.3 Screenshots of Interface

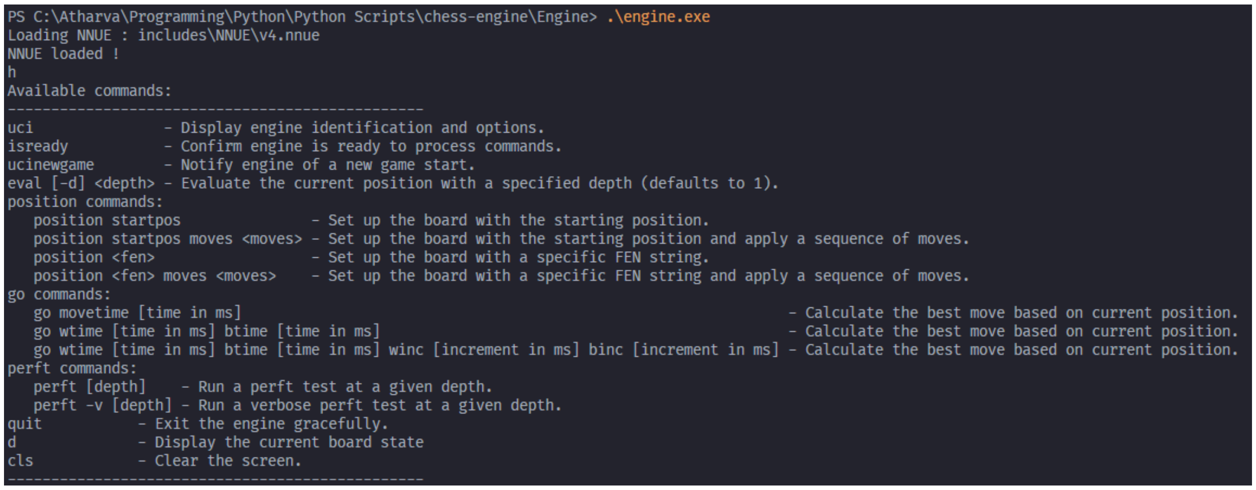
CLI Version

Main Menu:



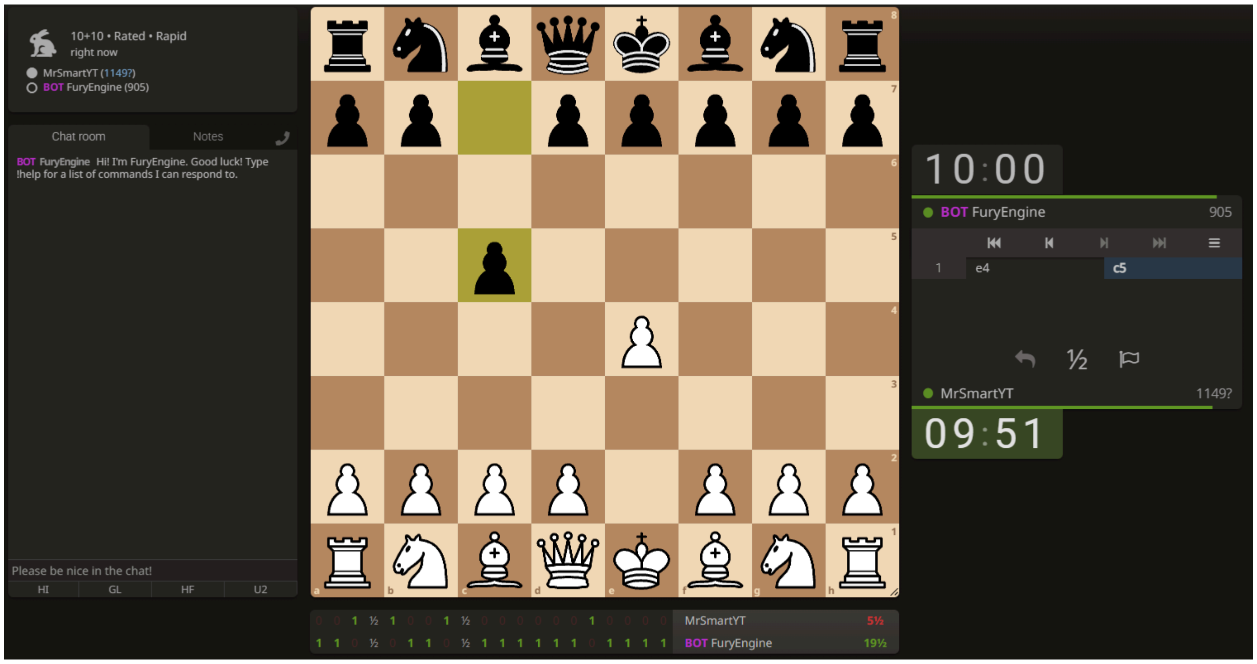
```
PS C:\Atharva\Programming\Python\Python Scripts\chess-engine\Engine> .\engine.exe
Loading NNUE : includes\NNUE\v4.nnue
NNUE loaded !
```

This allows the user to select the various options and interact with the CLI as portrayed above and below:



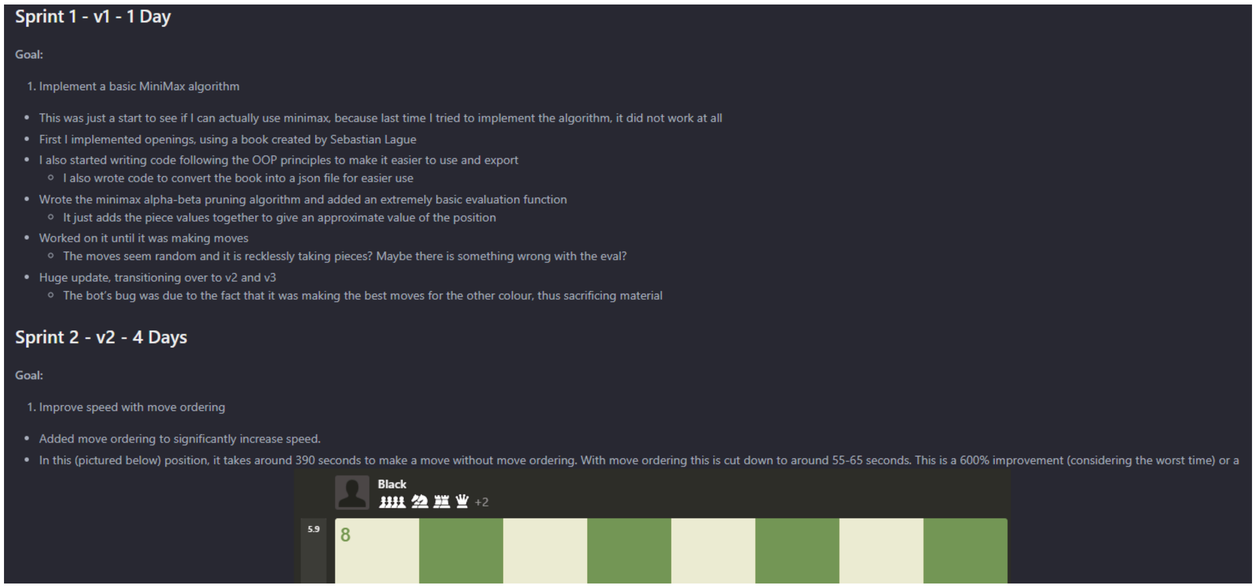
```
PS C:\Atharva\Programming\Python\Python Scripts\chess-engine\Engine> .\engine.exe
Loading NNUE : includes\NNUE\v4.nnue
NNUE loaded !
h
Available commands:
------------------------------------------------
uci               - Display engine identification and options.
isready           - Confirm engine is ready to process commands.
ucinewgame        - Notify engine of a new game start.
eval [-d] <depth> - Evaluate the current position with a specified depth (defaults to 1).
position commands:
   position startpos               - Set up the board with the starting position.
   position startpos moves <moves> - Set up the board with the starting position and apply a sequence of moves.
   position <fen>                  - Set up the board with a specific FEN string.
   position <fen> moves <moves>    - Set up the board with a specific FEN string and apply a sequence of moves.
go commands:
   go movetime [time in ms]                                                         - Calculate the best move based on current position.
   go wtime [time in ms] btime [time in ms]                                         - Calculate the best move based on current position.
   go wtime [time in ms] btime [time in ms] winc [increment in ms] binc [increment in ms] - Calculate the best move based on current position.
perft commands:
   perft [depth]    - Run a perft test at a given depth.
   perft -v [depth] - Run a verbose perft test at a given depth.
quit           - Exit the engine gracefully.
d              - Display the current board state
cls            - Clear the screen.
------------------------------------------------
```

Website Version



BOT FuryEngine Hi! I'm FuryEngine. Good luck! Type !help for a list of commands I can respond to.

4.4 Sprint Summary



Sprint 1 - v1 - 1 Day

Goal:

1. Implement a basic MiniMax algorithm

- This was just a start to see if I can actually use minimax, because last time I tried to implement the algorithm, it did not work at all
- First I implemented openings, using a book created by Sebastian Lague
- I also started writing code following the OOP principles to make it easier to use and export
  - I also wrote code to convert the book into a json file for easier use
- Wrote the minimax alpha-beta pruning algorithm and added an extremely basic evaluation function
  - It just adds the piece values together to give an approximate value of the position
- Worked on it until it was making moves
  - The moves seem random and it is recklessly taking pieces? Maybe there is something wrong with the eval?
- Huge update, transitioning over to v2 and v3
  - The bot's bug was due to the fact that it was making the best moves for the other colour, thus sacrificing material

Sprint 2 - v2 - 4 Days

Goal:

1. Improve speed with move ordering

- Added move ordering to significantly increase speed.
- In this (pictured below) position, it takes around 390 seconds to make a move without move ordering. With move ordering this is cut down to around 55-65 seconds. This is a 600% improvement (considering the worst time) or a

*Note: The full sized Sprint Logs are provided in the GitHub.

5. Testing and Evaluation

5.1 Testing Methods Used

To ensure a high build quality, various testing methods were used throughout the project's development. Below is a table of the methods used against the reasons they were used for:

Method	Use/Justification
Desk Checking	Casual checks, along with help from VS Code's linter, were performed after implementation of every feature to ensure that there were no obvious syntax/logic errors.
Unit Testing	Since most of the code was developed as individual units due to the adherence to the OOPs paradigm, unit tests were essential and conducted thoroughly to ensure that all individual units were fully error-free.
Integration Testing	Again, since most of the code was developed as individual units, integration testing was vital to ensure that all individual units worked together flawlessly
White-box Testing	When persistent errors were encountered, white-box testing was performed, with the help of a debugger, to run through all the code paths to identify the bug.

Alpha Testing	Before the GUI was developed, Alpha Testing was conducted by me to ensure the back-end side was working faultlessly.
Systems Testing	Before deploying the finished system, rigorous testing was performed on the GUI to ensure there were no persistent issues. To perform systems tests, the engine was pitted against various other engines available online and me.
Acceptance Testing	After the finished system was pieced together, it was presented to the client for acceptance testing to ensure that it meets their needs.

5.2 Test Cases and Results

Description	Expected Result	Actual Result	Pass/Fail
Enter invalid position	Error Message	Error Message	Pass
Enter valid position	Silence	Silence	Pass
Mate in 6 puzzle	Best Move	Sub-optimal Move	Fail
Kiwipete Perft at Depth 6	8031647685	8031647685	Pass
Enter unsupported time control	Error Message	Error Message	Pass
Find Best Move	Move within top 3 of Stockfish's suggestions	Top Stockfish Move	Pass
Speed Perft Test	3,000,000+ nps	4,253,087 nps	Pass

5.3 Evaluation Against Requirements

The solution hits all of the project requirements, as testified by the client: "This is a great bot. Like not only by skills but overall feeling. Some bots rapid fire moves but it takes time [which] makes it seem like you're playing a human." Overall, the solution hits all of the targets it set out to hit, moving like a human, playing strong chess, and being GUI compatible.

5.4 Improvements and Future Work

If I had more time, I would love to improve the NNUE model. Currently it was trained on only two billion games over 4 days, but I believe that training it for longer over more games would allow for more accurate positional understanding, especially in the earlier stages of the game. I also want to work on more heuristics like null-move pruning to make the engine even more efficient. Last but not the least, I would also have liked to set up a dedicated website for the documentation of the chess engine. Overall, however, I am quite pleased with the solution I was able to deliver in my limited time.

6. Feedback and Reflection

6.1 Summary of Client Feedback

All of the clientele feel that the engine exceeds their expectations, and following are some of their comments:

Siddhant Bhale (Year 9)

“The implementation of the engine is great. It is truly fun to play against it. The GUI is easy to use and is appropriately selected. Maybe there could be an option to select difficulty, but other than that, this engine exceeds all of my expectations.”

Natalie Biyo (Year 11)

"I'm genuinely impressed with this chess engine. It strikes that perfect balance of being challenging enough to keep me engaged, but also providing that sense of being beatable, which most other engines don't offer. The move choices feel more thoughtful, and the 'pondering' over complex positions really adds to the immersion, making it feel less like playing against an algorithm and more like a real opponent. It's truly a unique experience."

Akshat Khurana (Year 11)

"This engine has completely revitalised my interest in playing chess against AI, and by extension, Chess. The previous frustrations have vanished. Instead, I find myself learning and improving because the engine makes relatable 'mistakes' or takes lines that feel more human. The user interface is also very intuitive and clean, allowing me to focus entirely on the game. Great work!"

Daniel Hay (Year 11)

"From a strategic perspective, this engine is fascinating. It's strong enough to push me, but not so strong that I feel helpless. What stands out most is the pacing; the varied thinking times make the game flow much more naturally. It avoids that robotic, instantaneous response that removes the fun out of playing against computers. I can definitely see myself spending a lot more time playing against this."

Warren Mao (Year 11)

"My experience with this engine has been mostly positive. It delivers exactly what was promised: a powerful, yet beatable, chess opponent. The 'human-like' decision-making is clearly evident and makes for a far more enjoyable and less demoralising game. The overall design and responsiveness of the application are excellent. This is a significant improvement and what feels like a step in the right direction."

6.2 Personal Reflection

I was extremely pleased with my learning. I was overjoyed at having learnt a lot about a low-level programming language like C++, along with exploring more advanced use cases of the Object Oriented Paradigm like constructor overloading which proved invaluable. I also learnt a lot about compiler flags and various other techniques to squeeze out extra performance from their code. This hands-on experience has been incredibly rewarding and will undoubtedly benefit my future development endeavors.

7. Appendices

7.1 Full Diagrams

All of the diagrams can be explored at their original resolution here.

7.2 Full Test Logs

The full test logs, along with the test and benchmarking conducted are thoroughly documented in the sprint logs available on the project's GitHub repository

7.3 Exemplar Code Snippets

Main Function

```
int main () {

    init_nnue("includes\\NNUE\\v4.nnue");

    UciPlayer player;

    std::string command = "";

    while (lower(command) != "quit" && lower(command) != "exit" && lower(command)
!= "q")

    {

        std::getline(std::cin, command);

        ReceiveCommand(command, player);

    }

    Bot::LogToFile("CLOSING UCI bot");

    return 0;

}
```

Bot Class

```
class Bot{

    public:

        Bot();

        Bot(std::string fen, char game_stage);

        Bot(std::string fen);
        Board board;

        static void print_board(Board board);

        std::string get_best_move(Board& board, char colour, int depth);

        static void LogToFile(const std::string& message);

        float stat_eval(Board board, int depth);

    private:

        json openings_data;

        PieceTables piece_tables;

        char game_stage = 'o';

        int piece_values[13] = {1, 3, 3, 5, 9, 100, 1, 3, 3, 5, 9, 100, 0};

        std::string opening_move(const std::string& fen, char colour);

        std::string middle_game_move(int depth, Board& board, char colour);

        std::string middle_game_x_thread(int depth, Board& board, char colour);

        std::string end_game_move(int depth, Board& board, char colour);


        float minimax(int depth, float alpha, float beta, bool maximizing_player,
Board& board);

        float negamax(int depth, float alpha, float beta, Board& board);

        float eval_mid(Board board);

        float eval_end(Board board);

        std::string convert_fen(std::string fen);

        std::string OpeningBookPath = "includes\\OpeningBook\\book.json";

        int determineDepth(const Board& board);

        int get_random_index(const std::vector<std::string>& vec);
        float search_move(Move move, Board board, int depth, int colour);

        float calculate_phase(Board board);

        bool isCheck(Move move, Board& board);

        bool load_openings_data();

        void order_moves(Movelist& moves, Board& board);

};
```

Negamax Algorithm

```
float Bot::negamax(int depth, float alpha, float beta, Board& board){

    std::pair<GameResultReason, GameResult> isGameOver = board.isGameOver();

    if (isGameOver.first == GameResultReason::CHECKMATE){

        return -9999.0f * depth;

    } else if (!(isGameOver.first == GameResultReason::NONE)){

        return 0.0f;

    }

    else if (depth == 0) return evaluate_fen_nnue(board.getFen());


    Move move = Move();

    Movelist moves = Movelist();

    movegen::legalmoves(moves, board);

    float best_eval = -999999999999.9f;

    float evaluation = 0;

    order_moves(moves, board);

    for (auto move : moves) {
        // move = moves[i];

        board.makeMove(move);

        evaluation = -this->negamax(depth - 1, -beta, -alpha, board);

        board.unmakeMove(move);

        best_eval = std::max(best_eval, evaluation);

        alpha = std::max(alpha, evaluation);

        if (beta <= alpha) break;  // Beta cutoff

    }

    return best_eval;

}
```

Minimax Algorithm

```
float Bot::minimax(int depth, float alpha, float beta, bool maximizing_player,
Board& board){

    std::pair<GameResultReason, GameResult> isGameOver = board.isGameOver();

    if (isGameOver.first == GameResultReason::CHECKMATE){

        if (board.sideToMove() == Color::WHITE) return -9999.0f;

        else return 9999.0f;

    } else if (!(isGameOver.first == GameResultReason::NONE)){

        return 0.0f;

    }

    else if (depth == 0) return this->eval_mid(board);


    Move move = Move();

    Movelist moves = Movelist();
```

```
movegen::legalmoves(moves, board);

if (maximizing_player){

    float maxEval = -9999.0f;

    float evaluation = 0;

    order_moves(moves, board);

    for (int i = 0; i < moves.size(); i++){

        move = moves[i];

        board.makeMove(move);

        evaluation = this->minimax(depth - 1, alpha, beta, false, board);

        board.unmakeMove(move);

        maxEval = std::max(maxEval, evaluation);

        alpha = std::max(alpha, evaluation);

        if (beta <= alpha) break;  // Beta cutoff

    }

    return maxEval;

}

else{

    float minEval = 9999.0f;

    float evaluation = 0;

    order_moves(moves, board);

    for (int i = 0; i < moves.size(); i++){

        move = moves[i];

        board.makeMove(move);

        evaluation = this->minimax(depth - 1, alpha, beta, true, board);
```

```
            board.unmakeMove(move);

            minEval = std::min(minEval, evaluation);

            beta = std::min(beta, evaluation);

            if (beta <= alpha) break;  // Beta cutoff

        }

        return minEval;

    }

}
```

Parallel Search

```
std::string Bot::middle_game_x_thread(int depth, Board& board, char colour){

    Movelist moves = Movelist();

    movegen::legalmoves(moves, board);

    order_moves(moves, board);


    float evaluation;

    Move move = Move();

    const int num_iterations = moves.size();

    std::vector<std::thread> threads;

    std::vector<std::pair<double, std::string>> data(num_iterations);

    std::mutex results_mutex;

    for (int i = 0; i < num_iterations; ++i) {

        threads.emplace_back(

            [&, i]() {
                float result = this->search_move(moves[i], board, depth, 1);

                std::lock_guard<std::mutex> guard(results_mutex);

                data[i].first = result;

                data[i].second = uci::moveToUci(moves[i]);

            }

        );

    }

    for (auto& thread : threads) {

        if (thread.joinable())

            thread.join();

    }


    std::sort(data.begin(), data.end(), [](auto &a, auto &b) {

        return a.first > b.first;

    });

    return data[0].second;

}
```

7.4 Secure and Safe Software Implementation

While the actual engine doesn't actively use any safety practices like encryption, it uses various other techniques. For example, due to the presence of threads, all shared variables need to be thread safe. Moreover, the communication between the GUI and the CLI is done only after an encrypted tunnel has been established between the frontend (Lichess) and the backend (the chess engine). Along with everything mentioned above, Lichess already uses various techniques like encryption, hashing, and 2 factor authentication to protect user accounts and passwords. Therefore, I believe that my implementation of the solution is the safest, most secure implementation possible.

7.5 Credits

Finally, I want to thank the following people/organisations for their contributions to this project:

- Niels Lohmann - For providing the JSON library for C++ which allowed the creation of the opening book to be as simple as it was.
- Max (Disservin) - For providing the blazing fast library that is at the backbone of the chess engine, without which the engine might not be performing as fast as it is.
- Sebastian Lague - For providing the idea and motivation for this project
- Chess Programming - For teaching me how chess engines work in the backend.
- Stockfish Evaluation Guide - For providing a simple, yet powerful, graphical tool to understand how NNUEs work