Stockfish GUI

Software Design Document

Group 2

Zamil Bahri, Yousef El-Qawasmi, Stephen Morris, Phoebe Schulman

Date: 11/18/2021

Table of Contents

1.0 INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of our interactive Stockfish GUI program. This program serves as an analysis tool for a chess board rather than a full game. It's designed to find the legal moves of each chess piece and use the engine Stockfish to find the best moves.

2.0 SYSTEM OVERVIEW/FUNCTIONAL DESCRIPTION

2.1 Context

A FEN ("Forsyth–Edwards Notation") is a string that represents the current state of a chessboard. Each piece is represented by a letter, empty spaces are represented by a number, capital letters are white pieces, and lowercase letters are black pieces. It starts from the top left corner of the board, moves to the right, and then down each row. For example:
N7/P3pk1p/3p2p1/r4p2/8/4b2B/4P1KP/1R6 w - - 0 1
Means that a white Knight (N) is in the top left corner, followed by 7 empty spaces. On the next line below it (/) there's a white Pawn (P), 3 spaces, black Pawn (p), etc.

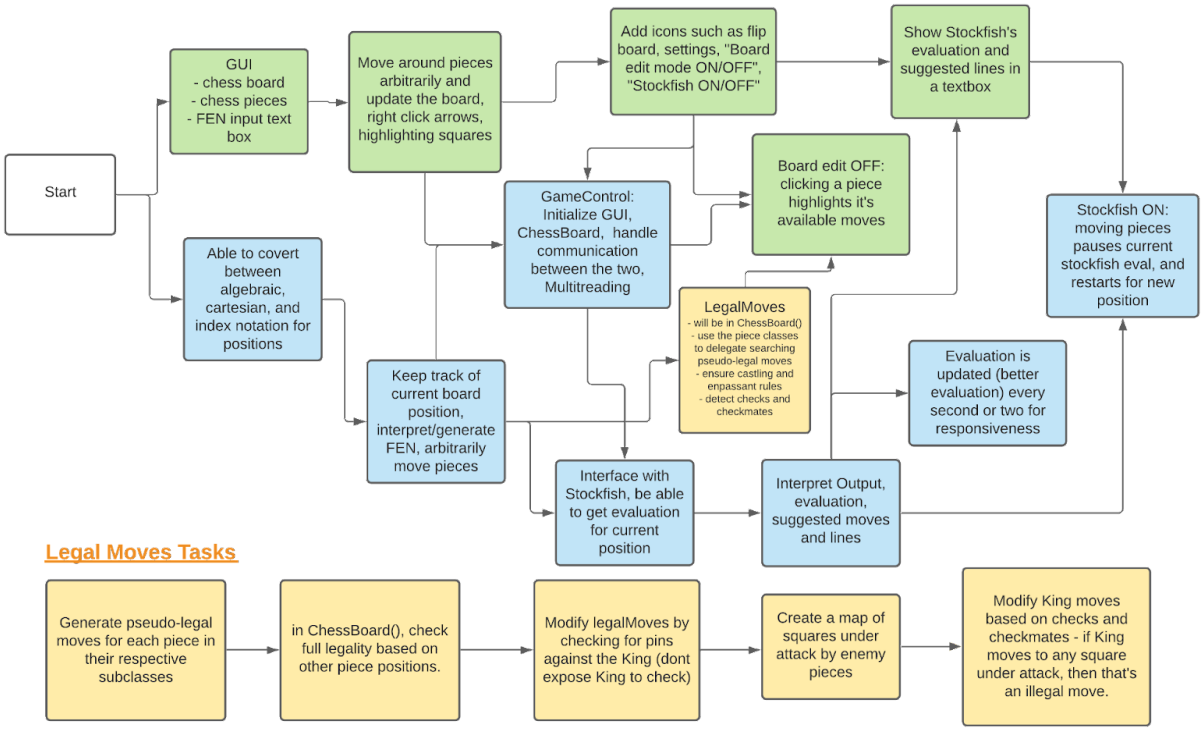
Stockfish is an open source chess engine developed in C++. It can be used to analyze chess positions based on a particular FEN input. It outputs a numerical evaluation of the position, where a positive number represents an advantage for white, and a negative number represents an advantage for black. It also returns what it thinks the best move is, as well as the best continuation if that particular move is played.

2.2 Developmental Goals and Prioritization

Design goals (sorted by highest to lowest priority):

1. Build the chess board based on a FEN input.
2. Be able to send and receive commands from Stockfish using the command line terminal.
3. Configure Stockfish settings from within the program, such as setting the movetime and the number of threads to allocate for Stockfish's use.
4. Input the FEN and get the numerical evaluation, best move, and the best line
5. Be able to generate legal moves from the FEN input.
6. Create a GUI where the chessboard is rendered, the user can input the FEN, toggle Stockfish on/off, set the move time, and number of cores.
7. The GUI outputs the numerical evaluation, the best move, and the suggested line based on the best move.
8. Clicking on any piece in the chessboard shows the available moves for the piece (using legal move generation method). Pieces can only move to squares that are legal.

9. The GUI also shows the list of moves that have already been made by the user.



PERT chart displaying the design goals. Solving the program was separated into 3 main parts: handling Stockfish, finding legal moves, and building the GUI.

2.3 Software Design Considerations

Software requirements were:

1. The programming language that was used was Java.
2. The IDE's used were Eclipse and VS Code, allowing us to integrate them with Github.
3. Software will use an Object Oriented approach. Chessboard will be an object that has ChessPiece. ChessPiece itself will be an abstract class, but each individual piece will have their own class that inherits from ChessPiece, and will be responsible for generating their own pseudo-legal moves.
4. Software will use Model-View-Controller (MVC) architecture. The Model will be the logic (Chessboard, ChessPiece, individual piece classes, Stockfish interaction). The View will be the GUI which will be responsible for outputting to the screen and user interaction. The Controller will be responsible for handling communication between the Model and the View, which won't directly communicate with each other.
5. Multithreading is implemented using the Thread and SwingWorker class. The GUI will be the main thread, while any calls made to Stockfish will be on separate threads. This way the GUI won't freeze while Stockfish processes logic.

6. Multithreading will also be implemented during legal move generation using Producer Consumer architecture. The producers will be the individual chess pieces that will generate pseudo-legal moves, and the consumer will be a process that evaluates the legality of the pseudolegal moves.

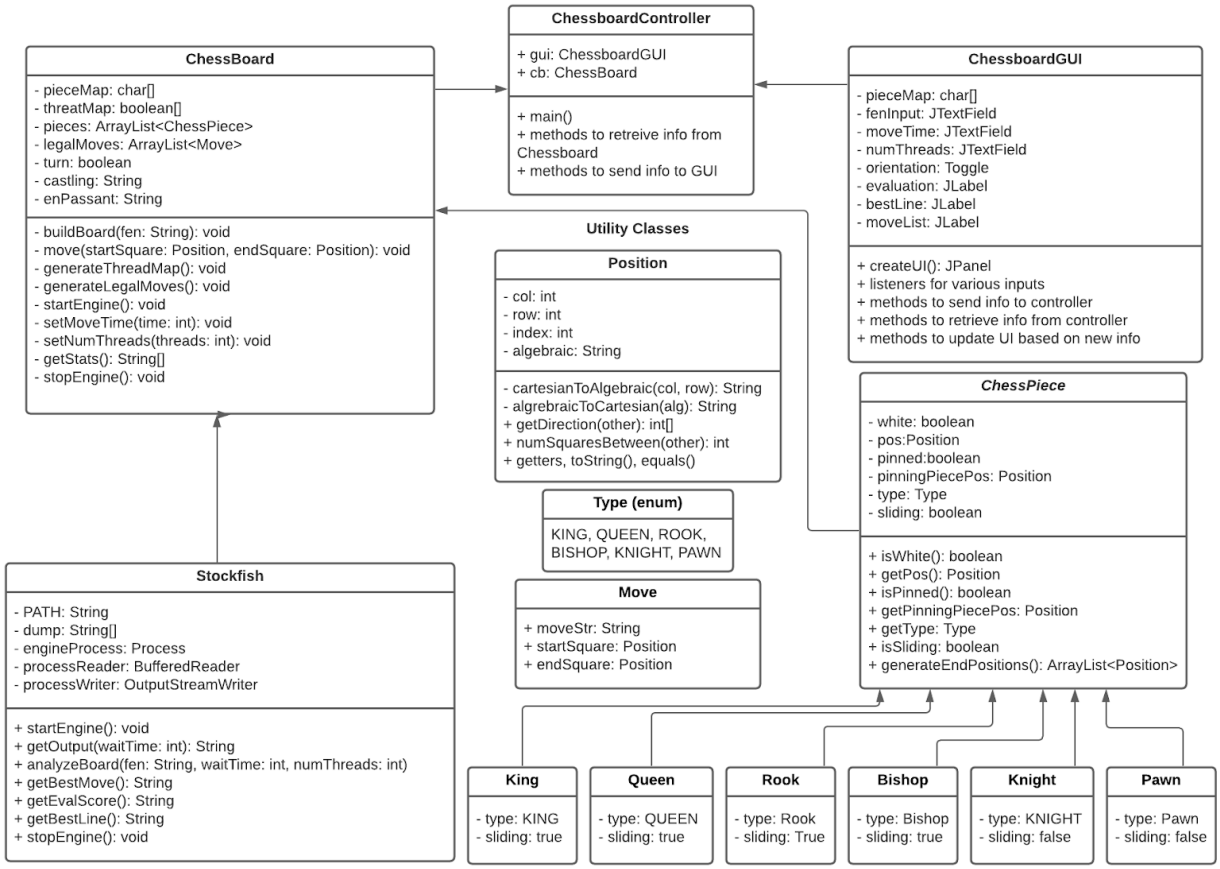
2.4 Functionality

The program would get the following inputs: FEN, time given to evaluate (movetime), and number of threads from the user. Then the program would use Stockfish to find the best moves to make as well as show the numerical evaluation.

Also, in order to prevent the user from making illegal moves, it would need to calculate the legal moves based on the current position. This would be reflected in the GUI when the user clicks on the piece, and the GUI would show the available moves for that particular piece.

3.0 SYSTEM ARCHITECTURE (strategy & details)

3.1 UML Diagram



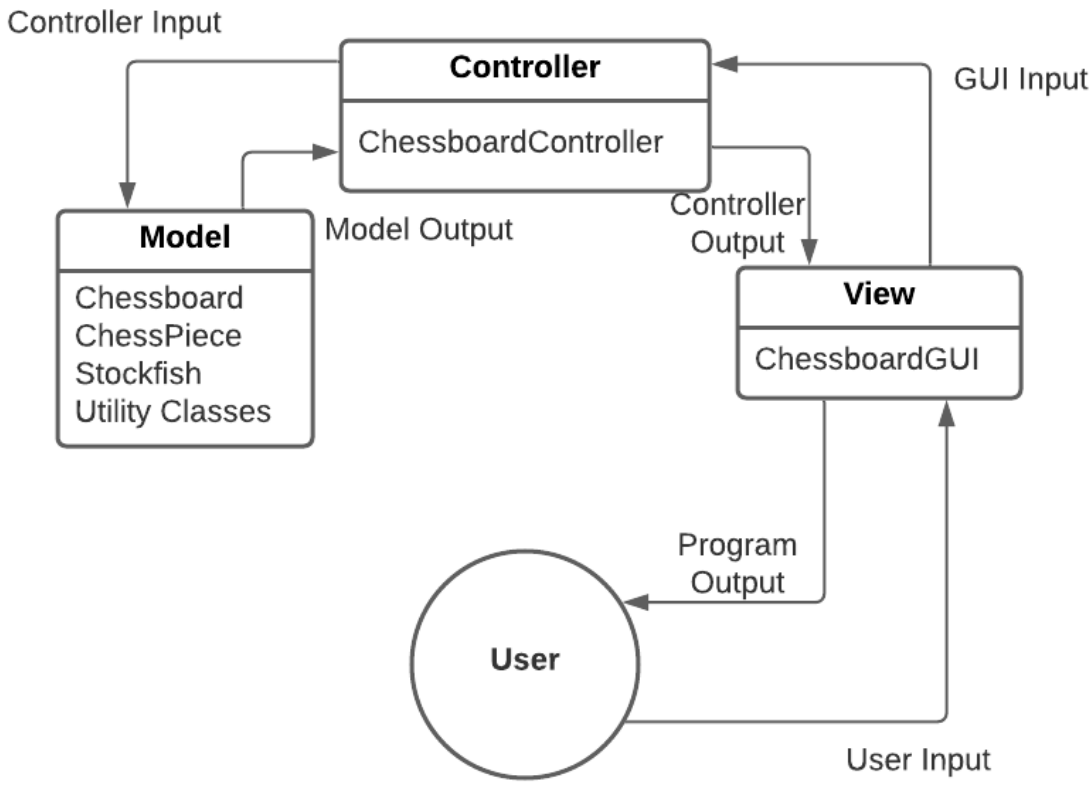
UML Diagram

3.2 Architectural Design (strategy)

We used the Model-View-Controller (MVC) architecture for this project.

1. Model: handles all logic and tracks the state of the game. This includes creating data structures for storing pieces, board positions, communicating with Stockfish, and pieces moving/interacting with each other.
2. View: is responsible for the front end of the project. The chessboard and all outputs are rendered here. This is also where the user can input the FEN, the movetime, and the number of threads to allocate to Stockfish.
3. Controller: handles interaction between Model and View. The Model and View don't interact with each other directly, which keeps the project organized and results in high cohesion and low coupling.

3.3 Decomposition Description (details)



Model View Controller layout

The major subsystems became Packages for Model (handles legal moves, interacts with Stockfish), View (makes the GUI), and Controller (contains the main program).

3.3.1 Model

Chessboard

Keep track of the game state. Builds a chess board and handles the interaction of all the pieces. Interacts with Stockfish in order to find the best move and numerical evaluation of the current position.

ChessPiece

An abstract class that must be initialized using the classes for each individual piece (King, Queen, Rook, etc). Responsible for storing information about each piece, such as its position on the board, colour, as well as methods for generating moves for that piece.

Stockfish

Responsible for sending commands and receiving output from the Stockfish engine.

Position

Defines a spot on the board (eg: a1), as well as provides functionality to easily get the column, the row, and the index of that position.

Move

Defines a movement from one Position to another Position (eg: a1b2).

Type

An enum that defines a chess piece's name (eg: PAWN).

3.3.2 View

Responsible for initializing and designing the GUI, where the user interaction happens. Action listeners for user input would be defined here, as well as methods for sending and receiving information from the controller.

3.3.3 Controller

Connects the chess board logic to the GUI. This allows the subsystems to collaborate with each other. This is where the specific action relating to the user inputs would be executed and the Chessboard class would be called based on the inputs. These methods would use multithreading by implementing the SwingWorker class to call Chessboard on a different thread.

4.0 DATA FLOW

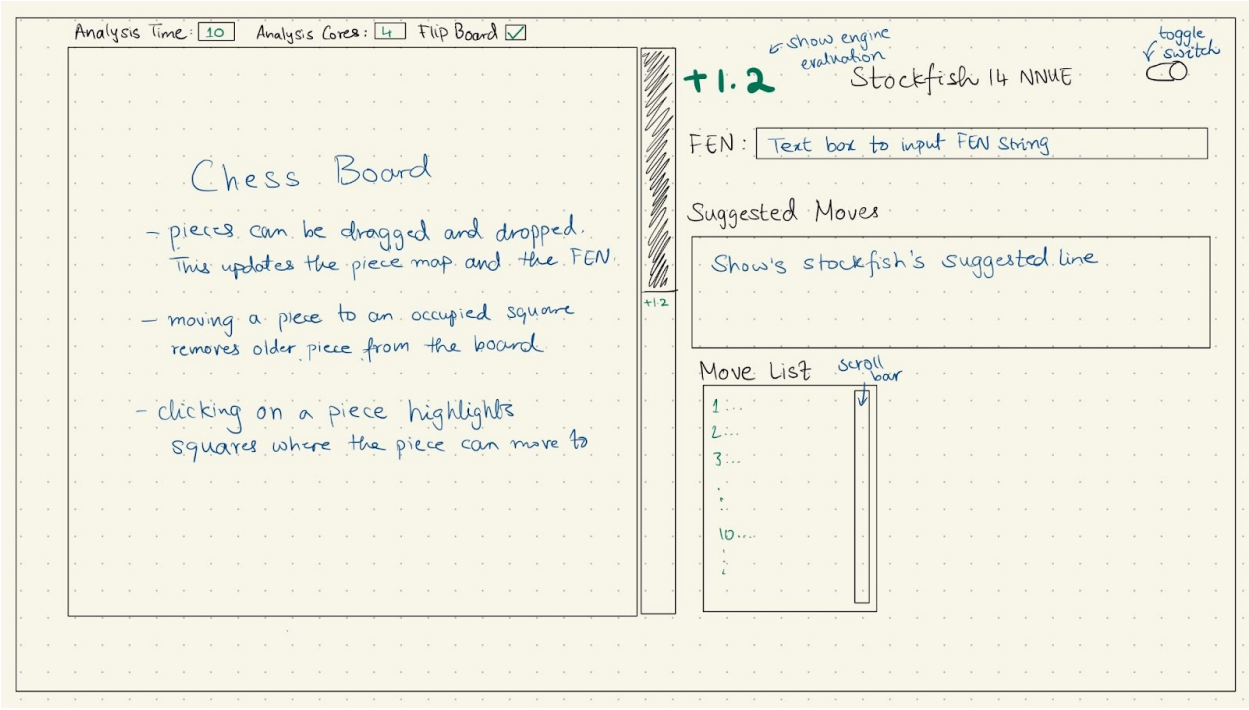
The input FEN string is broken down into its individual components (board state, turn, castling, en passant) and processed.

An array of characters is used to represent each square on the board. Black pieces are represented by lowercase letters, white pieces are represented by uppercase letters, and empty squares are represented by the period symbol. This is useful to find out exactly which piece is on a particular square.

Furthermore, an array list is used to store information about individual ChessPieces. This way, functions for each ChessPiece can be called in an iterative manner when performing computation.

5.0 Overview of User Interface

Our user interface includes a textbox for the user to type in a FEN string, as input to the program. They can also drag and drop pieces if the move is legal. Both FEN string and moving pieces around updates the chess board and its chess pieces.



GUI idea. Consists of Interactable chess board and pieces, input (FEN, time, and cores), previous moves, and a numerical evaluation (Eg: 1.2).

A prototype GUI

6.0 CONCLUSION

In summary, the purpose of our project was to make a chess analysis tool using Stockfish, such as being able to find the best moves given the current state of the board. This was implemented in MVC architecture and was broken up into model handling Stockfish and legal moves, and the view being the GUI.

7.0 REFERENCES

1. 262588213843476, A. (n.d.). *Stockfish - description of the universal chess interface (UCI)*. Gist. Retrieved November 28, 2021, from https://gist.github.com/aliostad/f4470274f39d29b788c1b09519e67372.
2. *Chess Board editor*. lichess.org. (n.d.). Retrieved November 28, 2021, from https://lichess.org/editor?fen=rnb1kbnr%2F1p1p1p1p%2F5qp1%2Fp1p1p3%2FP2P1B2%2F8%2F1PPQPPPP%2FRN2KBNR%2Bw%2BKQkq%2B-%2B0%2B1.
3. Rahular, R. (n.d.). *Rahular/Chess-MISC: MISC tools for rStock*. GitHub. Retrieved November 28, 2021, from https://github.com/rahular/chess-misc.
4. Stefan-Meyer Kahlen. (2004, April). *Description of the universal chess interface (UCI)*. UCI protocol. Retrieved November 28, 2021, from http://wbec-ridderkerk.nl/html/UCIProtocol.html.
5. SyntaxErrorSyntaxError 6311 silver badge44 bronze badges, & SmallChessSmallChess 21.1k22 gold badges3636 silver badges7676 bronze badges. (1963, December 1). *Working with UCI Protocol (coding)*. Chess Stack Exchange. Retrieved November 28, 2021, from https://chess.stackexchange.com/questions/12580/working-with-uci-protocol-coding.
6. derekbanas. (2013, February 21). *MVC Java Tutorial*. YouTube. Retrieved November 28, 2021, from https://www.youtube.com/watch?v=dTVVa2gfht8.
7. MrOverGryph. (2018, November 16). *ACP horse race project using multithreading and graphical user interface in Java*. YouTube. Retrieved November 28, 2021, from https://www.youtube.com/watch?v=8SkzzpJ2gkU.
8. Jonny93Jonny93 111 silver badge22 bronze badges, & ChristopheChristophe 57.7k55 gold badges5858 silver badges115115 bronze badges. (1967, December 1). *How to constuct a UML diagram for a simple chess game with Gui?* Stack Overflow. Retrieved November 28, 2021, from https://stackoverflow.com/questions/58354719/how-to-constuct-a-uml-diagram-for-a-simple-chess-game-with-gui.
9. *How to create software design documents*. Lucidchart. (2020, November 18). Retrieved November 28, 2021, from https://www.lucidchart.com/blog/how-to-create-software-design-documents.
10. *How to write a software design document + free template*. How To Write a Software Design Document + Free Template. (n.d.). Retrieved November 28, 2021, from https://slite.com/templates/software-design-documentation.