**SOFT10101 Computer Science Programming**

**Coursework 2023**

**Title:** Chess Express

**Student ID:** N1103546

**Name:** Lethiwe Mwendwa

**Libraries Used:** SDL2 and SDL\_Mixer

Graphical user interface

Description automatically generated

Specification

Description

Chess Express: A FEN notation expression chess engine created by Lethiwe Mwendwa

A chess program where you’re able to play chess with a friend offline and learn how to read FEN notation by yourself by importing FEN notation files, and seeing the string dynamically change as you make moves.

Software requirements

Here are software requirements that I was able to achieve.

-Register clicks

-Make pieces movable

-Way to identify each tile on the board. (8 ranks and 8 files)

-Way to know if one block is black or white.

-Make graphics for each piece

-Store how each of the pieces can move

-Register checkmates

-Have a menu, allowing player to navigate through options (changing game fen and starting a game)

-Set up FEN notation reader

-Set up a FEN notation position writer

-Get all basic movement done

-Getting basic rules of piece movement done

enable players to quit at any point.

-display pleasing GUI

-implement sound effects relevant to the events happening

**Chess Pieces**

-Graphics for each piece

-How each of the pieces can move

-Assign the pieces the tag “black” or “white”, and change sprite colour depending on that.

-”on the board” state, and “dead” state. If piece dies, delete it

**User Input**

-Register clicks

-Make pieces movable

-Only let current player select pieces

**Start of match**

-Start in the mode chosen in the menu (taken from a .txt file)

-Error handling with preventing invalid file paths

-assign pieces to players after translating FEN

-place the pieces in the correct starting position before game starts

-Display who’s turn it is

**In Match**

-loop the turns

-allow player input

-prevent illegal moves

-stop game if checkmate is met, and move into ‘end of match’ state

**End of Match**

-display who won and who lost

-exit to menu, allowing player to: play again with default fen, or fen of their choice.

**Diagram, text

Description automatically generated**

How I figured out how to implement the Piece Movements. Had a eureka moment, jotted down all my thoughts immediately.

Design and Implementation

Program Flow

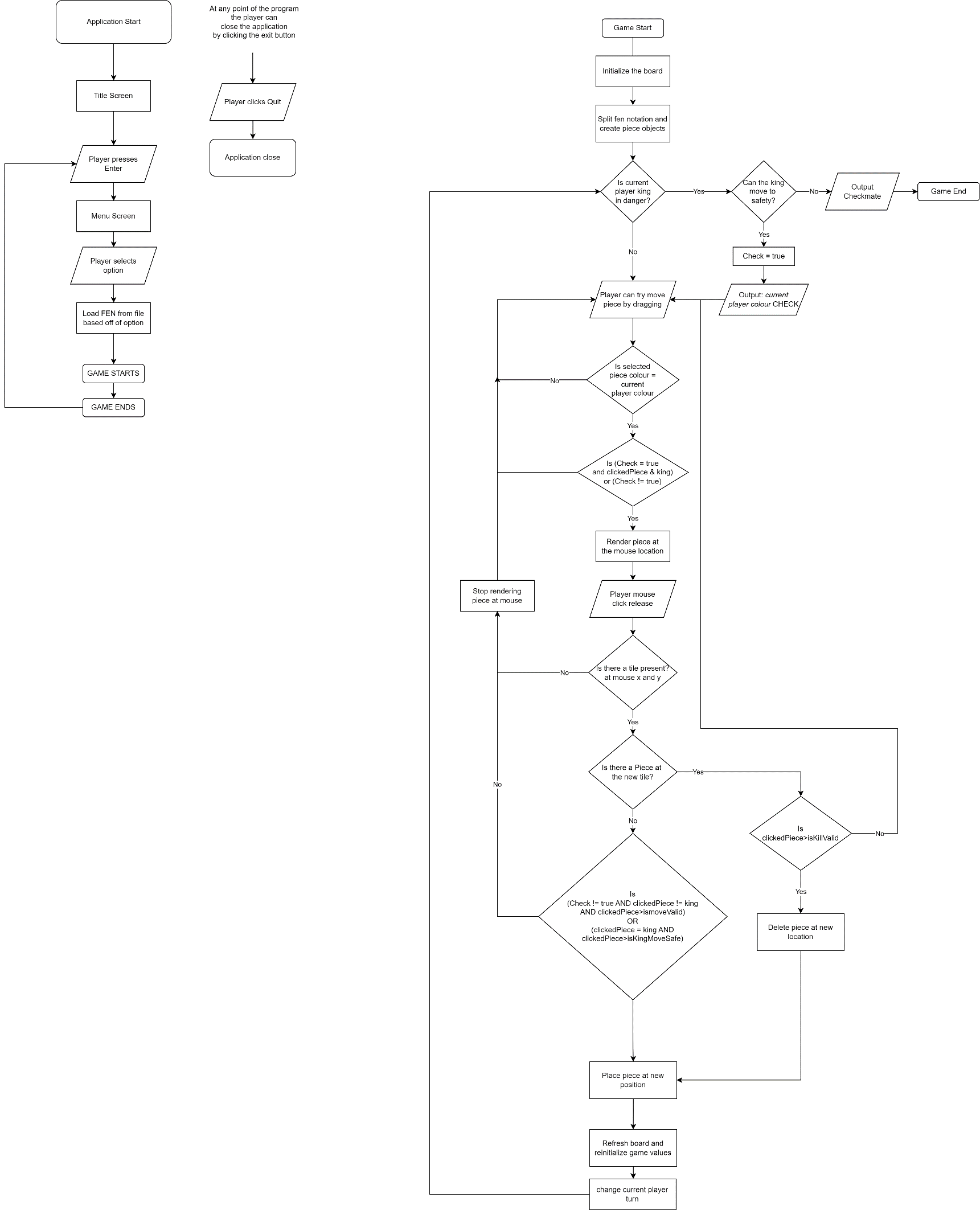
The structure that I went with in my code for each state is an if statement followed by the while statement that loops in that state. I did this to deal with initializations and running code that can’t run in the main while loop, as they would constantly be called. I treated these if statements as spaces for preconditions.

Text

Description automatically generated

Diagram, schematic

Description automatically generated



Classes

This project contains eight classes. Board, Piece, and the 6 children of Piece; King, Knight, Pawn, Queen, Rook, Bishop.

***Board*** has methods to initialize the board, display the board, and update the board and fen notation after a move. Has majority of the general logic.

***Piece*** is an abstract class that represents a chess piece. It contains properties to identify the various values that relate to the piece, e.g., the row and column. It also holds abstract methods for calculating the piece's valid moves and checking if a move is legal.

***King*** inherits from the Piece class and overrides the abstract methods for calculating valid moves and checking legal moves based on the unique movement rules of the king.

***Knight:*** inherits from the Piece class and overrides the abstract methods for calculating valid moves and checking legal moves based on the unique movement rules of the knight.

***Pawn:*** inherits from the Piece class and overrides the abstract methods for calculating valid moves and checking legal moves based on the unique movement rules of the pawn, such as the initial two-square move.

***Queen:*** inherits from the Piece class and overrides the abstract methods for calculating valid moves and checking legal moves based on the unique movement rules of the queen, which combines the movements of the rook and bishop.

***Rook:*** inherits from the Piece class and overrides the abstract methods for calculating valid moves and checking legal moves based on the unique movement rules of the rook, which can move horizontally or vertically any number of squares.

***Bishop:*** inherits from the Piece class and overrides the abstract methods for calculating valid moves and checking legal moves based on the unique movement rules of the bishop, which can move diagonally any number of squares.

Graphical user interface

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

File Structure

The .txt files used in the program are in the “gamefiles” directory in the game project.

(NB, if you’re running from the ChessExpress.exe you must manipulate the “gamefiles” folder in there, and vice versa)

Each text file consists of a string in the structure of Forsyth-Edwards Notation (FEN). No need for any splitting beforehand, I made the program able to handle raw FEN strings by using “Board::fenSplitter(exampleFen[]);”, which splits these values up.

Test Plan

*The following is a text plan for the project: (examples I haven’t changed yet)*

|  |  |  |
| --- | --- | --- |
| **Action** | **Test Method** | **Success Criteria** |
| Go to Menu | Press Enter | Application graphic should change, and menu selections should appear on console |
| Load Game | Press 2 then input a file path | Program is able to cope with incorrect file paths etc.  Passes the FEN value to the game |
| Load Default Game | Press 1 | A new game should start with all pieces in their initial positions, and the player who moves first should be white |
| Move Piece | Click Piece and drag to new location | Move if the move is legal, and the game state should be updated accordingly. If the move is illegal, the piece is snapped back |
| Kill Piece | Click Piece and drag to new location with enemy | Kill if the move is legal, and the game state should be updated accordingly. If the move is illegal or colour is wrong, the piece is snapped back. |
| Check | Put opponent king in danger | “CHECK” outputted |
| Checkmate | Put the opponent's king in a position where it cannot escape | “CHECKMATE” outputted |
| Exit in Menu (Console) | Input 3 | Program ends |
| Exit Normally (Main window) | Click the X button | Program ends |
| Change Size of window | Resize window width or height | The contents of the window  dynamically re-render and shift |

Evaluation

Despite the vast scope of the project, I managed to create a fully functional and responsive chess program. However, I was unable to complete some of the more challenging extra tasks. While I did encounter some issues with program efficiency and encountered stack overflow problems with a particular function, Board::noPiecesCanUncheck(), I am satisfied with the overall outcome of the project.

In the future, I plan to refine the implementation of certain functions and utilize flowcharts more effectively to streamline the development process and make it easier to identify and fix bugs. Despite encountering some unexpected challenges, such as having to dynamically update the fen notation after each turn, I was able to meet my initial project specifications within the two-month timeframe.

Overall, Chess Express was a success, and while there is room for improvement, I am pleased with the results.

Demo

The Demo is an attached video called “LethiweMwendwaDemo”