Queen’s Chess

Ver 1.0

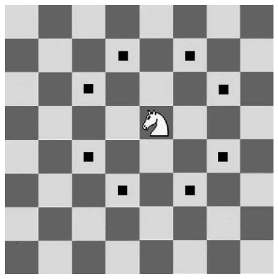
Developed by Queen Studios
In affiliation with UCI

Developers: Neel Sankaran , Oliver Amjadi , Jordan Queen, Sai Sandeep Reddy B., AJ Smyth, Yi Sien Ku

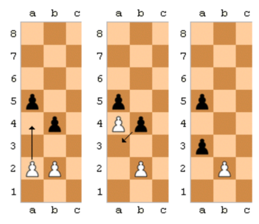
Table of Contents

Glossary

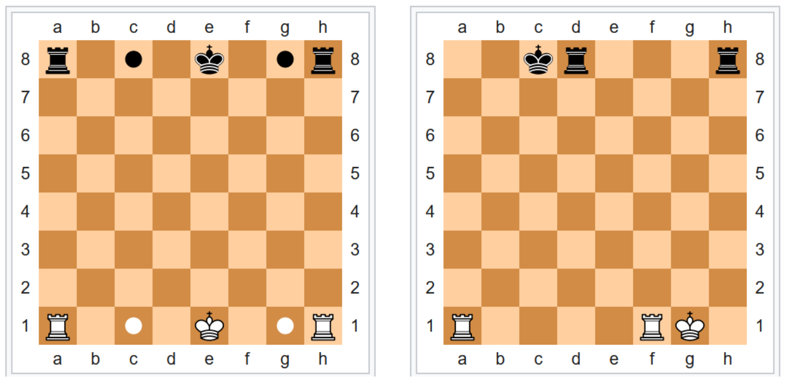
Pawn - The pawn is a chess piece that can move forward one tile or two tiles on a player's turn or can capture a piece diagonally one spot forward towards the left or right.
Rook - A castle looking chess piece that can move forward or backward or side to side.
Bishop - A chess piece that can move diagonally
Knight - The knight is a chess piece that can move in an L shaped pattern. It can capture pieces that are two moves forward, backward or side to side and then left or right.



Queen - A chess piece that can move in the same way that both the rook and bishop can move
King - The main chess piece. If lost, indicates loss. The king can move left 1 position, right 1 position, up 1 position, or down 1 position.
Chess Algebraic Notation - A convention for notating the moves that occur in a game. Read sequentially it can be used to reproduce an entire match.
Capturing en Pessant - If a pawn moves forward twice, then an enemy pawn can capture it on the next term as if it was only moved one space forward.



Castling - The rook can move beside the king and then the king and rook can switch pieces.



1. **Struct** - A collection of variables with different(or the same) type under one reference.

2. **2D Array** - An array of arrays.
3. **Pointer -** A pointer is a variable that holds the address of value with a given type.
4. **Minimax -** an algorithm that searches a decision tree and finds the best course of actions given a function that generates a value for a given game state.
5. **Alpha Beta Pruning -** an algorithm used in conjugation with minimax to reduce the number of decisions analyzed. It can be thought of as a measure to ignore redundant decisions or choices that lead to the same outcome.

1 Software Architecture Overview

Main data types and structures

Data Types:

EPieceType

EColor

EGameState

***see section 3 for more information**

Data Structures:

Board

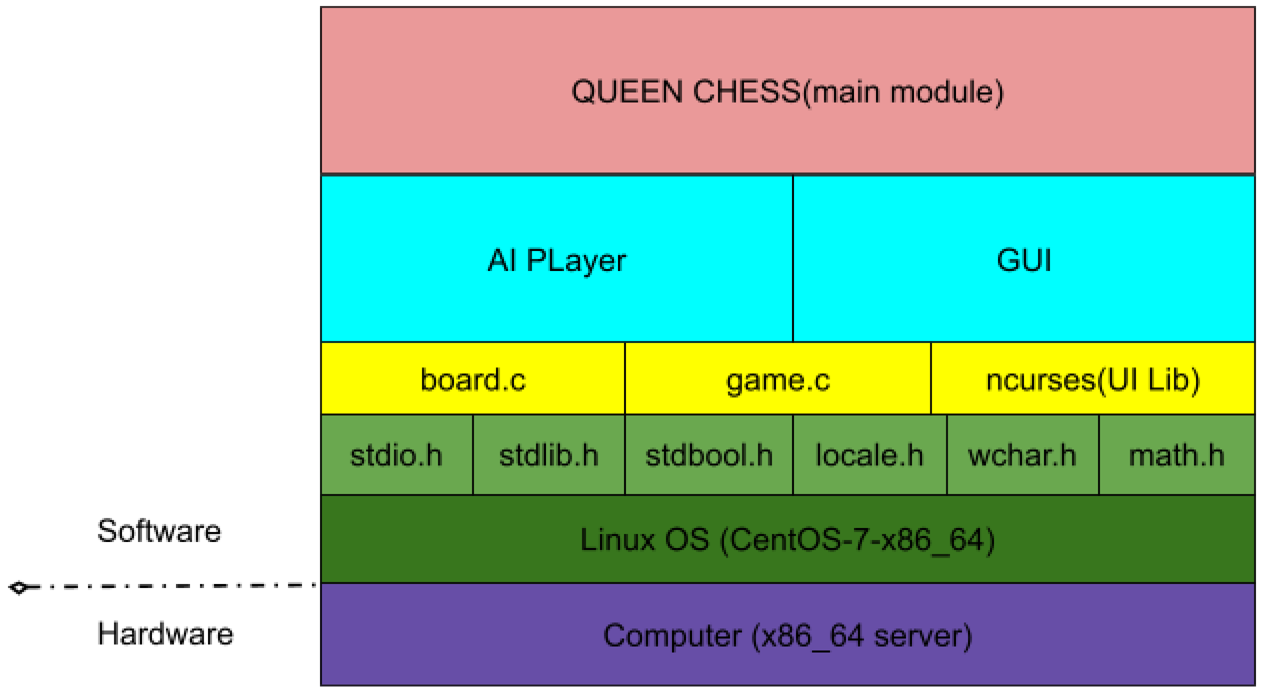
Piece

MOVE

***see section 3 for more information**

Major software components

Diagram of module hierarchy:



Module interfaces

game.c - validation of game moves, and controls logic that allows the player to move their piece in a certain fashion.

Functions:

- Move
- IsValid
- IsInCheck

- IsMated
- Capture
- Promote
- ValidMoves

board.c - contains the code used to print the board and the data structures for the board
Functions:
- PrintBoard
- FillBoard
- GetUnicode

main.c - contains the main game loop

Overall program control flow:
1. The game is started by the player from the main menu.
2. The chess board is initialized then printed.
3. The player proposes a move.
4. The move is analyzed then the board is updated and reprinted if the move is valid.
   a. If the move is invlaid the player is prompted to re-propose a move.
   b. If the move results in a checkmate the game is ended.
5. The AI then looks at the player's move and calculates the best possible move from the current game state.
   a. There are many ways of doing this but we will probably be using a *minimax*[4] algorithm with *alpha beta pruning*[5] for optimization.
   b. The AI player can only make valid moves.
6. The game board is then checked for a checkmate and ends if there is one.
7. If the game has not ended the loop resets to step (2) in which the player proposes a move.
8. The player can exit the game at any time.
9. Exiting the game leads back to the main menu.

2 Installation

System Requirements:

System Characteristic	**Recommended specification**
Processor	64-bit Opteron, EM64T

RAM	1 GB or Higher
Swap space	1 GB or Higher
Disk space	500 MB of Free Space
Operating System	Linux Kernel Stable Release 5.0+ (Latest: 5.17.1)

Setup and Configuration:
Boot up your linux operating system and ensure that it is running properly so that it can run the file without error. Access the directory where the binary file is located and follow the installation instructions to successfully download and run the chess game.

Installation:
Download the binary file, open a terminal window and navigate to the directory to which the binary was downloaded via "cd." Configure the permissions to allow the binary to run with "sudo chmod +x queenChess.bin". Now run the executable via ./queenChess.bin, taking care that you are still in the correct directory.

Uninstallation:
Simply delete the binary. To do this, open a terminal window, navigate to the directory containing the program via "cd" and remove the program with "sudo rm queenChess.bin".

3 Documentation of packages, modules, interfaces

3.1 Description of Data Structures:

EPieceType (`board.h`):

EpieceType is an enum *struct*[1] that is used inside the Piece struct to define the type that a piece is(eg. King, Queen, Bishop, etc…). Note that a type exists for EMPTY. This is to ensure that the Piece struct can also describe a space where there is no piece present(this will be discussed in detail later).

```
typedef enum {
    EMPTY,
    PAWN,
    ROOK,
    KNIGHT,
    BISHOP,
    QUEEN,
    KING
} EPieceType;
```

EGameState (`board.h`):

Describes a win condition. Each value describes a specific win condition according to the color winning.

```
typedef enum {
    WHITE_WIN,
    BLACK_WIN,
    STALEMATE
} EGameState;
```

EColor (`board.h`):

EColor is an enum similar to EPieceType that simply encodes the color of a piece. It is an attribute within the Piece struct. Since our gameboard is represented as a *2D array*[2] of Piece structs, the Piece struct needs to be compatible with the description of an empty slot. As a result, no color must be an option within EColor.

```
typedef enum {
    WHITE,
    BLACK,
    //no color used for empty tiles, which are still technically pieces
    NO_COLOR
} EColor;
```

Piece (`board.h`):
Describes a piece. Pieces are constructed into 2D arrays (chess boards), and each tile is represented by a piece type. Empty tiles are also represented by empty pieces, but with a color value of NO_COLOR and a piece of EMPTY. The hl member represents if the tile is highlighted.

```
typedef struct {
    EColor color;
    EPieceType piece;
    bool hl;
} Piece;
```

Board (`board.h`):
A structure whose only member variable is a 2D-array of *pointers*[3] to piece objects. Represents a chess board. Can be initialized to the initial chess board state with the FillBoard() method.

```
typedef struct {
    //board[file][rank], [0][0] corresponds to A1, [7][7] to H8
    Piece *board[8][8];
} Board;
```

Move (`game.h`):
A Move struct simply holds all the values that parameterize a move in chess. Since we will have access to the chess board array inside our code, the only values we will need to make a move will be the starting address of the piece and the ending address of the piece. These addresses are stored in terms of rank and file within the Move struct.

```
typedef struct {
    int r0;
    int f0;
    int r1;
    int f1;
} MOVE;
```

3.2 Descriptions of Functions and Parameters:

Move(`game.h`):

```
char *Move(Move *m, Board *b);
```

IsValid(`game.h`):

The IsValid function checks if a specified move is valid given the current game state specified by the board parameter. A boolean is returned with true representing that the move is valid and false representing that the move is invalid.

```
bool IsValid(Move *m, Board *b);
```

Parameters:

- Move pointer: a pointer to the move being analyzed.
- Board pointer: a pointer to the board in which the move is taking place.

IsInCheck(`game.h`):

The IsInCheck function returns true if a specified move results in a check within the game state specified by the board parameter.

```
bool IsInCheck(Move *m, Board *b);
```

Parameters:

- Move pointer: a pointer to the move being analyzed.
- Board pointer: a pointer to the board in which the move is taking place.

IsMated(`game.h`):

The IsMated function returns true if the player specified by the color parameter has been checkmated within the game state specified by the board parameter.

```
bool IsMated(EColor color, Board *b);
```

Parameters:

- EColor: the value of the color of the player being analyzed.
- Board pointer: a pointer to the board in which the player is playing.

Capture(`game.h`):

The Capture function removes a piece specified by the rank and file parameters from the board specified by the board parameter. The Piece is then added to a sideboard for display purposes.

```
void Capture(int f0, int r0, Board *b);
```

Parameters:

- int f0, int r0: the rank and file of the piece being captured.
- Board pointer: a pointer to the board in which the piece resides.

Promote(`game.h`):
The promote function promotes a piece at a specified rank and file within a specified board.

```
void Promote(int f0, int r0, Board *b);
```

Parameters:

- int f0, int r0: the rank and file of the piece being promoted.
- Board pointer: a pointer to the board in which the piece resides.

ValidMoves(`game.h`):
These functions are internal helper functions for the IsValid function. Each function returns an array of moves which represent the possible moves a single piece can take given a location and a board state.

```
void getValidMovesQueen(int f, int r, Board *, Move **);
void getValidMovesRook(int f, int r, Board *, Move **);
void getValidMovesKnight(int f, int r, Board *, Move **);
void getValidMovesBishop(int f, int r, Board *, Move **);
void getValidMovesPawn(int f, int r, Board *, Move **);
void getValidMovesKing(int f, int r, Board *, Move **);
```

Parameters:

- int f, int r: the rank and file of the piece being analyzed.
- Board pointer: a pointer to the board in which the piece resides.
- Array of Move pointers: a pointer to an array of Move Pointers used to store all the possible moves for the specified piece. This array will be passed in by the user calling the function(this is a common C convention for returning arrays).

PrintBoard(`board.h`):
This function simply prints the specified board with a clean and understandable format.

```
void PrintBoard(Board *board);
```

Parameters:

- Board pointer: a pointer to the board being printed.

FillBoard(`board.h`):
This function initializes a board pointer with piece values matching the starting board of a standard chess game.

```
void FillBoard(Board *board);
```

Parameters:
- Board pointer: a pointer to the board being filled.

GetUnicode(`board.h`):
This function returns the unicode character of a piece with a specified color and type. This is for display purposes exclusively.

```
wchar_t GetUnicode(EPieceType, EColor);
```

Parameters:
- EPieceType: enum for the piece's type.
- EColor: enum for the piece's color.

3.3 Detailed description of input and output formats

Syntax/format of a move input by the user

- Currently, the user can input the location of the piece he/ she wants to move, and can enter the desired location he/ she wants to move the piece to, with a space in between both the locations. The location of a piece is determined by its file and rank number.
- For example, if a user wants to move a pawn in location B2 to B4, he inputs "B2 B4". The system then checks if the move is valid, and moves the piece to the desired location if it is valid.
- Projection: Our current method of moving pieces is by using keyboard input, but we plan on changing this, and use the cursor as the input. We will highlight the piece clicked on, and show the possible moves, which can be chosen using the cursor itself. This is a very intuitive and easy way of playing the game compared to the keyboard input, because of which we think it will be more suitable.

Syntax/format of a move recorded in the log file

- The initial file and rank of the piece are called "f0" and "r0" in the code and the final file and rank are called "f1" and "r1". When the user inputs the intended move, the value gets stored into these variables for initial position and final position respectively. The log file already contains the position and value of each piece because of the use of the FillBoard function at the start of the game.

- We then take the given values and run them through the function IsValid to check the validity of the move. We also run the moves through functions IsInCheck, IsMated, Capture and Promote to check for a checkmate and other special conditions. If any of the special conditions are satisfied, we alter the move in a way that is required for the specific condition. Then we use *Move to move the piece from the initial position to the final position.

4 Development plan and timeline

4.1 Partitioning of tasks

Weeks 1 and 2	Weeks 3 and 4	Week 5
● Instruction Manual ● Software Spec ● 500 lines of code for alpha ○ initial board design ○ and validating movements	● Set up Algebraic Chess Notation to control movement ○ Deal with user input ● Try to get complete running game ● Start AI	● Test,troubleshoot, and finish AI to the best of our abilities

4.2 Team member responsibilities

Ian Ku - Documentation, Software Validation, Board Utilities, Instruction Manual
Oliver Amjadi - Validation moves, AI Design, Board Utilities, User input, Instruction Manual
Jordan Queen - AI Design, User input
Neel Sankaran - GUI Design, AI Design
Sandeep - AI Design, Board Utilities
AJ Smyth - GUI design, AI Design, Board Utilities, User input

Copyright and Licensing

Contact Information

info@queen.com

Index

References

https://www.chess.com/terms/en-passant
https://www.chess.com/terms/chess-notation