# The Best Chess Game Ever

Version 1.0 (Software Specification)

Team:

***Team Charlie***

Members:

***Devin Pham, Vivian Lam, Allison Cornell, Adrien Guillaume, Michael Yang***

EECS 22L (Winter)
University of California, Irvine

# Table of Contents:

# Glossary:

**Bishop:** May only move diagonally as many squares as long as the pathway is not blocked.

**Castle:** A special move which involves moving the king and rook simultaneously. This will be the only time where two pieces can be moved in the same turn. There exists two types of Castle:
-*Castle Long*: *Queenside Castling*
-*Castle Short: Kingside Castling*
Both types of Castling consist of moving the King two squares either right or left depending on the type of Castling, and moving the rook on the square besides the King closest to the center. Keep in mind that this move can only be done if there are no other chess pieces in between the King and Rook, if neither pieces (King and Rook) have already moved, and if the King does not move out of a Check, into a Check, or over a Check. This special move allows the King to be in a safer position while giving the Rook a powerful position in the middle of the board.

**Check:** The act of attacking the opponent's King. When in Check, the player can call out "Check" to inform his opponent of the threat. In this case, the opponent must either move his King in a safe square, capture the attacking piece, or move another piece in between the King and the attacking piece.

**Checkmate:** The act of attacking the opponent's King such that it cannot escape by any means. The game is then over, and the attacking side wins.

**CPU:** Computer generated algorithm that plays against another player.

**En Passant:** from the French term "in passing". This move only occurs after a pawn moves two squares from its starting position, and passes an enemy pawn. The enemy pawn then has the opportunity to capture the passing pawn as if it had only moved forward one square. The right to capture "en passant" must be made immediately during the next move by the enemy's pawn, or else it will be lost in the following moves.

**Fork:** an attack where a piece threatens two or even three enemy pieces at the same time.

**King:** Can only move 1 step in the surrounding squares.

**Knight:** May only move in the shape of a ‘L’. For example, 2 steps forward and 1 step right.

**Pawn:** For each pawn, the first move may either be 1-2 steps forward. After the first usage of that Pawn, they may only move 1 space forward at a time. The Pawn may also move diagonally forward 1 step to capture the opponent’s piece.

**Queen:** Can move 1 step in the surrounding squares. Can also move vertically, horizontally, and diagonally as many spaces as long as the pathway is not blocked.

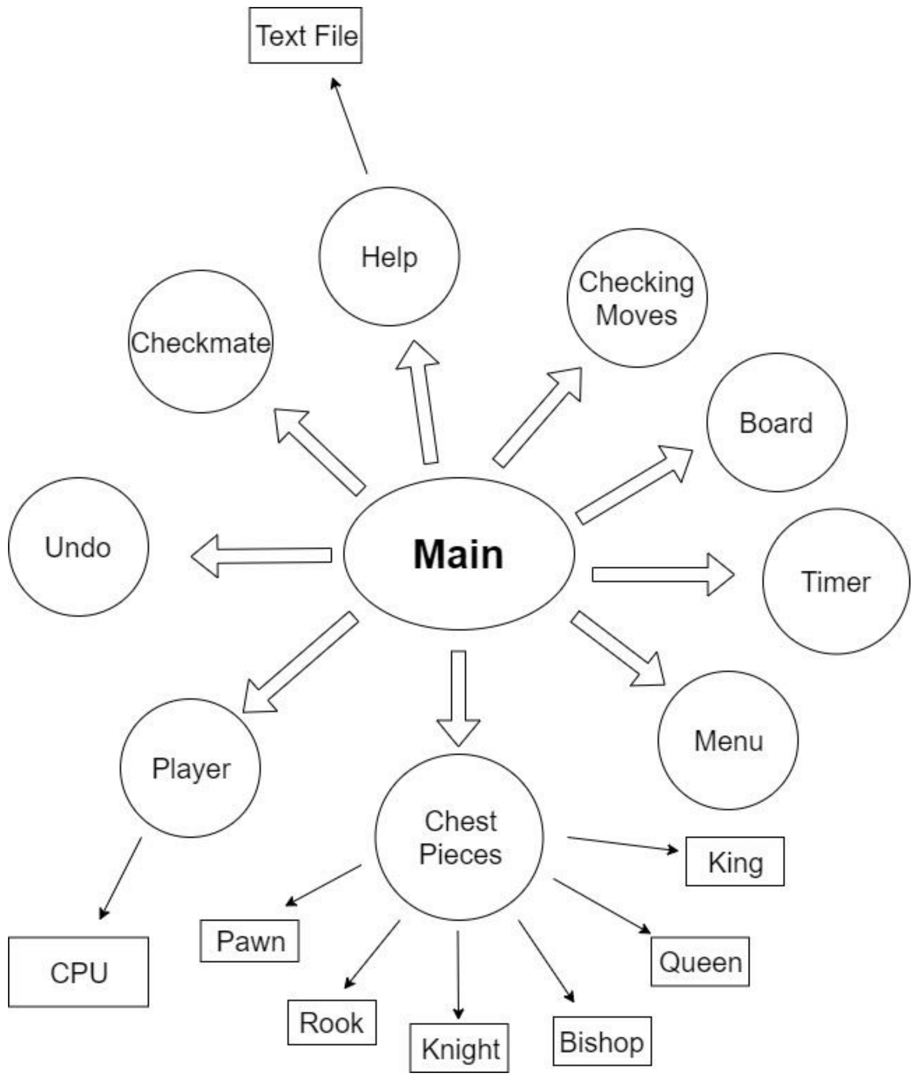
**Rook:** May move vertically or horizontally as many squares in the horizontal or vertical direction as long as the pathway is not blocked.

# 1: Software Architecture Overview

## 1.1. Main Data Types and Structures

The main data types include a structure containing all the chess pieces (King, Queen, Bishop, …), an array for the board [8][8], and other modules depending on integers.

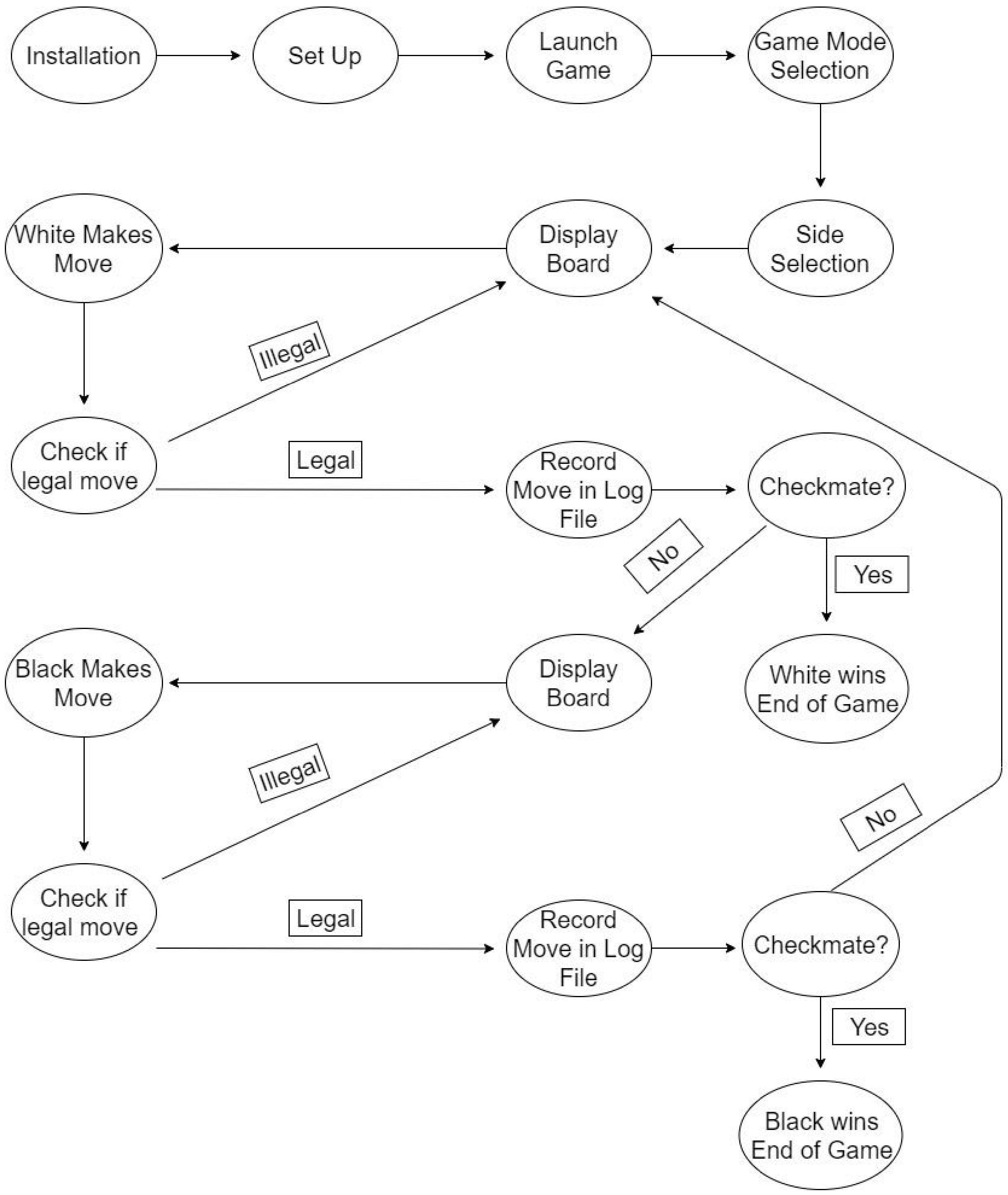
## 1.2 Major Software Components



## 1.3 Module Interfaces

- Main
  - main()
- Board
  - displayBoard()
  - log()
- Checking_Moves
  - pawn(int array[])
  - queen(int array[])
  - king(int array[])
  - knight(int array[])
  - bishop(int array[])
  - rook(int array[])
- Checkmate
  - checkmate(board)
- CPU
  - cpu(board)
- Help
  - help()
- Log
  - log(int round)
- Menu
  - Challenger()
  - Color()
  - PrintMenu()
- Player
  - player()
- Timer
  - timer()
- Undo
  - undo(int round)
- Chess_pieces
  - struct chess_pieces{ }

## 1.4 Overall Program Control Flow



# 2: Installation

---

## 2.1 System Requirements and Compatibility

| | |
|---|---|
| OS: | Linux with Display Managers Compatible with X11 and SDK/GTK |
| Processor: | 2 GHz Intel Dual-Core Processor |
| Memory: | 900 MB Available |
| Graphics: | Video Card with at least 1GB VRAM and Supports Shader Model |
| Storage: | 1 GB Available |

## 2.2 Setup and Configuration

The chess software will be compressed into a package with the extension ".tar.gz". To untar please type:

```
gtar xvzf BinaryArchive.tar.gz
```

Please read the provided documentation before continuing with the installation process by typing:

```
evince chess/doc/chess.pdf
```

## 2.3 Building, Compilation, Installation

Please type the following to run the package:

```
chess/bin/chess
```

# 3: Documentation of Packages, Modules, Interfaces

## 3.1 Detailed Description of Data Structures

- Board
    - Function where the 2 dimensional array is used to display the chess board. It accesses the log function to show the current board in play.
    - Skeleton Code:

      ```
      displayBoard(...)
      {
              /* Log() */
              /* Displays the Board */
      }
      ```
- Checking_Moves
    - Module that includes functions to check if the move is valid for the different pieces
    - Skeleton Code: (general idea is the same for all pieces. pawn() is just used as an example)

      ```
      pawn(int array[ ])
      {
              /*gets the value of where the piece was originally*/
              /*gets the value of where the piece is being moved*/
              /*compares them to see the offset*/
              /*determine if the move is valid based off of that offset*/
      }
      ```
- Checkmate
    - Module contains a function that checks for a checkmate or a check.
    - Skeleton Code:

      ```
      checkmate(board array)
      {
              /*gets the current position of the pieces from the board*/
              /*checks the criteria for the king to be in check*/
              /*if the king is in check then display a message to the user*/
              /*check the criteria for checkmate*/
      ```

```
        /*display a message to the user*/
        /*if checkmate then end the game*/
}
```

- CPU
  - Calculates the best possible moves for the computer to make
  - Skeleton Code:

```
cpu(board)
{
        /*looks at where the pieces on the board currently are*/
        /*call the checking moves functions*/
        /*out of the valid moves pick the best one*/
        /*make the move*/
}
```

- Chess_pieces
  - struct chess_pieces{ }
    - A struct to represent all of the different chess pieces
  - Skeleton Code:

```
struct chess_pieces
{
        *King
        *Queen
        ……...
}
```

- Help
  - Displays the official chess rules at the beginning of the game and if the user asks for help during the game by typing **help**. Typing the keyword will pull the chess rules on the terminal to remind the user.
  - Skeleton Code:

```
help()
{
        /* Chess Rules

        Chess Pieces Movement
            ○ Pawn: For each pawn, the first move may either be 1-2 steps
              forward. After the first usage of that Pawn, they may only move 1
              space forward at a time. The Pawn may also move diagonally
              forward 1 step to capture the opponent's piece.
            ○ Rook: May move vertically or horizontally as many squares in the
              horizontal or vertical direction as long as the pathway is not
              blocked.
```

        - Knight: May only move in the shape of a ‘L’. For example, 2 steps forward and 1 step right.
        - Bishop: May only move diagonally as many squares as long as the pathway is not blocked.
        - Queen: Can move 1 step in the surrounding squares. Can also move vertically, horizontally, and diagonally as many spaces as long as the pathway is not blocked.
        - King: Can only move 1 step in the surrounding squares.

    Gameplay
        - The White pieces always make the initial move.
        - If the Pawn makes it across the board, the Pawn must be promoted to either: a Queen, Rook, Knight, or Bishop.
        - If the player hasn’t moved their King and Rook, they may move their King 2 steps towards the Rook and place the Rook on the other side of the King to protect it, known as “Castling”. Castling may also be applied with a Queen and a Rook. Castling can’t be used to avoid being checked or getting into check.
        - If a pawn moves two squares instead of one from its starting position in order to avoid capture, the opponent can capture the just moved pawn as it “passes” through the first square which is also known as “En Passant”. The opponent must do the en passant capture on their next turn or they may not do so. The result of the en passant capture is the same as if the pawn only moved one square instead of two.
        - When the King is at risk of being checked”, it is “checked”.
        - If the player is unable to make any moves to get out of being “checked” then the game is “checkmate” with that player losing the match.
        - If the play remains unchecked and has no legal moves, the match will result in a draw. */

    }
- Log
    - In the log function, it keeps track of all movements made with each team. It shows the rounds they are on, which move to which move, whether a piece has been captured, and whether the piece has been promoted to a certain piece.
    - Skeleton Code:

```
log(int round)
{
        /* Round # */
        /* Black: # to # */
        /* White: # to # */
        /* Displays whether if capture is made */
        /* Displays if piece has been promoted */
        /*Displays if there is a check*/
        /*Displays if there is a checkmate*/
}
```

- Main.c
    - Initially asks the user if they would like to start a game. Asks whether or not they would like to play against another player or against a CPU. Ask the user what team they would like. Displays the help module (rules of the game) before starting the game.
    - Skeleton code:

```
main()
{
        /* Asks user if they want Player vs Player or Player vs CPU */
        /* Obtains user input until it is valid */
        /* Asks user what team they would like */
        /* Obtains user input until it is valid */
        /* Displays the rules of the game */
        /* Tells user to ask for help during game if needed */
        /* Asks user if they would like to begin the game */
        /* Obtains user input until it is valid */
        /* Displays PrintMenu() */
        /* If user wants to make movement, accesses CheckMoveValid() */
        /* Loops until user chooses to exit game or checkmate */
}
```

- Menu
    - Has separate functions for menu displays. First function shows the options for Player vs Player or Player vs CPU. Another function shows the option to choose their team color. The last function will show whether the player wants to make a movement, undo, help, or exit.
    - Skeleton Code:

```
Challenger(...)
{
        /* 1. Player vs Player */
        /* 2. Player vs CPU */
}

Color(...)
{
        /* 1. White Piece */
        /* 2. Black Piece */
}

PrintMenu(...)
{
        /* 1. Make Movement */
        /* 2. Undo Move */
        /* 3. Help */
        /* 4. Exit Game */
}
```

- player()
  - Keeps track of whose turn it is and calls the functions in the Checking_Moves module. Called in main.
  - Skeleton Code:

```
player()
{
        /*checks whose turn it is*/
        /*calls the functions in Checking_Moves (pawn, rook, etc)*/
}
```

- Timer
  - Sets a one minute timer for the user if they want to use it.
  - Skeleton Code:

```
timer()
{
        /*set timer for 1 minute*/
        /*reset timer after the 1 minute expires*/
}
```

- Undo
  - In the undo function, it undos the previous move the user made. Accesses the log function and erases the previous move.
  - Skeleton Code:

```
undo(int round)
{
        /* Checks if there is an undo to be made */
        /* log(int round--) */
}
```

## 3.2 Detailed Description of Functions and Parameters

- displayboard(...)

    - Displays an 8x8 chess board with the chess pieces at their starting positions. After the player makes a move the board will reprint with the updated location of the piece.
    - Returns the board
- Checking_Moves
    - This is a module which contains functions for each piece. Pawn(int array[ ], color), Rook(int array[ ], color), etc.
    - The functions for each piece check if the move the player made is valid. They all take in the move the player made as an array position and the color of the player. They save the position as the current position for that piece.
    - Returns if the move is valid or not
- checkmate(*board*)
    - Function takes in the positions of all of the pieces on the board.
    - Looks for where the current pieces are on the board and checks to see if the king is in check. If the king is in check then the player is told. If the king is in checkmate the player is told and the game ends
    - Returns if there was a check or checkmate
- cpu(*board*)
    - This function takes in the position of the pieces on the board and calculates the best possible move based on the valid moves. This function is used when the user wants to play against a CPU instead of another player.
- help()
    - If the user types “help” a text file will open and display with all of the rules of chess.
- log(int round)
    - Will scan what the user typed as their move and save it in a text file
- main()
    - While loop to go through the game process until there is a checkmate
    - Returns 0
- Menu
    - Module contains functions that print the options for the user so they can pick their color, if they want to play against the CPU, etc. Scans for the users choices and makes sure they are valid inputs. If the input is invalid the user will be prompted to try again.
    - Functions include: Challenger(), Color(), PrintMenu()
- player()
    - Keeps track of which colors/players turn it is. Calls the functions that checks if the moves are valid or not that are in the Checking_Moves module.
- Chess_pieces
    - struct chess_pieces{ }
        - A struct to represent the different chess pieces
- timer()

    - The timer function will set a 1 minute timer if the user chooses to play with a timer when the option is displayed in the menu. The timer will reset for the player after each turn. In other words the player will have 1 minute to make their move per turn. If the user does not make their move within that time then a random move will be made for them.
    - Returns 0
- undo(int round)
    - Checks if an undo is possible. If it is possible the piece moves back to the position it was in before and the user will be prompted to move again.

## 3.3 Detailed Description of Input and Output Formats

Syntax/format of a move input by the user:

If the user wants to make a move, the syntax is the following (user inputs are shown in red):

```
/*play has already chosen their challenger and their color*/

1. Make Movement
2. Undo Move
3. Help
4. Exit Game
Please enter a number: 1
Make your move (example format: A2 to A4): A3 to A4
Please enter a number: 1
…….
```

Syntax/format of a move recorded in the log file:

After each move the user/cpu makes, the move will be recorded in the log file in the following format:

```
Round #
Black: # to #
White: # to #
…….
/*Displays whether if capture is made*/
capture
/* Displays if piece has been promoted */
promotion
/*Displays if there is a check*/
check
```

```
/*Displays if there is a checkmate*/
checkmate
```

# 4: Development Plan and Timeline

## 4.1 Partitioning of Tasks

Devin Pham:

- CPU Module and Undo function

Allison Cornell:

- Modules containing the function for checking if chess moves are valid
- The function to print the board

Vivian Lam:

- Creating the function for the log of player moves
- A function that will keep track of the color for each player and use the checking moves function.

Adrien Guillaume:

- Creating the struct for the chess pieces
- Timer that the player has the option to use

Michael Yang:

- Create the menu the player will see when they open the game with all of the prompts for the player to choose their color, if they want to play against a CPU or another player, etc
- The help menu that will display the rules of chess to the player
- Function to check for a checkmate/check

## 4.2 Team Member Responsibilities

Devin Pham: Manager

Makes sure the team is up to par with their duties, makes sure the team follows through with their deadlines, and checks if each team member is doing their part.

Allison Cornell: Recorder

Keeps a record of everything said or made amongst the team. Records each team members duties for each assignment.

Vivian Lam: Presenter

Addresses the issues the team members are facing and then addresses it with the class.

Adrien Guillaume & Michael Yang: Reflector

Reevaluates the team's efforts and discusses what options are the best.

# Back Matter:

---

## Copyright

## References

U.S. Chess Federation. “Let's Play Chess .” *US Chess Federation:* www.uschess.org/archive/beginners/.

Dang, Quoc-Viet. “Project 1.” EECS 22L. University of California - Irvine, California. 19 Apr, 2020.

# Index: