# Ultimate Chess

V1.0

EECS 22L

**University of California Irvine**
**Henry Samueli School of Engineering**

## Team Bug Busters

Chris Rodriguez: Presenter

Raymond Duong: Recorder

Sunshine Jennings: Reflector

Orion Serup: Reflector

Ian Poremba: Manager

# Table of Contents

# Glossary

- ❑ **Pieces:**
  - ❑ **Bishop** - Each side has **two bishops** on a white and black tile. There are **two bishops** on each side located respectively on the **first and eighth row on the players own side, three spaces away from the corners**, next to the king and queen. The bishop can only **move diagonally** along the color it starts on. It may only capture pieces of the opposing team that are in a direct diagonal line with it, and may not capture a piece that has another piece of either color in between the piece it is trying to capture and itself. It also is **incapable of "jumping"** over a piece that is in its way. Typically known in value as being worth **three points**, although this is for reference as the game is not based on points.

  - ❑ **King** - Each player has **one king**, initially located on the **first and eighth row** of either team's side of the board. It starts **next to the queen**, for the starting side it is located to the right of the queen on the 4th space from the right side of the board. The opposing team's king lies on the same column directly on the opposite side of the board. The king may **move one square in any direction**, and under special circumstances has a special move called castling. It may capture any opposing piece that is within one space of it in any direction. If the **king is being threatened by an enemy piece it must be moved to safety or protected** by another piece. If there is **no way to save the king the opponent wins** the game.

  - ❑ **Knight** - Each player has **two knights** to start the game. They are located respectively on the **first and eighth row on the players own side of the board, two spaces away from the corners**, in between the rooks and the bishops. The knight **moves in an L shape** such that it moves two spaces horizontally or vertically followed by one space horizontally or vertically, but may not move in a straight line, such that if it moves two spaces vertically it must then move one space horizontally. It may capture any opposing piece that is on a tile it may move to. Additionally the knight **can leap over other pieces**. It is typically known in value as being worth **three points**, although this is for reference as the game is not based on points.

  - ❑ **Pawn -** Each player has **eight pawns** to start the game. They are **located respectively side by side along the second and seventh row** on the players own side. Pawns can **move either two spaces forward or one space forward** from its initial position. **After its initial move, it can only move one space forward** unless it is a capture move. On capturing moves it may either **capture an opposing piece one space diagonally** and in front of it or under special circumstances it may perform an en passant. Typically known in value as being worth **one point**, although this is for reference as the game is not based on points.

- ❏ **Queen** - Each player has **one queen**, initially located on the **first and eighth row** of either team's side of the board. It starts **next to the king**. For the starting side, it is located to the left of the king on the 4th space from the left side of the board. The opposing team's queen lies on the same column directly on the opposite side of the board. queens **can move diagonally, horizontally, and vertically** any distance up to another piece. **It may not "jump" over any piece**. Each player gets only one queen. Typically known in value as being worth **nine points**, although this is for reference as the game is not based on points.

- ❏ **Rook -** Each player has **two rooks** to start the game. They are located respectively on the **first and eighth row on the players own side on the corners** next to the knights. Rooks **can move horizontally and vertically** on the board, along with the ability to be part of castling under certain circumstances. Typically known in value as being worth **five points**, although this is for reference as the game is not based on points.

- ❏ **Special moves:**
  - ❏ **Castle** - A special move in which under certain circumstances, the **king may move two spaces**. Instead of his normal one space toward either rook, have the rook move to the opposite side one space over from the king. This may only be done if:
    - ❏ The **king has not been moved** during the game.
    - ❏ The intended **rook has not been moved** during the game.
    - ❏ There are **no pieces in between** the king and the rook.
    - ❏ The king is **not in check or checkmate**.
    - ❏ The rook is **not being threatened** by an opposing piece.
    - ❏ The **tiles in between the rook are not being threatened** by any piece of the opposing team.
  - ❏ **En Passant** - A special pawn capture move which happens **after one player moves a pawn two spaces forward** from its initial position. On the next turn, **an enemy pawn** located **on the same row** next to the moved pawn can then **"pass" through** by moving **diagonally one forward**. Such passing results in the pawn being captured as well. This is intended so a pawn may not get out of being captured by moving two spaces, so it is treated as though it moved only one.

- ❑ **Rules and Board States:**
    - ❑ **Check** - This occurs when the **king is being threatened** by a piece of the opposing team and has at least one possible move to get out of being threatened. The player in check **must make a legal move to get out** of check during his turn. Any other move that doesn't stop the king from being in check is invalid.

    - ❑ **Checkmate -** This occurs when the **king is being threatened** by a piece of the opposing team and has **no possible moves** to get out of being threatened. Thus, the game is over and the **player in checkmate loses the game**.

    - ❑ **Draw** - When a draw occurs the game being played ends and neither player is deemed the winner. This may happen under 4 circumstances.
        - ❑ Both players agree to call a draw and end the game.
        - ❑ There is a stalemate.
        - ❑ The fifty-move rule is invoked.
        - ❑ There is a lack of mating material.

    - ❑ **Fifty-Move Rule -** If neither a pawn has been moved nor a capture has been made in the last 50 moves. Either player can declare the match a draw

    - ❑ **Lack of Mating Material** - This results in a draw and occurs when both sides are left with one of the following.
        - ❑ Only a king and one knight
        - ❑ Only a king and one bishop
        - ❑ Only king

    - ❑ **Stalemate -** If a player has **no legal moves** but is **not in check** then there is a stalemate and the game is declared a draw.

# Software Architecture Overview

## 1.1 Main Data Types and Structures

- **Player:** Is a typedef struct that has all the information on a player. It **contains a list of Pieces** that are captured by a player, information on the **player color**, and indexes of **where the player's pieces are** on the board.
- **Piece:** Is **an unsigned 16 bit integer** that **stores what a certain piece** is. Each specific type of piece is given a **specific ascii char**, Ex. PAWN = ‘P’. Using ASCII every char has a **specific numerical value** and it **helps differentiate between pieces.** This value was also **shifted 1 left to store the color of the piece**. Ex. Least significant bit is 1 so it is white.
- **Board:** Grid will be used to implement the board. Grid is made of a **1-D Array of size 64 filled with Pieces**
- **Gamedata:** Is a struct that **holds multiple other structs** and other **necessary information** about the game. It stores information such as the players and certain player flags along with the settings struct.
- **Move:** A struct that contains move details including the starting index, the end index, and the specific piece that was moved. Ex. WPAWN
- **MoveList:** Another important structure list **possible moves** a piece can take. It is useful when **testing out temporary moves** in AI or other functions like check.
- **Stack:** A form of double linked list that **stores the moves made.** It behaves like a stack because each stack has a **head** and **tail move**. Using functions like **PushMove** and **PopMov**e we can **remove or add a recent move.**
- **Settings:** Store the settings for the game such as **gamemode, piece color** etc.
- **Status:** A structure that **contains the state of the game** (in progress, black wins, white wins, tie, black quits, white quits and stalemate).
- **AI:** Is a structure that holds the difficulty level of the AI and the AI’s player structure

**Main Data types used:**

- Struct
- Enum
- Grid
- Stack
- Index (a typedef as uint_fast8_t)

## 1.2 Major Software Components

• Diagram of module hierarchy

**Ultimate Chess Module Hierarchy**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | **Makefile** Makefile | | | | | | |
| | | | | | **Main** main.c main.h | | | | | | |
| **Game** Game.c Game.h | **Board** Board.c Board.h | **Player** Player.c Player.h | **GameData** GameData.c GameData.h | **Gameplay** Gameplay.c Gameplay.h | **Gamedata** Gamedata.c Gamedata.h | **Settings** Settings.c Setting s.h | **AI** AI.c AI.h | **MoveValidation** MoveValidation.c MoveValidation.h | **AIGameplay** AIGameplay.c AIGameplay.h | **Moves** Moves.c Moves.h | **Menu** Menu.c Menu.h |

- ❏ **Board.c:** This module contains functions for the chess board which include **printing the board** and managing the pieces
- ❏ **Player.c:** Contains player function including function for **adding/removing captured piece** by a player
- ❏ **Setting.c: Player Settings** which may include AI modes
- ❏ **AI.c:** Contains code for **AI implementation**
- ❏ **AIGameplay.c:** Contains code for the **AI algorithm**
- ❏ **Moves.c:** Contains functions for **creating moves** and utilities for **editing the move stack**
- ❏ **Main.c: Software code**, implements all functions from other modules
- ❏ **Menu.c:** Holds the implementation of the **menu functions** to print and decide settings
- ❏ **Game.c:** Holds all of the **actual game modes**
- ❏ **GameData.c:** This module contains utility functions that give information about the **current state of the game**
- ❏ **MoveValidation.c:** This module contains functions for checking **if moves are valid**
- ❏ **MoveList.c:** This module contains functions that **generate lists of valid moves** for each piece

- ❏ **GamePlay.c:** Contains all functions for gameplay including **making regular moves** and **special moves.**
- ❏ **Makefile: Compiles all of the code** and create executable for game

## 1.3 Module Interfaces

• API of major module functions

**Board.c:**

- ❏ **inline Piece GetPiece(const Board* const board, const Index index):**
  Returns Piece at a specific index on the board

**MoveValidation.c:**

- ❏ **bool IsValidMove(const Board* const board, const Color color, Index from, Index to):**
  Returns a boolean value to make sure if move is valid based on the index of the move (from and to) and the player based on color.

**Moves.c**

- ❏ **void PushMove(MoveStack* const stack, Move* const move):**
  Push move made to stack (records move made).

**Gameplay.c**

- ❏ **Move MakeMove(GameData* const data, Player* const player, const Move move)**
  takes in the current players move and if a piece is captured adds it to the captured list and then completes the move

**MoveList.c**

- ❏ **uint8_t GenerateMoveList(const GameData* const data, MoveList* const list, Player* const player, const Piece piece)**
  Generates a list of all available moves for every piece for the specified player.

**Player.c**

- ❏ **void SetDefaultPieceLoc(Player* const player, const bool isWhite)**
  Sets all pieces to their default locations

**Game.c**

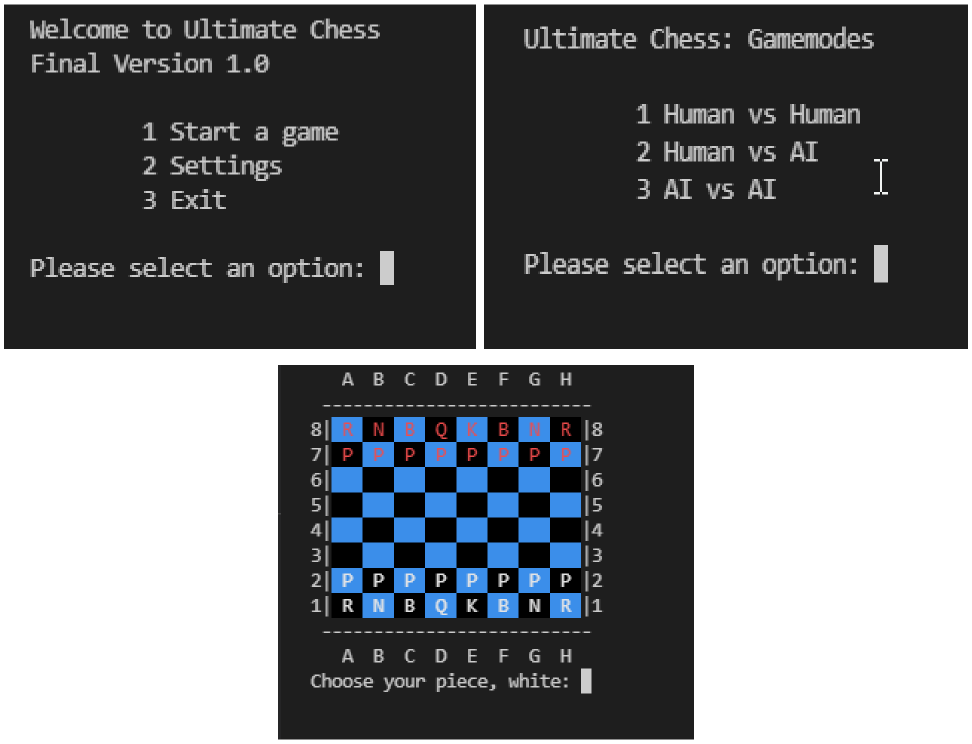
- ❏ **void Game(GameData* const data)**
  Contains function calls and necessary code to run all game modes

**AIGameplay.c**

- ❏ Contains the functions for the AI’s gameplay and move choices.

## 1.4 Overall Program Control Flow

The program will start out in the main menu where the user has the option to start a game, enter the settings menu or exit the program. If the user selects the settings menu they can change board and piece colors, if they choose to start a game they can pick a game mode and choose which side they want to play. During the game the player can exit the program at any time by typing "QQ." When the game finishes players can return to the main menu and will have the option to download a log of the game.



```
Welcome to Ultimate Chess
Final Version 1.0

        1 Start a game
        2 Settings
        3 Exit

Please select an option:
```

```
Ultimate Chess: Gamemodes

        1 Human vs Human
        2 Human vs AI
        3 AI vs AI

Please select an option:
```

```
    A  B  C  D  E  F  G  H
  -------------------------
 8| R  N  B  Q  K  B  N  R |8
 7| P  P  P  P  P  P  P  P |7
 6|                        |6
 5|                        |5
 4|                        |4
 3|                        |3
 2| P  P  P  P  P  P  P  P |2
 1| R  N  B  Q  K  B  N  R |1
  -------------------------
    A  B  C  D  E  F  G  H
 Choose your piece, white:
```

*Typical gameflow for a game bootup*

# Installation

## 2.1 System Requirements, Compatibility

Computer or Server running **Linux CentOS 6.10**
Minimum Requirements:
Minimum Disk Space/Recommended **- 1 GB / 5 GB**
Minimum Memory Requirement - **6273MB**

For More On Linux CentOS 6 Requirements:
https://wiki.centos.org/About/Product

## 2.2 Setup and Configuration

**Download all files** from Github repository for the chess game. Make sure to have putty or some alternative installed.

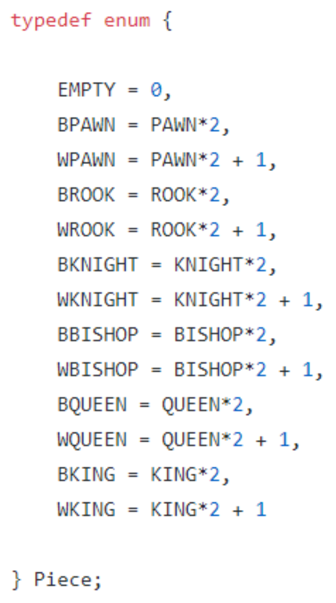
## 2.3 Building, Compilation, Installation

Compilation is done through the Makefile. The "make" command will compile all the necessary object files, libraries, and compile the executable game. **Run "make"** in the Linux command line and then **run ./bin/Ultimate_Chess** to begin playing. To run the test, build a debug version and run ./UltimateChess --test.

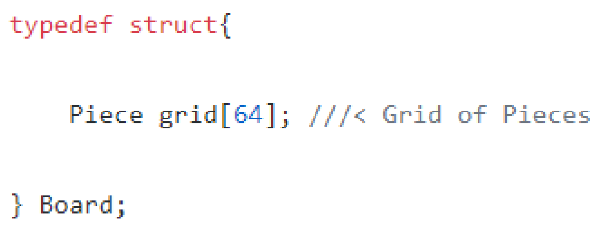
# Documentation of packages, modules, interfaces

## 3.1 Detailed Description of Data Structures

- **Main data structures used** are linked lists, trees and arrays, below are critical snippets of source code

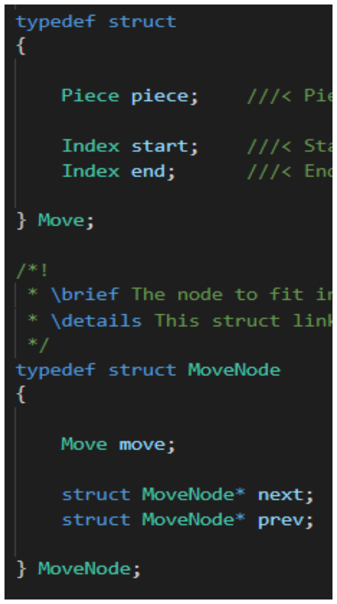


```c
typedef enum {

    EMPTY = 0,
    BPAWN = PAWN*2,
    WPAWN = PAWN*2 + 1,
    BROOK = ROOK*2,
    WROOK = ROOK*2 + 1,
    BKNIGHT = KNIGHT*2,
    WKNIGHT = KNIGHT*2 + 1,
    BBISHOP = BISHOP*2,
    WBISHOP = BISHOP*2 + 1,
    BQUEEN = QUEEN*2,
    WQUEEN = QUEEN*2 + 1,
    BKING = KING*2,
    WKING = KING*2 + 1

} Piece;
```



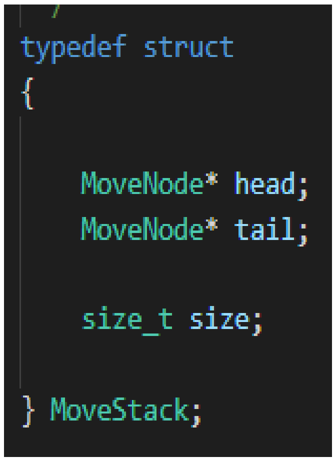
```c
typedef struct{

    Piece grid[64]; ///< Grid of Pieces

} Board;
```

*Figure 2:Piece enum for specific chess pieces*    *Figure 3: Board with array of pieces*



```c
typedef struct
{

    Piece piece;     ///< Pie

    Index start;     ///< Sta
    Index end;       ///< End

} Move;

/*!
 * \brief The node to fit ir
 * \details This struct linl
 */
typedef struct MoveNode
{

    Move move;

    struct MoveNode* next;
    struct MoveNode* prev;

} MoveNode;
```

*Figure 4:MoveNode And struct Move*



```c
 /
typedef struct
{

    MoveNode* head;
    MoveNode* tail;

    size_t size;

} MoveStack;
```

*Figure 5: Movestack linked list*

## 3.2 Detailed Description of Functions and Parameters • Function Prototypes and Brief Explanation

- ❏ **GetGameStatus:** Takes in board, **returns enum states** whether a player is in check it is a draw by **stalemate**, **limiting pieces**, or **50 move limit**, or there is a **checkmate**
- ❏ **GetPiece:** Returns **piece type**. Takes in two parameters which are the board and index.
- ❏ **IsValidMove:** Takes in player number, index of a piece and index of where to move, **checks if it is a valid mov**e, returns bool true if allowed false if not
- ❏ **MakeMove:** Takes in board array, player number, index of a piece and index of where to move, moves the piece no return
- ❏ **PrintBoard:** Takes in board array as a parameter and **prints out the board** nothing returned
- ❏ **ResetBoard: Resets the board** and its pieces positions on board.
- ❏ **SetPiece: Sets a piece to a certain index** on the board. Takes in 3 parameters which are the board, index and piece.
- ❏ **Game:** Contains the menu and **function calls** for playing **Human v Human, Human v AI, and AI v AI**
- ❏ **Menu:** This function contains all of the menu options and the function calls for print menu
- ❏ **GenerateMoveList:** This function contains the function calls for generating moves for each piece and forms a single list of all available moves for the given player.

## 3.3 Detailed Description of Input and Output Formats

**User Interface:**

- ❏ Movement: Users will have the ability to input where they wish to move. Input is in the following format - [Game Piece] [Starting position] [Ending Position]. Will return false if move is not valid.

**Output:**

- ❏ Log recording: Game log for each move will be in the following format - [Piece color] [Piece Type] from: [Start position] To: [End Position]

# Development plan and timeline

## 4.1 Partitioning of Tasks

**Coding:**

Set up headers for all of the major functions, start all the major functions written in the software components by the 19th
Finish board operation functions by 25th
Finish AI modules and hints by 30th
Finish Special moves and Gui by 5th

## 4.2 Team Member Responsibilities

**Chris Rodriguez:** Presenter
Work on setting header and source file.
Present every week what the team accomplished and what each member is doing.

**Raymond Duong:** Recorder
Work on the Gameplay module and Player module.
Record and take notes on team ideas.

**Ian Poremba:** Manager
Finish board module
Oversee that tasks are getting worked on and finished on time.

**Sunshine Jennings:** Reflector
Work on the Settings module and help with other large modules depending on how long it takes.

**Orion Serup:** Reflector
Responsibilities: Work on AI module

## Back matter

**Copyright**

**References:**

EECS 22/22L lecture material
CentOS: https://wiki.centos.org/About/Product
Stack Overflow: https://stackoverflow.com/
Current Authors: Raymond Duong, Sunshine Jennings, Orion Serup, Chris Rodriguez, Ian Poremba

# Index