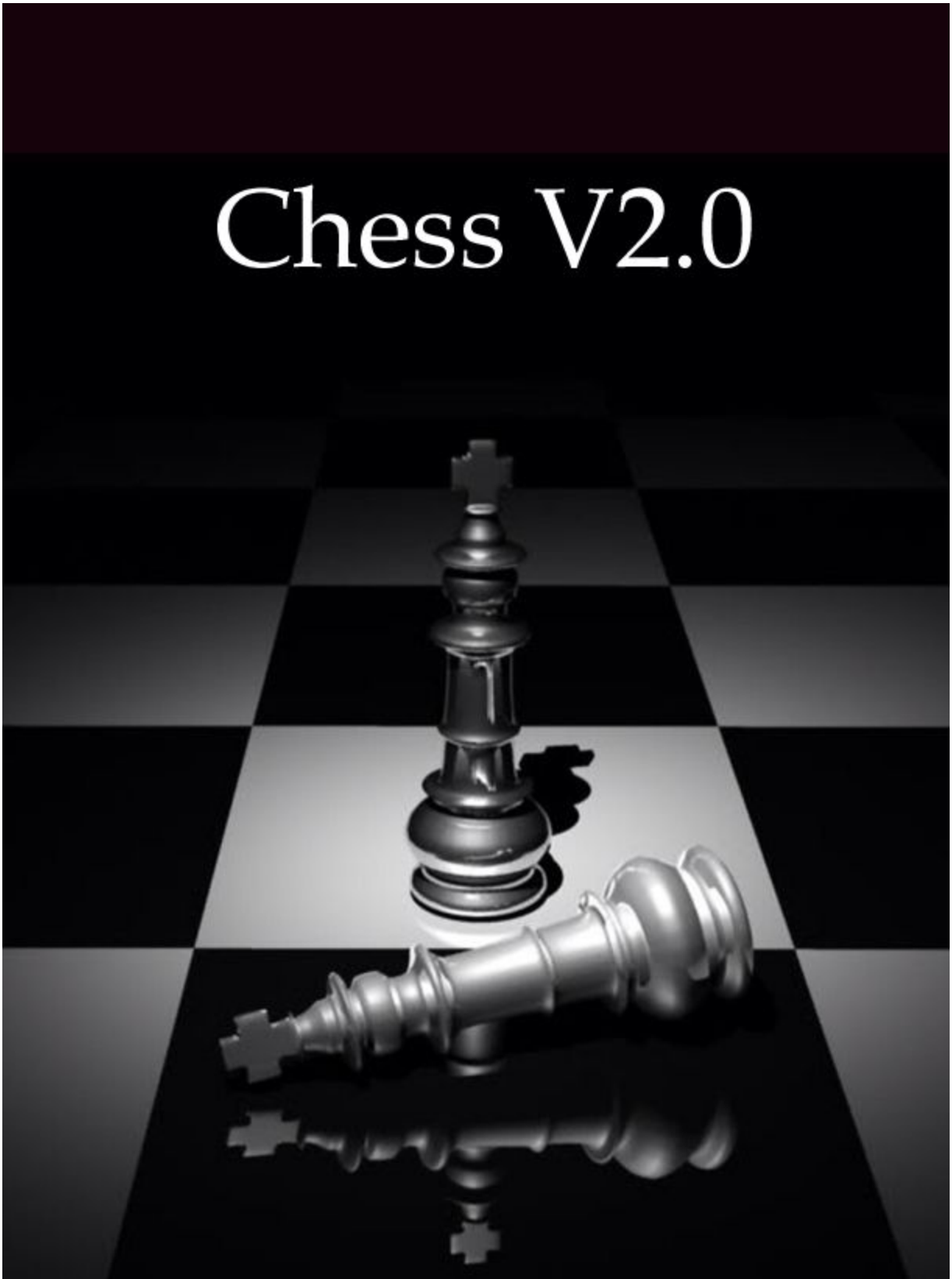
Chess V2.0



Breakdown Tech
Renetta Nelson, Chris Fernando, Giselle Meza, Joey Ortiz,
Brandon Schnedar, Ethan Groenow
Henry Samueli School of Engineering

Table of Contents

1. Glossary

Square: A data structure that acts as one square on the chessboard. There are a total of 64 of them each being either black or white.

Log file: A outside .txt file that will act as the backup storage for the game. Hitting the undo button will call the log file and return last moved piece to its former place.

PIECES: A data structure that will hold the information regarding what type and color a piece is.

Makefile: A script that will produce the executable final program. It makes it easier to compile the program by doing all the necessary compilations with one command (make).

Math Library: A library in C that contains math functions.

SDL Library: A graphics library in C that contains the functions necessary to build a GUI

2. Software Architecture Overview

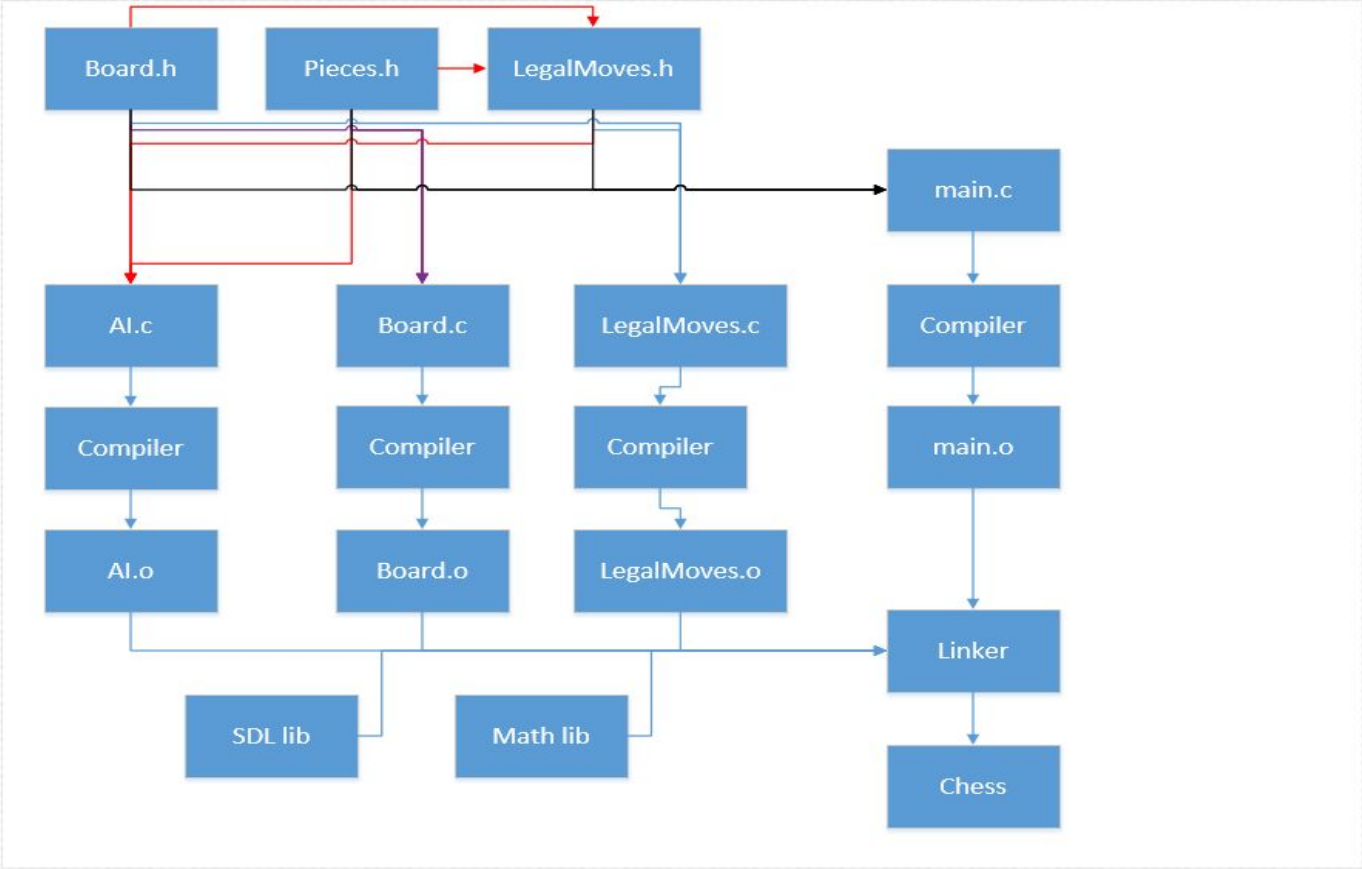
2.1 Main data types and structures

- Pieces
  - Int piece_type (King, Queen, Knight,etc)
  - Int Color (Black and White)
  - Int Value
  - Int Past_move

- Squares
  - PIECE *piece
  - Int color
  - Int x,y

- Board
  - SQUARES *board[8][8]

2.2 Major software components



2.3 Module interfaces

- Based on the inputted coordinates from the user, the “main” module checks the pieces associated with those coordinates and calls the appropriate piece functions

2.4 Overall program control flow

- Setup:
    - Display game settings menu
        - Game Settings (Player vs. AI, Player vs. Player, etc.)
    - Display difficulty menu
        - If Player vs. AI, user input for AI difficulty
        - If AI vs AI, user input for both difficulties

- Display Board:
    - User input for choosing a side (Black or White)
    - If Player vs. Player, “Player 1” chooses side and “Player 2” is assigned the remaining side
- Players makes moves (Player vs. AI):
    - White player moves first (If player chooses black, AI moves first before user chooses move)
    - If black is in checkmate, display board and message “white wins”, otherwise black player makes a move
    - If white is in checkmate, display board and message “black wins”, otherwise white player makes a move
    - Everytime the player makes a move, unless the move leaves the other in checkmate, the AI will move before displaying the board again.
    - At any time on the player’s move they can click the “undo” option to go back a turn. If no moves had been made prior, an error message will be displayed.
- Players make moves (Player vs. Player):
    - Same as Player vs. AI except after every move the players make, the board is displayed and the message switches back and forth between “white player’s move” and “black player’s move”
- AI vs. AI
    - Display board for every move the AI makes. No user input required
- End of Game:
    - Once one side is put into checkmate and the victory message is displayed, the player is given the option to “undo” or “exit” the game.

3. Installation

3.1 System Requirements

- ❏ The system requirements necessary to run Chess v 1.0 smoothly are:
    - ❏ 2GHz dual core processor
    - ❏ 2GB RAM
    - ❏ 25 GB hard drive space
    - ❏ Linux operating system available

- ❏ With these requirements in mind while programming, we as a team must be wary of the capabilities of the system and not try to overload the computer. For example, it would take way too long to search through every single possible game because after three turns, there are over 121 million possible games that could unfold in chess. The game must run under the defined system requirements.

3.2 Setup and Configuration

- To setup and configure Chess v 2.0
    1. Ensure that you have all the necessary tar package
    2. To extract the files type “gtar xvzf *package*.tar.gz” (the package name will either be Chess_V1.0 if you are downloading the user version, or Chess_V1.0_src if you downloading the source code version)
    3. Type in “make” in the linux command line
    4. Run the program by typing “./Chess” in the command line
    5. Optional: Add debugging support in the Makefile and source files

3.3 Building, Compilation, Installation

- To build the program, a makefile is necessary to create the executable due to the large size of the program. The structure of the Makefile is dependent on how the final chess program will be put together. Our Chess program will depend on four source files, Board.o, AI.o, LegalMoves.o, and Main.o. Each source file is compiled into an object file that is then linked at the very end to the Chess program. This file will be executable and can be ran with ./chess in the linux command line. Below is a temporary template of what our makefile will look like.

```
#Definitions
CC= gcc
DEBUG=  -g -DDEBUG
CFLAGS= -ansi -std=c99  -Wall -c
LFLAGS= -Wall

#############################################################################

#Object Files


#Target for Board.o
Board.o: Board.h Pieces.h Board.c
        $(CC) $(CFLAGS) Board.c -o Board.o


#Target for LegalMoves.o
LeagalMoves.o: Pieces.h Board.h LegalMoves.h LegalMoves.c
        $(CC) $(CFLAGS) LegalMoves.c -o LegalMoves.o

#Target for AI.o
AI.o: Board.h Pieces.h LegalMoves.h AI.h AI.c
        $(CC) $(CFLAGS) AI.c -o AI.o


#Target for Main.o
Main.o: Board.h Pieces.h LegalMoves.h AI.h  Main.c
        $(CC) $(CFLAGS) Main.c -o Main.o

Old_main.o: Board.h Pieces.h LegalMoves.h AI.h  Old_main.c
        $(CC) $(CFLAGS)   Old_main.c -o Old_main.o

#############################################################################

#Executable Files
Chess: Main.o LegalMoves.o Board.o
        $(CC) $(LFLAGS) -lm  Main.o LegalMoves.o Board.o -o Chess
        cp Chess /users/ugrad2/2017/winter/team13/chess/bin

Old: Old_main.o LegalMoves.o AI.o  Board.o
        $(CC) $(LFLAGS) -lm Old_main.o AI.o  LegalMoves.o Board.o -o Old

OTest: Old_main.o LegalMoves.o AI.o  Board.o
```

4. Documentation of packages, modules, interfaces

4.1 Data Structures

- Board
  - Sample Code:

```
typedef struct squares{
        PIECE *piece;
        int color;
        int x,y;
}SQUARES;

typedef struct board{
        SQUARES board[8][8];
}GAME;
```

  - Occupied - square will point to a piece if occupied
    - Unoccupied - it will be NULL pointer
  - Coordinates - 0 to 63
    - GUI representation with be standard A1 to h8 as it interprets each square integer
- Pieces
  - Sample Code:

```
typedef struct piece{
    int color;
    short piece_type;
    int past_move;
    int value;
 }PIECE
```

  - int color
    - Odd - White
    - Even - Black
  - int color also defines the type of piece
    - King = 12 (White King)

- King = 11 (Black King)
- Queen = 10 (White Queen)
- Queen = 9 (Black Queen)
- Bishop = 8 (White Knight)
- Bishop = 7 (Black Knight)
- Knight = 6 (White Knight)
- Knight = 5 (Black Knight)
- Rook = 4 (White Rook)
- Rook = 3 (Black Rook)
- Pawn = 2 (White Pawn)
- Pawn = 1 (Black Pawn)

4.2 Functions and parameters

- GetPiece(SQUARES board[8][8])
  - This function will retrieve the piece that the user typed in. It will return a PIECE pointer.
- Get_Current_Position(SQUARES board[8][8])
  - During the player's turn, it will ask the user to type in where they would like to move. This will then return a pointer to the SQUARE that they chose. For beta, we may ask the user to type where they would like to the move the piece instead of clicking.
- Check(PIECE *king ,SQUARES *board)
  - This function will check if the king is in check. If the king is in check, it will alert the player that their king is vulnerable to a direct attack. It will also check for checkmate. If there is no possible way to get out of check, the function will declare a checkmate and the game will end.
- Legal move functions
  - Qmove(SQUARES * curr_position, SQUARES *destination, GAME game)
  - Kmove(SQUARES *curr_position, SQUARES *destination, GAME game)
  - Hmove(SQUARES *curr_position, SQUARES *destination)

    - Bmove(SQUARES *curr_position, SQUARES *destination, GAME game)
    - Pmove(SQUARES *curr_position, SQUARES *destination, GAME game),
    - Rmove(SQUARES *curr_position, SQUARES *destination, GAME game)
        - These functions will check if the move is legal for the piece and if it is legal, it will move the piece and update the board
- Board Functions
    - log_move(FILE *file, SQUARES *curr_position, SQUARES *destination)
    - undo()
    - SetupGame(GAME game)
        - Sets up the beginning board
    - UpdateBoard(SQUARES *board)
        - After a move has taken place, the board will be updated and redrawn

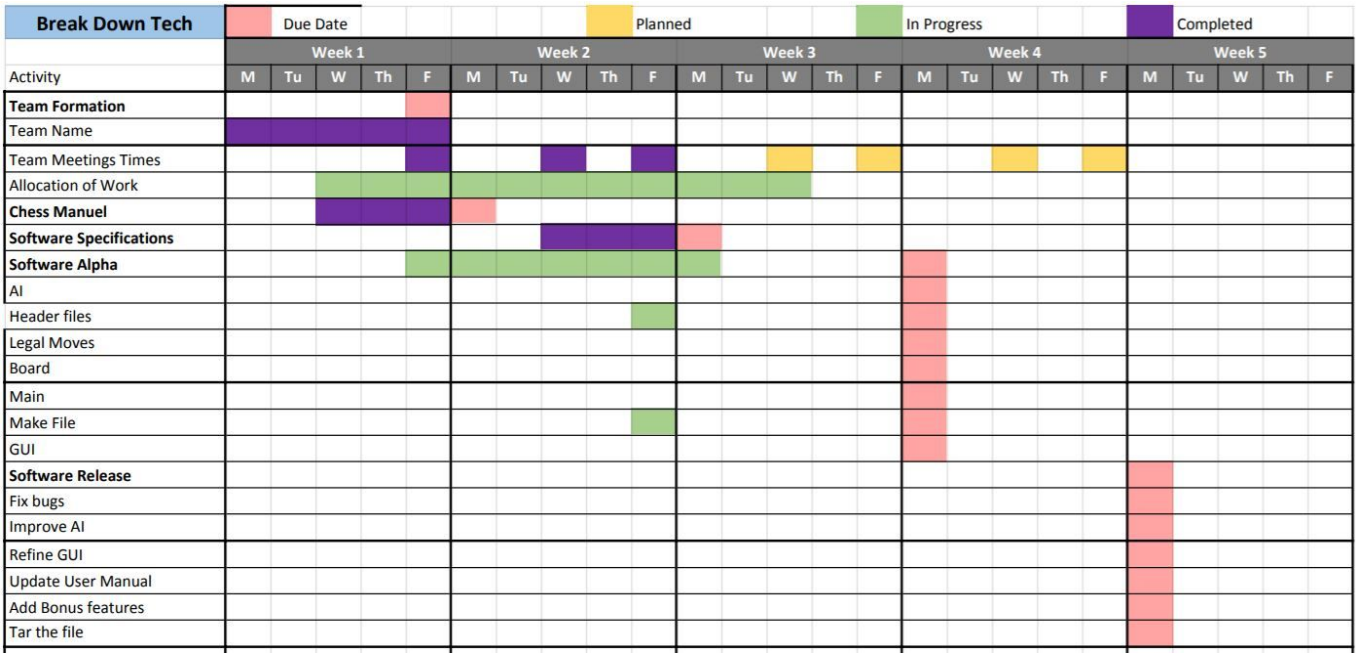
4.3 Input and output format

- User Input
    - Typed Input
        - Piece selection
        - Game settings
        - To square destination
            - Type in coordinates of piece and desired destination
        - End game menu
- Output
    - Error messages
    - Images (Board, pieces, etc)
    - Log file
        - Text file

- Logs piece moved from original and new coordinates
- Winner notification
  - "Player (Black or White) has won!"

5. Development plan and timeline

5.1 Gantt Chart



5.2. Team Responsibilities

- ❑ A.I.c: Brandon
- ❑ Board.c: Chris & Brandon & Ethan
- ❑ LegalMoves.c: Renetta & Giselle
- ❑ Main.c: Joey & Ethan & Chris
- ❑ Makefile: Ethan
- ❑ Pieces.h: Joey
- ❑ Board.h: Joey & Ethan
- ❑ LegalMoves.h: Renetta
- ❑ GUI: Chris

6. Copyright

7. References

Chess Burst, Team. “Illegal Moves.” *ChessBurst.com*, www.chessburst.com/chess-rules/illegal-moves.html.

“Chess Wallpaper.” *Http://Wallpaper-Gallery.net/Single/Chess-Wallpaper/Chess-Wallpaper-9.Html*.

Federation, US Chess. “Learn to Play Chess.” *US Chess*, USCF, www.uschess.org/content/view/7324/28/.

8. Index