Software Specifications

Producers:
KNIGHT—Daniel Ring
KNIGHT—Pino Cao
KNIGHT—Jiajun Liu
KNIGHT—Shahrooz Maghsoudi
KNIGHT—Gabriel Meneses Vieira
KNIGHT—Miao Yu
KNIGHT—Yinxue Li
KNIGHT—Ngor Kendrick Seav

V 1.0

Table of Contents

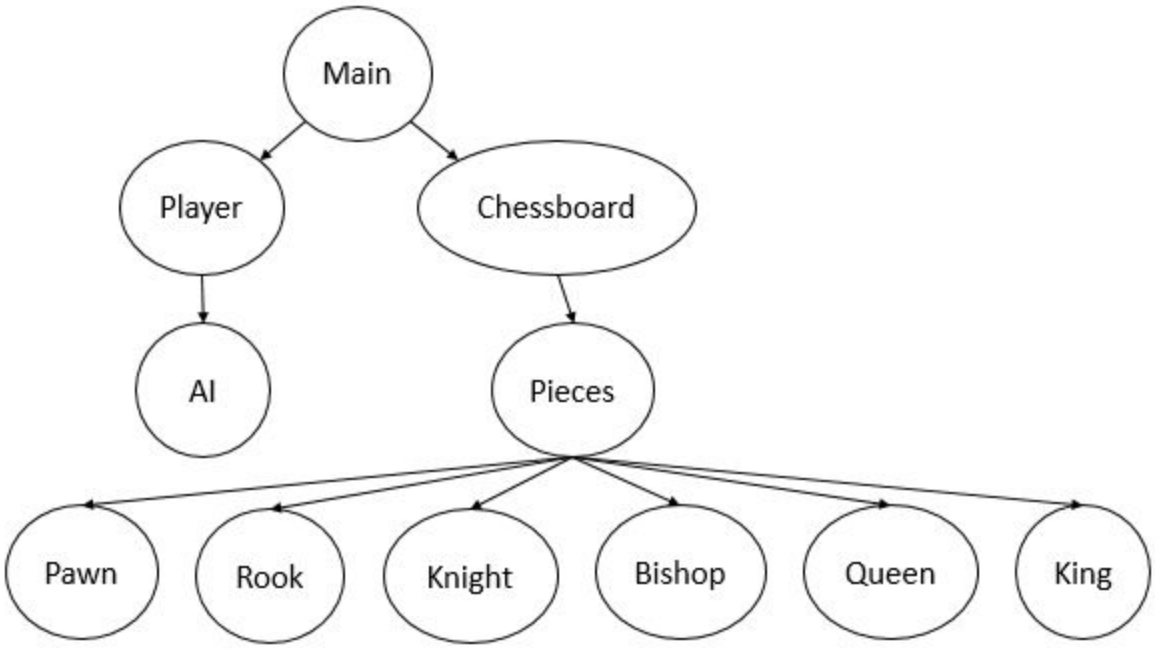
Software Architecture Overview

Main data types and structures

The main type of data structure used in this software is the Class data structure. Every module is a Class. The chessboard also contains a 2D array of pieces.

Major software components

Diagram of program structure



Explanation:

1. Different Pieces Module: All are dependent on Public function of Class Pieces:

```
    virtual bool checkMove(Piece[8][8] board, int fromRow, int
fromCol, int toRow, int toCol, bool threatened);
    virtual bool move(Piece[8][8] board, int fromRow, int fromCol,
int toRow, int toCol);
    virtual bool revertMove(Piece[8][8] board, int fromRow, int
fromCol, int toRow, int toCol);
    virtual bool checkmate(Piece[8][8] board);
```

2. Pieces:

```
Public:
virtual chess_t getType();
virtual char getDisplayChar();
virtual bool checkMove(Piece[8][8] board, int fromRow, int
fromCol, int toRow, int toCol, bool threatened);
virtual bool move(Piece[8][8] board, int fromRow, int fromCol,
int toRow, int toCol);
virtual bool revertMove(Piece[8][8] board, int fromRow, int
fromCol, int toRow, int toCol);
virtual bool checkmate(Piece[8][8] board);
(will be explained later)
Private:
chess_t type;
side_t side;
char display;
```

3. Chessboard: depend on Pieces module and function: *getwinner* [to determine whether the game is over]
4. Main: depends on the Player && AI Module, Chessboard Module

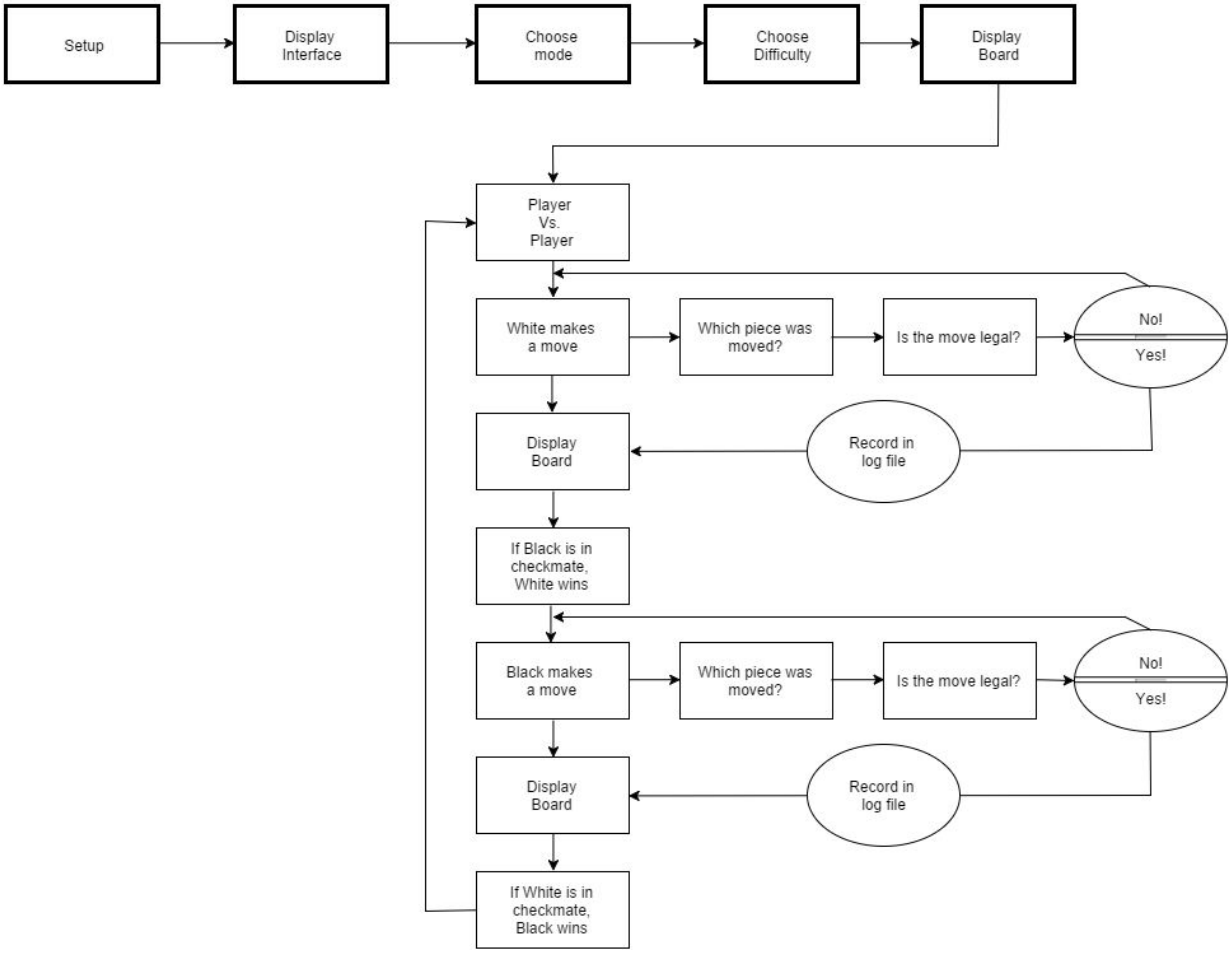
Module interfaces :

In the part of King, Queen, Pawn, etc, they all depend on <math.h>: Since we have calculate the movement of different pieces. To get the destination location of pieces, we need to use math library.
In the part of Chessboard, it depends on the <stdio.h>: since we use printf function to print out the chessboard.
In the part of Main, it depends on the <time.h>: Since we try to add the timer function for the chess program.

API of major module functions

No libraries outside C++ are included so far.

Overall program control flow



Installation

System requirements, compatibility

- Linux-based OS
- x86-compatible processor
- Minimum 4 MB of disk space
- Minimum 256 MB of RAM
- Monitor and keyboard

Setup and configuration

No setup or configuration is required to use the software after installation.

Building, compilation, installation

1. Open a terminal window
2. Use the command `cd` to navigate to the correct folder
3. Extract the source code from the archive with the command
   `tar xzf Chess_V1.0.tar.gz`
4. Build the software with the `make` command.
5. To begin a game, run the `chess` executable in the generated “bin” folder.

Documentation of packages, modules, interfaces

Detailed description of data structures

All the data structures used are classes. The classes are listed below:

- AI
- bishop
- chessboard
- king
- knight
- pawn
- piece
- player
- queen
- rook
- types

The classes bishop, king, knight, pawn, queen and rook are subclasses of the piece class. AI class is subclass of player. Types holds an enumerated data structure for the piece types and chessboard holds the position of every piece.

Detailed description of functions and parameters

Function prototypes and brief explanation

```
void AI::getMove(ChessBoard &board, int *fromRow, int *fromCol, int
*toRow, int *toCol)
```

Make the AI choose a movement on the board

```
bool Bishop::checkMove(ChessBoard &board, int fromRow, int fromCol,
int toRow, int toCol)
```

Implements the rules of movement for the Bishop

```
void ChessBoard::display()
```

Display the chessboard on the screen

```
Piece* ChessBoard::getPiece(int row, int col)
```

Return the piece on the selected square

```
side_t ChessBoard::getWinner()
```

Determine if the game is over

```
bool ChessBoard::checkMove(side_t side, int fromRow, int fromCol, int
toRow, int toCol, bool displayErrors /* = false */)
```

Check if a move is possible

```
bool ChessBoard::move(int fromRow, int fromCol, int toRow, int toCol)
```

Move a piece, also responsible for part of the log

```
void ChessBoard::swap(int fromRow, int fromCol, int toRow, int toCol)
```

Swap two pieces

```
void ChessBoard::promote(side_t side, int row, int col)
```

Handles the logic for pawn promotion

```
bool ChessBoard::isThreatened(int row, int col)
```

Determine if a square is threatened by another piece

```
void ChessBoard::availableMoves(int moves[8][8][8][8], side_t side)
```

Store a list of all possible moves in a 4D array

```
bool King::checkMove(ChessBoard &board, int fromRow, int fromCol, int
toRow, int toCol)
```

Implements the rules of movements for the king, including castling

```
bool King::move(ChessBoard &board, int fromRow, int fromCol, int
toRow, int toCol)
```

Also implements the rules of movements for the king, including castling

```
bool King::checkmate(ChessBoard &board, int row, int col)
```

Handle checkmate logic.

```
bool Knight::checkMove(ChessBoard &board, int fromRow, int fromCol,
int toRow, int toCol)
```

Implements the rules of movements for the knight.

```
bool Pawn::checkMove(ChessBoard &board, int fromRow, int fromCol, int
toRow, int toCol)
```

Implements the rules of movements for the pawn, including en passant.

```
bool Pawn::move(ChessBoard &board, int fromRow, int fromCol, int
toRow, int toCol)
```

`Also` implements the rules of movements for the pawn, including en passant.

```
bool Pawn::enpassantCheck(ChessBoard &board, int fromRow, int
fromCol, int toRow, int toCol)
```

Check for en passant

```
chess_t Piece::getType()
```

Return type of chess piece

```
side_t Piece::getSide()
```

Return the side the piece is on

```
char Piece::getDisplayChar()
```

Return character to display for chess piece

```
bool Piece::getMoved()
```

Check if the piece has moved

```
bool Piece::setMoved()
```

Set the piece moved flag, for use when moved by another piece

```
bool Piece::getCaptured()
```

Check if the piece was captured

```
void Piece::setCaptured()
```

Set the piece captured flag

```
bool Piece::checkMove(Piece[8][8] board, int fromRow, int fromCol,
int toRow, int toCol, bool threatened)
```

Check if a move is valid

`bool Piece::move(Piece[8][8] board, int fromRow, int fromCol, int toRow, int toCol)`

Make a move if it is valid

`void Player::getMove(ChessBoard &board, int *fromRow, int *fromCol, int *toRow, int *toCol)`

Get a movement from the player

`side_t Player::getSide()`

Return the side of the player

`void Player::setSide(side_t _side)`

Set the side flag of the player

`bool Queen::checkMove(ChessBoard &board, int fromRow, int fromCol, int toRow, int toCol)`

Implements the rules of movements for the ueen.

`bool Rook::checkMove(ChessBoard &board, int fromRow, int fromCol, int toRow, int toCol)`

Implements the rules of movements for the Rook.

Detailed description of input and output formats

Syntax/format of a move input by the user

The user inputs the beginning location of the piece to be moved to the desired location.
e.g “Enter a move (algebraic format; e.g. b2b3): a4a5”
These locations are then stored into the fromRow, fromCol, toRow, and toCol int variables that are declared in main.cpp.
The chessboard move function then identifies what piece is in the initial location, and checks if that move is possible

Syntax/format of a move recorded in the log file

The log file is stored in two columns in long algebraic notation in UTF-8. First column is movements made by the white pieces and the second is by the black. Each line record one turn

♟e4-e5 ♘f3-c6
♝b5-a6 ♖a1-a3

Development plan and timeline

Partitioning of tasks

Task 1: Basic structure of the chess project
Task 2: Modules for the movement of different pieces
Task 3: AI
Task 4: Log file

Projected timeline

	Week 1 (1/11/16)	Week 2 (1/18/16)	Week 3 (1/25/16)	Week 4 (2/1/16)	Week 5 (2/3/16)
Basic Structure	X	✓	✓	✓	✓
Modules	X	X	✓	✓	✓
AI	X	X	✓	✓	✓
GUI	X	X	X	X	X
Timer	X	X	X	X	X
Log File	X	X	X	✓	✓

Team member responsibilities

Daniel - Build the program structure, write the chessboard and several pieces, and design tests.
Pino & Yinxue Li & Jiajun Liu - Work on the algorithm for different pieces.
Miao & Shahrooz - Work on the AI component and strategy.
Shahrooz - Timer component to control how long each player takes to move.
Kendrick - Research and work on the GUIs (most likely SDL).
Gabriel - Work on implementing the log file into the game. Update the User Manual and the software specifications.

Back Matter

Copyright

Reference

"Chess: Knight Vs Pawn 02." *By Dmbarnham on DeviantArt*. Web. 12 January 2016. <http://dmbarnham.deviantart.com/art/Chess-Knight-Vs-Pawn-02-303022493>.