Software Specifications

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

V 1.0

Table of Contents

Software Architecture Overview ------------------ 3
    Main data types and structures
    Major software components
        Diagram of module hierarchy
    Module interfaces ------------------ 4
        API of major module functions
    Overall program control flow ------------------ 5
Installation
    System requirements, compatibility
    Setup and configuration ------------------ 6
    Building, compilation, installation
Documentation of packages, modules, interfaces
    Detailed description of data structures
        Critical snippets of source code
    Detailed description of functions and parameters ------------------ 7
        Function prototypes and brief explanation
    Detailed description of input and output formats
        Syntax/format of a move input by the user
        Syntax/format of a move recorded in the log file
Development plan and timeline ------------------ 8
    Partitioning of tasks
    Team member responsibilities
Back Matter ------------------ 9
    Copyright
    Reference
    Index

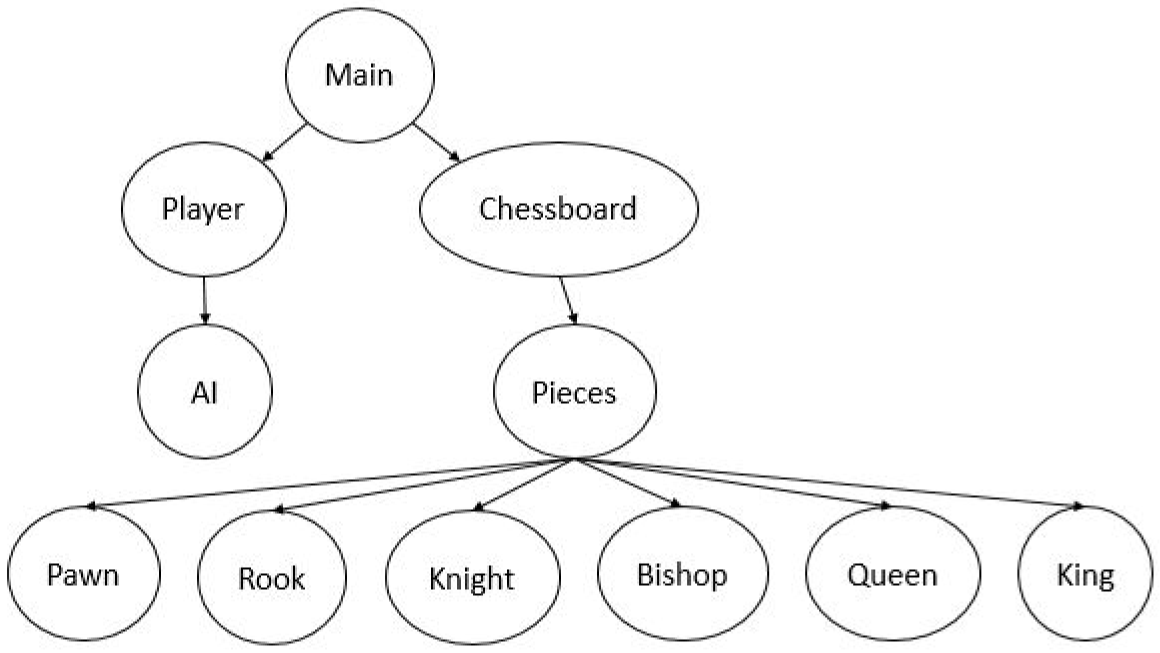
Software Architecture Overview

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 module hierarchy



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);
```

(These functions will be explained later)

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

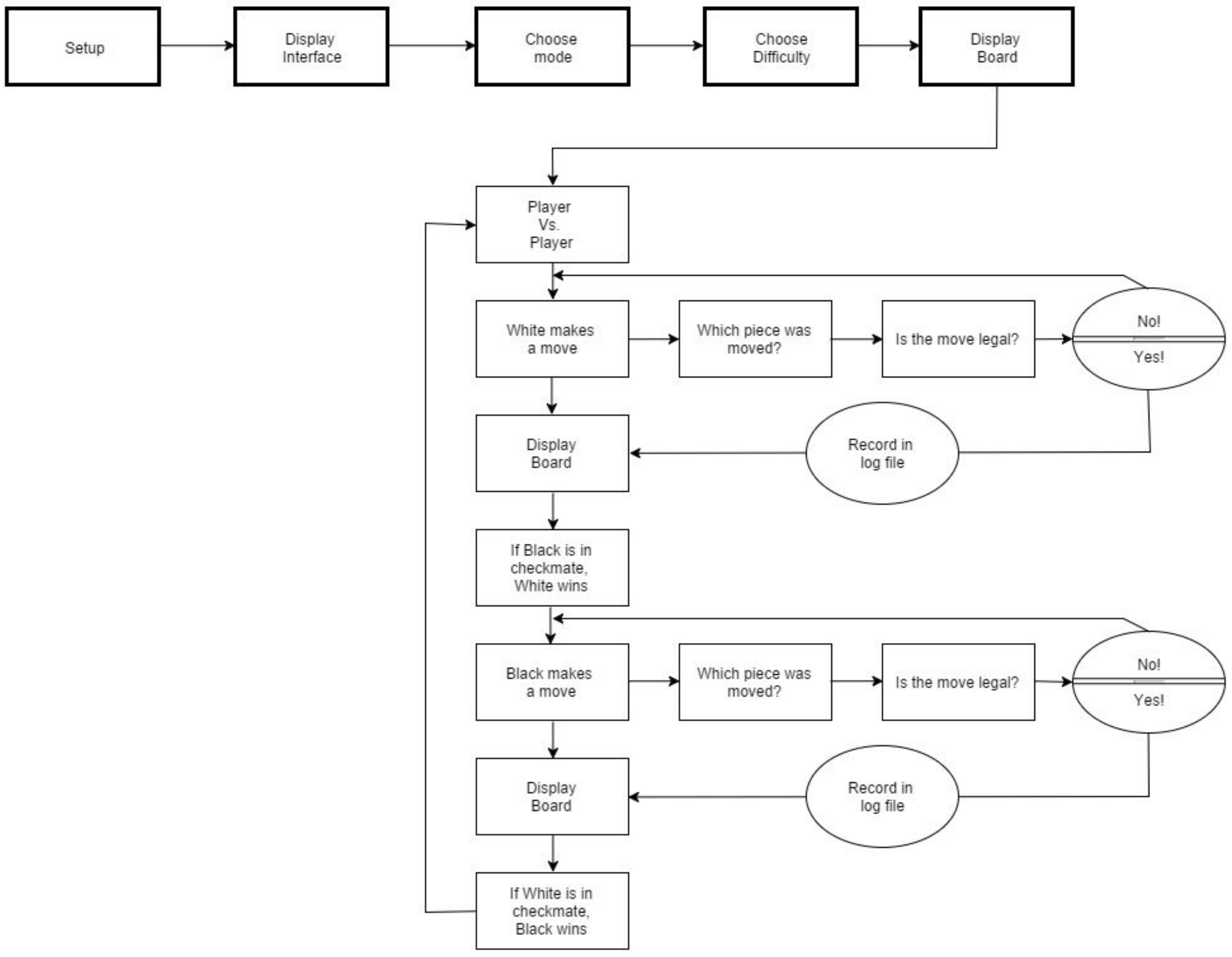
Module interfaces :

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 12 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 KnightPP-v1.0.tar.gz`
4. Build the software with the `make` command.
5. To begin a game, run the `KnightPP` executable in the generated “bin” folder.

Documentation of packages, modules, interfaces

Detailed description of data structures

Critical snippets of source code

Detailed description of functions and parameters

Function prototypes and brief explanation

```
void ChessBoard::display()
```
Display the chessboard on the screen

```
side_t ChessBoard::getWinner()
```
Determine if the game is over

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

```
bool ChessBoard::isThreatened(int row, int col)
```
Determine if a square is threatened by another piece

```
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::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

```
bool Piece::revertMove(Piece[8][8] board, int fromRow, int toRow, int
toCol)
```
Revert the previous move

```
side_t Piece::checkmate(Piece[8][8] board)
```
Check if the game is over

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 algebraic notation. First column is movements made by the white pieces and the second is by the black. The first letter represents the piece is moved (none for Pawn, R for the Rook, N for the Knight, B for the Bishop, Q for the Queen and K for the King) and the other two characters are the ending coordinates.
```
1.  e4  e5
2. Nf3 Nc6
3. Bb5  a6
```
In case of a Checkmate it will show `1 - 0` if the whites win and `0 - 1` if the blacks win.

Development plan and timeline

Partitioning of tasks

Task 1: Basic structure of the chess project (DONE)
Task 2: Modules for the movement of different pieces (Almost DONE)
Task 3: AI
Task 4: GUI
Task 5: Timer
Task 6: 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	X	✓	✓
GUI	X	X	X	✓	✓
Timer	X	X	X	✓	✓
Log File	X	X	X	✓	✓

Team member responsibilities

We finished the user's manual and developer's manual together.
In the following weeks:
Daniel - Improve the basic structure for this project in order to be more efficient.
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

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>.

Index

Modules P.3,4
Pieces P.4
Chessboard P.4
Main P.4
Control Flow P.5
Installation P.5
System Requirement P.5
Data structure P.6
Function prototypes P.6,7
User input P.7
Log file P.7
Tasks P.8
AI P.8
GUI P.8