Assignment #2

CS 225, Summer 2018

Due Date:	19th June 2018 2359hrs
Topics covered:	Type Erasure, Virtual Functions, Class Design
Deliverables:	To submit one zip file containing the files `Player.h`, `HumanPlayer.h`, `HumanPlayer.cpp`, `RandomPlayer.h`, `RandomPlayer.cpp` according to CS225 syllabus naming conventions for the zip file.
Compiler flags :	For gnu, it would be `-std=c++1z -Wall -Wextra -Werror -pedantic` For Visual Studio 2017 (Command line), it would be `/W4 /WX /std:c++latest /nologo /EHa`
Objectives:	To apply the Type Erasure pattern for polymorphic behaviour and use of the C++14 standard libraries e.g., `<random>`, `<regex>` etc.

1 Programming Statement: To extend the Chess Engine

In this assignment, the student expected to extend the Chess Engine in previous assignment. The current state of the Chess Engine in previous assignment does not encourage any testing or use of different Artificial Intelligence strategies when it comes to computer chess playing. We would like to design the classes such that the designed framework can allow different types of playing strategies to be tested.

2 `Player` class

In order to help the testing of different implementations of Artificial Intelligence, we make use of the `Player` class that defined as the following:

```
class Player
{
public:
   Move PlayMove(const Board&, Colour);
};
```

The `Player` class is an interface class that has a member function `PlayMove` takes in a `Board` object and `Colour` and returns a `Move` object depending on the current positions of the `Board` for the `Colour` passed in. For example, if we were to desire to know the next move for Black, we would call `PlayMove(board, BLACK)`.

3 Game class

Any derived class of the `Player` class would be used in a `Game` class object to test the different A.I. strategies of the `Player` class. For example, consider the following use-case:

```cpp
std::ifstream fs("default_board_state.txt");
Player p1 = HumanPlayer();
Player p2 = HumanPlayer();
Game g(p1, p2, WHITE, fs);
g.PlayGame(true);
return 0;
```

In this use-case, `HumanPlayer` is simply another class that also implements the interface `PlayMove` member function, which will described in section 4. The above use-case would allow two users to play a game of chess using the Chess Engine we developed in previous assignment. The implementation of the `Game` class is provided. We provide an explanation of the `PlayGame` member function here. The function prototype of the `PlayGame` member function is the following:

```cpp
Colour PlayGame(bool print = false);
```

Since the `Game` object is created by two `Player` objects, the `PlayGame` member function takes turns to call the `PlayMove` method of each `Player` object until one of the kings have been captured. Then, the `PlayGame` member function returns the winning `Colour`.

4 `HumanPlayer` class

The HumanPlayer class should start with the following partial class declaration:

```cpp
class HumanPlayer
{
public:
   HumanPlayer( std::istream& = std::cin,
                /*Shows the default input stream
                  for reading in input */
               bool print=true
               /*Human Player just reads input without
                 printing if print is false*/
             );
   Move PlayMove(const Board&, Colour c);
};
```

The `PlayMove` member function `HumanPlayer` class reads in input from the standard input and returns the move keyed in by the user, provided that the input is valid and the move is legal. Consider the following printout of the board which shows the labelled coordinates of the board.

```
  A B C D E F G H
 +-+-+-+-+-+-+-+-+
0|R|N|B|Q|K|B|N|R|
 +-+-+-+-+-+-+-+-+
1|P|P|P|P|P|P|P|P|
 +-+-+-+-+-+-+-+-+
2| | | | | | | | |
 +-+-+-+-+-+-+-+-+
3| | | | | | | | |
 +-+-+-+-+-+-+-+-+
4| | | | | | | | |
 +-+-+-+-+-+-+-+-+
5| | | | | | | | |
 +-+-+-+-+-+-+-+-+
6|p|p|p|p|p|p|p|p|
 +-+-+-+-+-+-+-+-+
7|r|n|b|q|k|b|n|r|
 +-+-+-+-+-+-+-+-+
```

The user is expected to key in the move in the following format:

```
(<col-letter><row-num>-<col-letter><row-num>)
```

As an example, a move of the knight in B7 may look like the following:

```
(B7-A5)
```

or

```
(B7-C5)
```

There are 3 categories of input from users:

- Invalid input. These are input that does not conform to the above format. They may look like `( B7-A5)`, for example, this is wrong because there's an additional space after the first bracket.
- Illegal moves. These are input that are valid, but illegal from Chess point of view. For example, `(D7-H4)` may be a valid input but illegal at the moment because the queen is blocked by some other piece. The best way to check if the move is legal is to see if the move can be found in the vector returned by `Board::GetAllLegalMoves`.
- Valid and Legal moves. These are input that are valid i.e., conform to the pattern of the input described above and they are also legal moves that conform to rules of chess.

A sample ChessEngine.exe is provided for you to test the effect of the sample inputs. The following show a sample of possible inputs from the default initial position of the Chess.

```
Your turn to play:
( A1-A3)
Invalid input. Please enter your move again.
(A1-A3)
Invalid move. Please enter another move.
```

The above shows the difference between the invalid input (because of the space between the "(" and "A") and illegal moves.

The basic algorithm of the HumanPlayer is as the following (in psuedo-code):

```
legal_moves = board.GetAllLegalMoves
while(true)
{
    print "Your turn to play:\n" if needed
    Read input from console.
    while(input is invalid)
        print "Invalid input. Please enter your move again.\n"
        Read input from console
    convert input from string into a Move object
    find a match for the move object in the legal_moves
    if match is found, return the Move object
    else print "Invalid move. Please enter another move.\n"
};
```

Detecting the validity of the input could be helped by using the C++11 regex library (not compulsory but a suggestion). Link can be found here.

5 RandomPlayer class

The RandomPlayer class should start with the following partial class declaration:

```cpp
class RandomPlayer
{
public:
    RandomPlayer(unsigned seed);
    /*

    */
   Move PlayMove(const Board & board, Colour c);
};
```

The RandomPlayer implements the PlayMove by obtaining the list of legal moves from the Board::GetAllLegalMoves(). The idea is to choose any one of the legal moves with equal probability (this is called uniform distribution).

C++11 provides good support for having your own pseudo random number generator (PRNG). In the <random> header file, we find implementation for PRNG such as `std::mt19937`. The student is recommended to use the above default seed to seed the `std::mt19937` object before using the `std::uniform_distribution<int>` object to generate a suitable choice for the legal moves.

6 Your assignment

Your task is to design, declare and define the following:

- `Player` class in `Player.h`
- `HumanPlayer` class in `HumanPlayer.h` and `HumanPlayer.cpp`
- `RandomPlayer` class in `RandomPlayer.h` and `RandomPlayer.cpp`

according to the above specifications described in the pdf. An example `ChessEngine.exe` is given to you with the following `main` function:

```
struct GameConfig
{
    std::string filename;
    Player p1, p2;
};
struct Parser
{
   ...
   int argc; char ** argv; GameConfig gc;
};
int main(int argc, char **argv)
{
    Parser p { argc, argv };
    p.parse();
    std::ifstream fs(p.gc.filename);
    Game g(p.gc.p1, p.gc.p2, WHITE, fs);
    g.PlayGame(true);
    return 0;
}
```

The following error message shows how `ChessEngine` is to be used.

```
Usage: ./ChessEngine <PlayerConfig:int> <BoardStateFileName:string> <RandomSeed:int>
       Player Config Choice:
       1. Human-Human
       2. Human-Random
       3. Random-Human
       4. Random-Random
       Any other choice is undefined
       RandomSeed is an optional argument. Default will be 1.
```

The new Chess Engine should be able to support 4 kinds of play. 1) Human-Human mode where we are playing 2 Human players against one another. 2) Human-Random mode where the first player is Human and second is Random. 3) Random-Human mode where the first player is Random and the second is Human. Finally, 4) Random-Random mode where both players are Random.

The default board state found in `default_board_state.txt` and the implementation of the `Game` class found in `Game.h` and `Game.cpp` are provided as well.

To help test your code extensively, we have prepared some chess playing records in the form of `*-input.txt` and `*-output.txt`. To test these files, you may run the following command:

```
./ChessEngine.exe 1 default_board_state.txt 1 < hh-1-input.txt > stu-hh1-output.txt
./ChessEngine.exe 2 default_board_state.txt 1 < hr-1-input.txt > stu-hr1-output.txt
./ChessEngine.exe 3 default_board_state.txt 1 < rh-1-input.txt > stu-rh1-output.txt
./ChessEngine.exe 4 default_board_state.txt 1 < rr-1-input.txt > stu-rr1-output.txt
```

Then, a `WinMerge` or `diff` of `output.txt` with the corresponding {`hh/hr/rh/rr`}`-1-output.txt` should show no difference at all. You must ensure that the `default_board_state.txt` file is in the same directory before you perform this test with the given `ChessEngine.exe`.

7 Rubrics

You are graded as below:

- Match the given driver output. For every test case not matched, 10 points will be deducted.
- Match the cases where we input either illegal or invalid inputs (for HumanPlayer). You should note that the inputs given are all valid. Please perform the testing of invalid or illegal inputs before you submit it. Failure to detect illegal or invalid inputs can result in deduction of 10 points per case up to a maximum of 30 points deducted.
- Match hidden test cases (not including the cases with invalid or illegal inputs). For every test case not matched, 10 points will be deducted, up to a maximum of 30 points in total. The condition mentioned above applies.
- The other rubrics as laid out in the CS225 rubrics applies to this assignment i.e., Magic Numbers, const correctness etc.

You are required to submit `Player.h`, `HumanPlayer.cpp`, `HumanPlayer.h`, `RandomPlayer.h` and `RandomPlayer.cpp`. All documentation and comments should be done in a style conforming to DOxygen standards (i.e., while index.chm is not required to be submitted, you must following DOxygen style for your function heading comments.)