Project: Chess

Due: 23:59, Sun 13 Dec 2020 Full marks: 100

Introduction

The objective of this project is to let you apply the OOP concepts learned, including inheritance and polymorphism, to programming through developing a game application – Chess – in C++.

Chess is a two-player strategy board game played on a checkered board with 64 cells arranged in an 8×8 grid. The initial board configuration is shown in Figure 1. In a text console environment, we use the letters listed in Table 1 to denote the six types of chess pieces on the board. To denote a position on the board, we use a coordinate system similar to a spreadsheet program, e.g. "A2" refers to the cell in the first column and the second row. Suppose that a C++ array `board` is used to represent the checkerboard, cell address "A2" will be mapped to `board[1][0]`. The original chess game has many complicated rules. We will simplify the game by relaxing some rules in this project. In our modified version of chess, each piece is assigned a specific *score* as shown in Table 1. The scores will be useful for determining the player who wins the game in some cases.

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

⟺

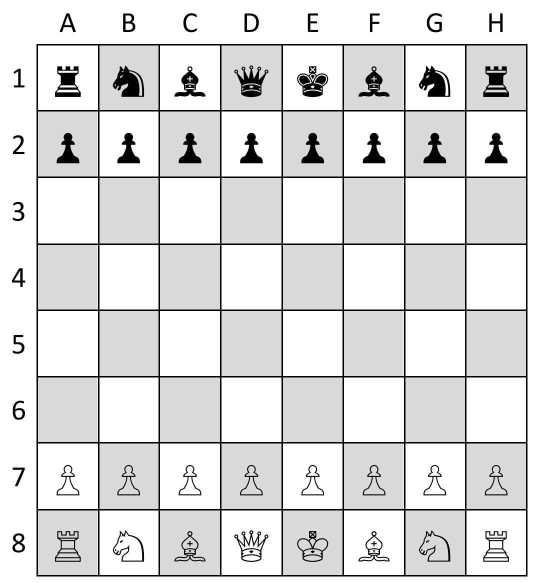

Figure 1: Initial configuration of the game board

Table 1: Chess pieces

Piece	Black Symbol	Black Label	White Symbol	White Label	Count	Score
King	♚	K	♔	k	1	1000
Queen	♛	Q	♕	q	1	50
Bishop	♝	B	♗	b	2	20
K**n**ight	♞	N	♘	n	2	10
Rook	♜	R	♖	r	2	5
Pawn	♟	P	♙	p	8	1

The two players, namely "Black" and "White" make moves in turns. In convention, White plays the first move. Each type of piece has its own way of moving. In Figure 2, the white dots mark the cells to which the piece can move into if there are no intervening piece(s) of either color (except the knight, which leaps over any intervening pieces).

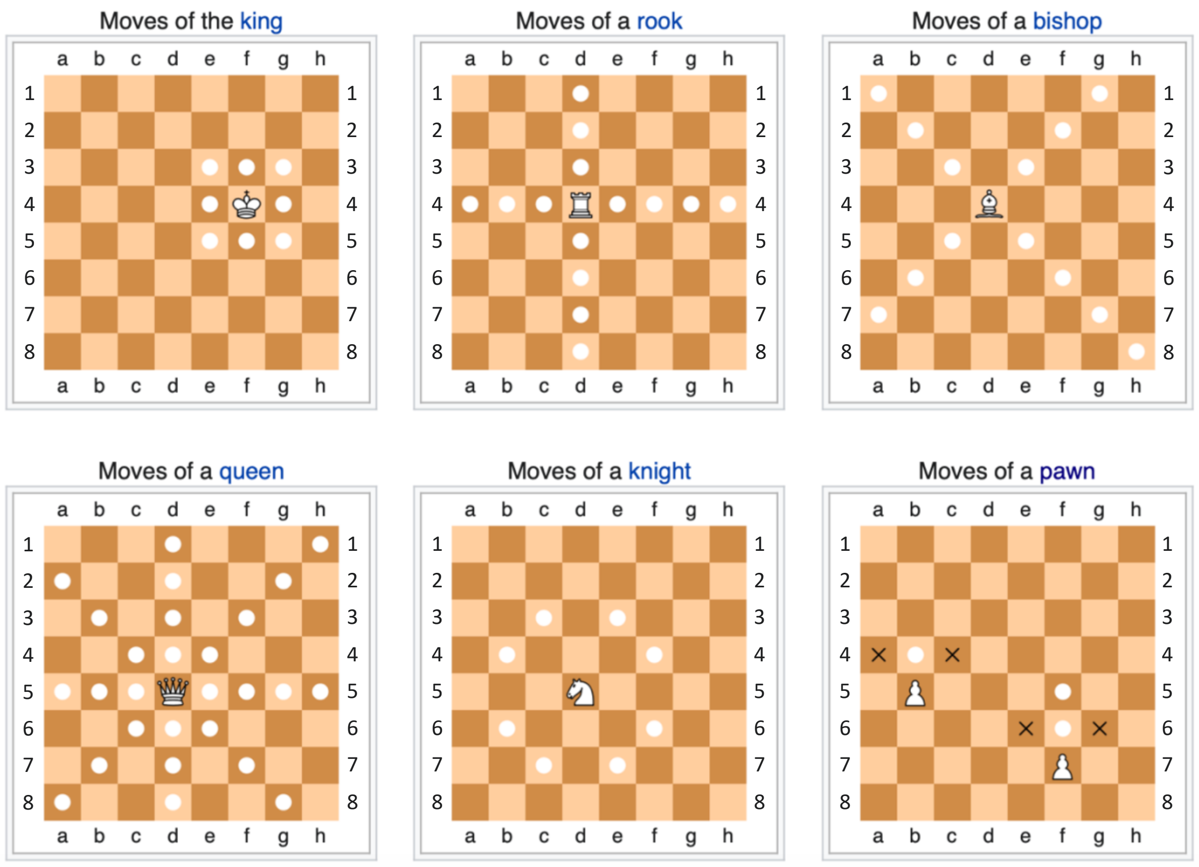

Figure 2: Valid positions for the moves of each kind of game piece

Summary of the rules of moving each piece:

- The **king** moves one cell in any of the eight directions as long as no piece of its own color occupies the target cell. If the target cell is occupied by an opponent piece, the king can capture it.
- A **rook** can move any number of cells along a row (*rank*) or column (*file*) but cannot leap over other pieces.
- A **bishop** can move any number of cells diagonally but cannot leap over other pieces.
- The **queen** combines the power of a rook and bishop and can move any number of squares along a row, column or diagonal, but cannot leap over other pieces.
- A **knight** moves to any of the closest cells that are not on the same row, column or diagonal. Thus, the move forms an L-shape: two cells vertically and one cell horizontally, or two cells horizontally and one cell vertically. The knight is the only piece that can leap over other pieces.
- A **pawn** can move forward to the unoccupied cell immediately in front of it on the same column, or on its first move it can advance two cells along the same column, provided both cells are unoccupied (white dots in the diagram); or the pawn can capture an opponent's piece on cell diagonally in front of it on an adjacent column, by moving to that cell (black "x"s).

For simplicity, special moves known as (1) *castling*, (2) *en passant*, and (3) *promotion* of a pawn (when it advances to the eighth rank) will also be skipped in this project. Please make sure **not to** implement these to let our grading be consistent.

The objective of the game is to *checkmate* the opponent's king by placing it under an inescapable threat of capture. To achieve that, a player's pieces are used to attack and capture the opponent's pieces, while supporting or protecting one another.

Game-over conditions:

The game is over when either of the kings is captured by an opponent's piece. Let's say Black's king is captured successfully by a white piece, then the White player wins. It is possible for the game to draw. In the original chess design, there are complicated rules about how to arrive at a draw game including stalemate, threefold repetition rule, fifty-move rule, mutual agreement and impossibility of checkmate. Again, we will skip them all and use a custom way to define a draw game as follows.

We extend the original chess game with two features: per-piece score and an *ad hoc* way to stop the game. Each piece is assigned a specific *score* as shown in Table 1. For example, the queen is much more worthy than other pieces but incomparable with the king – losing the king (worth 1000 points) will surely lead to losing the game. During a game run, either player can enter a *special imaginary move*, namely "Z0 Z0" (from cell "Z0" to cell "Z0" which are both invalid positions). The program will detect this input to stop the game. While both kings are not yet captured, the player with a higher total score of his/her pieces that are still alive (not yet captured) will win the game. If both players have the same score, the game will draw. This ad hoc halt mimics the draw by agreement (when the game becomes too long and boring) although our version of handling makes poor sense in that the stop is triggered by a single player alone rather than having "mutual agreement".

Program Specification

You have to write the program composed of a total of 25 files (C++ header and source files) listed in the following tables. We have provided a zip file of starter source code to assist your design of this sizeable program. Basically, you need not design but implement the game.

Header files for class definitions:

	File	Description	Remarks
1	`game.h`	The Game class interface	Provided
2	`board.h`	The Board class interface	
3	`piece.h`	The Piece class interface (an abstract base class)	
4	`player.h`	The Player class interface (an abstract base class)	
5	`human.h`	The Human class interface (inheriting the Player class)	
6	`machine.h`	The Machine class interface (inheriting the Player class)	
7	`king.h`	The King class interface (inheriting the Piece class)	
8	`queen.h`	The Queen class interface (inheriting the Piece class)	To be created
9	`bishop.h`	The Bishop class interface (inheriting the Piece class)	
10	`knight.h`	The Knight class interface (inheriting the Piece class)	
11	`rook.h`	The Rook class interface (inheriting the Piece class)	
12	`pawn.h`	The Pawn class interface (inheriting the Piece class)	Provided

Source files for client code and class implementations:

	File	Description	Remarks
13	`main.cpp`	The client program	To fill in TODO parts
14	`game.cpp`	The Game class implementation	
15	`board.cpp`	The Board class implementation	
16	`piece.cpp`	The Piece class implementation	
17	`player.cpp`	The Player class implementation	
18	`human.cpp`	The Human class implementation	
19	`machine.cpp`	The Machine class implementation	
20	`king.cpp`	The King class implementation	Provided
21	`queen.cpp`	The Queen class implementation	To be created
22	`bishop.cpp`	The Bishop class implementation	
23	`knight.cpp`	The Knight class implementation	
24	`rook.cpp`	The Rook class implementation	
25	`pawn.cpp`	The Pawn class implementation	Provided

Note the Remarks column of the above tables:

(1) "Provided": all these source files need no changes and should be kept unmodified.

(2) "To fill in TODO parts": in these files, some code segments are missing. Please look for all TODO comments in them for instructions about what to do and fill in the missing code to finish the implementation of each class.

(3) "To be created": such files are not provided and are to be created by you.

Game Flow

The `main()` function in main.cpp creates a Game object and calls its `run()` method to start a chess game. The `Game()` constructor accepts a game ***mode*** parameter which can be one of the following:

- Value 1 means "**Human vs. Human**", i.e. two human players enter moves via console;
- Value 2 means "**Human vs. Machine**", i.e. one human player enters moves via console and the opponent player is the computer (which makes random moves on the game board);
- Value 3 means "**Machine vs. Machine**", two computer players make random moves in turns without human attention.

(The `Mode` enum in game.h has defined these three constant values.)

When the `Game` object is being created, the constructor will create two Player objects of type (Human / Machine) according to the above mode setting, set game state as *RUNNING (0)* and create the Board object. The Board constructor will call `Board::configure()` method to set up the initial game board by filling the `cells` array with pointers to `King`, `Queen`, `Bishop`, … objects that are being created in the method. The created pieces will be added to each player's list of pieces as well.

The Game object's `run()` is the main loop of the program, which repeats calling the current player's `makeMove(board)` function and printing the board. The current player is flipped in each iteration. For a Human player, its `makeMove()` will prompt the user for two cell addresses, e.g. A7 B8, with the former denoting the source cell and the latter denoting the destination cell on the board. These text-based cell addresses will be translated to array indexes to access the Board object's `cells` array.

Reminder: As mentioned earlier, if a human player enters "Z0 Z0" as the input for his/her turn, this signals that the player wants to stop the game and print the final message about who wins.

For a Machine player, its `makeMove()` will make a random but valid move. To do so, first, pick a piece randomly from the player's `pieces` vector and get its position (y1, x1). (This part has been done for you in the provided source machine.cpp). Then, generate a random target position (y2, x2) and try moving the piece from (y1, x1) to (y2, x2). If the move is invalid, repeat these steps until a valid move is resulted.

Class Hierarchy

The basic structure of this complicated program can be broken down as follows.

Abstract base classes

There are two abstract base classes in this program:

Class **Piece** (piece.h and piece.cpp)

- It serves as the base or parent class of all game pieces like **King**, **Queen**, etc.
- Each Piece object has a label ('K' for King, 'Q' for Queen, etc.), color (BLACK or WHITE), score, and position (y, x) on the board, and a back link (a pointer) to the Board object. (Through that back link, the Piece object can reach and call methods of Board and Game when necessary.)
- It has a `move(int y, int x)` function for making a move. (We have provided the basic actions needed by a move, e.g. capturing the opponent piece if the target position has been occupied by an opponent piece.)
- It has a *pure* virtual function of signature `isMoveValid(int y, int x)`.

Class **Player** (player.h and player.cpp)

- It serves as the base or parent class of **Human** and **Machine** classes.
- Each Player object has a name (in string, "Black" or "White") and a color (in Color enum, BLACK or WHITE). (note: Color enum is defined in piece.h.)
- It has a *pure* virtual function of signature `makeMove(Board* board)`. This method must be implemented by the concrete classes Human and Machine.
- It keeps a list of pieces that are still alive and on the game board. (The list is implemented using a vector storing pointers to the Piece objects.)
- It provides methods to retrieve the count of pieces in the list, get a piece from the list, add a piece to or delete a piece from the list.

Concrete classes

Subclasses of **Piece**
We have provided the following two classes that inherit from Piece as examples of how to complete the implementation of a working game piece:

- Class **King** (king.h and king.cpp)
- Class **Pawn** (pawn.h and pawn.cpp)

They have implemented their own `isMoveValid(int y, int x)`. In particular, the Pawn class has a special need to override the `move(int y, int x)` function to reduce the steps it can move from 2 to 1 after making the first move (this is a chess game rule).

One of your main tasks is to add the following subclasses that inherit from Piece, each of which must implement the pure virtual function `isMoveValid(int y, int x)` according to the rules governing the moves of each piece type. Follow our provided examples (King and Pawn classes) and remember to add "include guard" compiler directives in all header files.

- Class **Queen** (queen.h and queen.cpp)
- Class **Bishop** (bishop.h and bishop.cpp)
- Class **Knight** (knight.h and knight.cpp)
- Class **Rook** (rook.h and rook.cpp)

Subclasses of **Player**

- Class **Human** (human.h and machine.cpp)
  - It implements `makeMove(Board* board)` to repeatedly prompt the current human player to enter two cell addresses of a move until a valid move is obtained. The ad hoc move "Z0 Z0" is detected here and if received, the game state is set to STOPPED and the method returns at once. In the loop body, it calls the `move()` method of the board object, which will validate the move and update the board if the move is confirmed valid.
- Class **Machine** (machine.h and machine.cpp)
  - It implements `makeMove(Board* board)` to make a random valid move on the board by a nested loop. First, pick a piece randomly from the player's `pieces` vector and get its position (y1, x1). Then generate a random position (y2, x2) and try moving the piece from (y1, x1) to (y2, x2). If the trial move is found invalid, repeat the second step until reaching a valid move or a maximum bound of trials (preset as 4096). If it happens that a valid move cannot be found within the maximum trials bound, pick a piece from the player's pieces list randomly again to redo the random search process. The maximum times of the repeated piece picking is 10 x piece count in the vector. In case that the program cannot find any valid move within the trials bound for all piece pickings, the move is regarded *forfeited* – the turn finishes with no actual move (no update of the board), and the turn is passed to the opponent player.
  - Note that ad hoc move "Z0 Z0" is NOT generated by a machine player. There is no premature termination of the game and no draw game for the machine vs. machine mode.

Note that each move will go through a three-level call hierarchy:

```
player->makeMove(board);           // player can be a Human or Machine object
                ↘
         board->move(y1, x1, y2, x2);
                      ↘
                 piece->move(y2, x2); // piece is an object of any Piece subclasses
```

The Board-level `move()` includes validation against illegal cases like accessing out-of-bound positions or moving a piece that belongs to the opponent. Read the TODO comments in board.cpp for more details.

The Piece-level `move()` includes capturing the opponent piece and is called only if calling `piece->isMoveValid(y2, x2)` returns true, i.e. passed validation against the moving rules of the specific piece type, e.g. Bishop can move only diagonally.

Assumptions

- The board size is always 8x8. Still, we use two global constants H and W to represent the board's height (number of rows) and width (number of columns). They are both set to 8.
- We assume that cell address inputs always follow the format of one letter (A-Z or a-z) plus one integer (1-26). Lowercase inputs like "a1", "h10", etc. will be accepted as normal. Except for "Z0" (case-sensitive) which is treated as a *sentinel* for game stop, you need NOT handle weird inputs like "AA1", "A01", "abc", "A3.14", "#a2", … for cell addresses and inputs like "-1", "6.28", "#$@", … for the number of pits. The behavior upon receiving these is unspecified, probably running into forever loop or program crash.

Restrictions

- You cannot declare any global variables (i.e. variables declared outside any functions). But global constants or arrays of constants are allowed.
- Your final program should contain a total of 25 files (the C++ header and source files listed above). Note your spelling - do not modify any file names.
  - For class names, use camel case (e.g. Knight); for file names, use lower case for all letters (e.g. knight.h, knight.cpp).
- You should not modify the provided code (unless specified otherwise).

Sample Runs

In the following sample runs, the blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (same text, symbols, letter case, spacings, etc.). Note that *there is a space after the ':' in the program printout*.

Sample run in Human vs. Human mode:

```
Choose game mode (1, 2, 3): 1↵
Round 1:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 |   |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p | p | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: e2 e3↵
invalid input!
White's turn: d7 d5↵
Round 2:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
```

White's turn should be moving a white piece (now located in rows 7 and 8).

This is a valid move. Cell address inputs are case-insensitive.

```
2 | P | P | P | P | P | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 |   |   |   | p |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p |   | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
Black's turn: e2 e4↵
Round 3:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P |   | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   | P |   |   |   |
  ---------------------------------
5 |   |   |   | p |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p |   | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: d5 e4↵
Round 4:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P |   | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   | p |   |   |   |
  ---------------------------------
5 |   |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p |   | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
Black's turn: f1 b5↵
Round 5:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K |   | N | R |
  ---------------------------------
2 | P | P | P | P |   | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   | p |   |   |   |
  ---------------------------------
5 |   | B |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p |   | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
```

```
White's turn: b7 b6↵
Round 6:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K |   | N | R |
  ---------------------------------
2 | P | P | P | P |   | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   | p |   |   |   |
  ---------------------------------
5 |   | B |   |   |   |   |   |   |
  ---------------------------------
6 |   | p |   |   |   |   |   |   |
  ---------------------------------
7 | p |   | p |   | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
Black's turn: b5 e8↵
Black wins!
```

Now the white king is *in check* (being attacked by the black bishop). White's turn should make a move that saves the king from the capture. But White made a foolish move here that leads to losing the game.

Sample run in Human vs. Machine mode:

```
Choose game mode (1, 2, 3): 2↵
Round 1:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 |   |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p | p | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: a7 a5↵
Round 2:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 | p |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 |   | p | p | p | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
Black's turn: G2 G4
Round 3:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P |   | P |
  ---------------------------------
```

```
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   | P |   |
  ---------------------------------
5 | p |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 |   | p | p | p | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: a8 a6↵
Round 4:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P |   | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   | P |   |
  ---------------------------------
5 | p |   |   |   |   |   |   |   |
  ---------------------------------
6 | r |   |   |   |   |   |   |   |
  ---------------------------------
7 |   | p | p | p | p | p | p | p |
  ---------------------------------
8 |   | n | b | q | k | b | n | r |
  ---------------------------------
Black's turn: F1 G2
Round 5:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K |   | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | B | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   | P |   |
  ---------------------------------
5 | p |   |   |   |   |   |   |   |
  ---------------------------------
6 | r |   |   |   |   |   |   |   |
  ---------------------------------
7 |   | p | p | p | p | p | p | p |
  ---------------------------------
8 |   | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: z0 z0↵
invalid input!
White's turn: Z0 Z0↵
Draw game!
```

By entering "Z0 Z0" (case-sensitive), White wants to stop the game. As no piece is captured, both players have the same total score. The game draws.

Sample run in Machine vs. Machine mode:

```
Choose game mode (1, 2, 3): 3↵
Round 1:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 |   |   |   |   |   |   |   |   |
  ---------------------------------
```

```
6 |   |   |   |   |   |   |   |   |
  ---------------------------------
7 | p | p | p | p | p | p | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: F7 F6
Round 2:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | P | P |
  ---------------------------------
3 |   |   |   |   |   |   |   |   |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 |   |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   | p |   |   |
  ---------------------------------
7 | p | p | p | p | p |   | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
Black's turn: H2 H3
Round 3:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 | R | N | B | Q | K | B | N | R |
  ---------------------------------
2 | P | P | P | P | P | P | P |   |
  ---------------------------------
3 |   |   |   |   |   |   |   | P |
  ---------------------------------
4 |   |   |   |   |   |   |   |   |
  ---------------------------------
5 |   |   |   |   |   |   |   |   |
  ---------------------------------
6 |   |   |   |   |   | p |   |   |
  ---------------------------------
7 | p | p | p | p | p |   | p | p |
  ---------------------------------
8 | r | n | b | q | k | b | n | r |
  ---------------------------------
White's turn: B8 A6

...

Round 64:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 |   | R | B |   |   | Q |   | R |
  ---------------------------------
2 |   |   | P | p | B | P | P |   |
  ---------------------------------
3 |   |   |   |   | K | N |   |   |
  ---------------------------------
4 | P |   |   |   | p | p |   |   |
  ---------------------------------
5 | p | P |   |   |   |   |   |   |
  ---------------------------------
6 |   | p | k |   |   |   | P |   |
  ---------------------------------
7 |   |   | r |   |   |   | q | p |
  ---------------------------------
8 |   | n |   |   |   | b | N | r |
  ---------------------------------
Black's turn: G6 H7
Round 65:
    A   B   C   D   E   F   G   H
  ---------------------------------
1 |   | R | B |   |   | Q |   | R |
  ---------------------------------
2 |   |   | P | p | B | P | P |   |
  ---------------------------------
3 |   |   |   |   | K | N |   |   |
  ---------------------------------
4 | P |   |   |   | p | p |   |   |
  ---------------------------------
5 | p | P |   |   |   |   |   |   |
  ---------------------------------
6 |   | p | k |   |   |   |   |   |
  ---------------------------------
7 |   |   | r |   |   |   | q | P |
  ---------------------------------
8 |   | n |   |   |   | b | N | r |
  ---------------------------------
White's turn: F4 E3
White wins!
```

- There are too many combinations of possible inputs. Please check your program correctness against the results produced by our sample program executable posted on Blackboard.

Submission and Marking

- Submit a zip of your source files to Blackboard (https://blackboard.cuhk.edu.hk/).
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of your main.cpp file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT plagiarize.*** Sending your work to others is subject to the same penalty for copying work.