

# Project: Lines of Action

Due: 23:59, Sat 9 May 2020 File name: LinesOfAction.cpp, gameplay.cpp Full marks: 100

## Introduction

The objective of this project is to practice object-oriented programming. You will implement a board game called *Lines of Action* (集結棋), which is played on an 8 × 8 board by two players Black and White. The initial game setup is shown in Figure 1(a). The symbols 'B', 'W', and '.' denote black piece, white piece, and empty square respectively. The rows and columns are numbers 0–7 and *lowercase* letters a–h respectively.

Two players take turns to move one of their pieces horizontally  $\leftrightarrow$ , vertically  $\updownarrow$ , or diagonally  $\nearrow\searrow$ . The piece moves exactly as many squares as there are pieces on the line in which it is moving. E.g., the B in a1 may move *two* squares to c1, because there are totally *two* pieces on row 1. A piece may jump over pieces of the same player but *not* over opponent's pieces (Figure 1(b)). A piece may land on and capture an opponent's piece, which will then be removed from the board (Figure 1(c)). A piece may *not* land on a piece of the same player. The goal of a player is to make all his/her pieces adjacent to each other vertically, horizontally, and diagonally (Figure 1(d)).

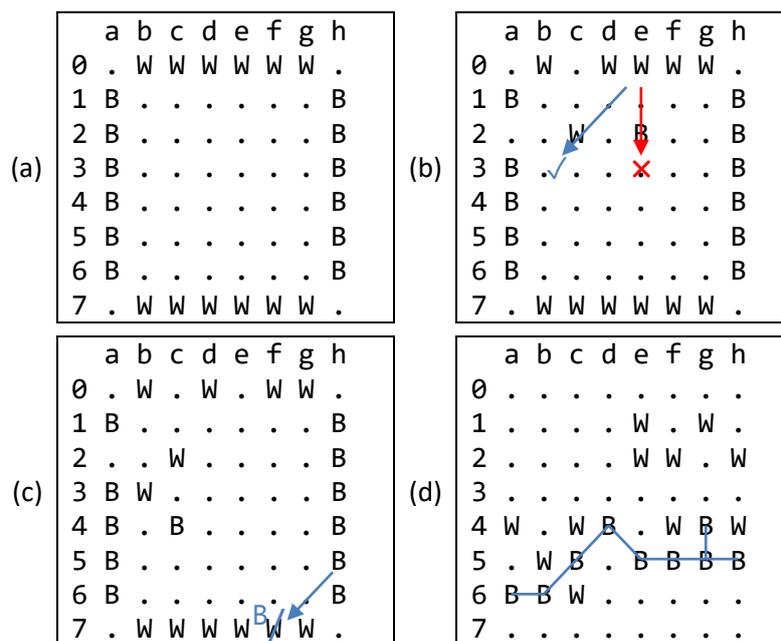


Figure 1: (a) Initial setup, (b) Jump over, (c) Capture, and (d) Black wins

Sometimes a player's move may result in the opponent forming all adjacent pieces due to capture. This commits suicide and the opponent wins. A move may also result in both players forming all adjacent pieces. Such simultaneous connection is considered as a draw. A player with only one piece left (due to captures) is by definition connected. A player may also have no possible moves; but we simply assume this will not happen in this project.

## Program Specification

You have to write your program in two source files `LinesOfAction.cpp` and `gameplay.cpp`. The former is the implementation of the class `LinesOfAction`, while the latter is a client program of class `LinesOfAction` which performs the game flow. You are recommended to finish the `LinesOfAction` class first before writing the client program. When you write the `LinesOfAction` class, *implement the member functions and test them individually one by one. Your two files will be graded separately, so you should not mix the functionalities of the two files.*

### Class `LinesOfAction` (`LinesOfAction.cpp`)

You are given the interface of the `LinesOfAction` class in the header file `LinesOfAction.h`. You *shall not modify* the contents of this header file. Descriptions of its members are given below.

```
class LinesOfAction {
public:
    LinesOfAction();
    void printGame();
    char getCurrentPlayer();
    void swapPlayer();
    bool move(string from, string to);
    bool hasConnected(char p);
    char gameOver();
private:
    char board[8][8];
    char currentPlayer, nextPlayer;
    int blacks, whites;
};
```

#### Private Data Members

##### `char board[8][8];`

The game board is represented by a two-dimensional array of `char`, storing either 'B', 'W', or '.'. The elements `board[0][0]`, `board[0][7]`, `board[7][0]`, and `board[7][7]` are the positions a0 h0, a7 and h7 respectively.

##### `char currentPlayer, nextPlayer;`

The player in the current move and in the next move respectively. They should be either 'B' or 'W'.

##### `int blacks, whites;`

The total number of black and white pieces on the board respectively.

#### Public Constructor and Member Functions

##### `LinesOfAction();`

This constructor creates a game object and initialize it to the setup in Figure 1(a). Black starts playing first. (So White is the next player.) There are 12 black and white pieces each on the board initially.

##### `void printGame();`

Prints out the game board in the format in Figure 1.

### char getCurrentPlayer();

Returns the current player of the game, i.e., the value of the data member `currentPlayer`.

### void swapPlayer();

Swaps the current and next players in the game. This is for changing turns during the game play.

### bool move(string from, string to);

Carries out the current player's move from the source position `from` to the landing position `to`. The parameters `from` and `to` are strings whose format is a column letter followed by a row number, e.g., "a1", "c7", and "d4". The member function shall check whether the `from` and `to` positions form a valid move. A move is valid if all the following conditions are satisfied:

- The parameters `from` and `to` are valid board positions. (Only lowercase letters can be valid.)
- The `from` position contains a piece of the current player.
- The move is either horizontal, vertical, or diagonal.
- The move is exactly as many squares as there are pieces on the line in which it is moving.
- The move does not jump over opponent's piece(s).
- The landing position `to` is either an empty square or an opponent's piece.

When the move is valid, the array `board` shall be updated to reflect the result of the move, and the data members `blacks` or `whites` shall be updated if it is a capture. The member function returns `true` if the move is valid; and `false` otherwise.

*Warning: this member function is difficult to implement!*

### bool hasConnected(char p);

This member function returns `true` if player `p` has all his/her pieces adjacent to each other vertically, horizontally, and diagonally; and `false` otherwise.

*Warning: this is really difficult to implement!*

### char gameOver();

This member function checks if the game is over. It returns either 'B', 'W', 'D', or '-', to mean the following:

Return value	Meaning
'B'	Black wins. (Black pieces are all adjacent but white pieces are <i>not</i> .)
'W'	White wins. (White pieces are all adjacent but black pieces are <i>not</i> .)
'D'	Draw game. (Black pieces are all adjacent <i>and</i> white pieces are also all adjacent.)
'-'	None of the above. (Game is not yet over.)

This member function can be written with the help of calling `hasConnected()`.

## Client Program (gameplay.cpp)

Your main program is a client of the `LinesOfAction` class; it performs the flow of the game.

1. Create a `LinesOfAction` object.
2. Prompt the player to make a move. The input consists of the source and landing positions, each of which is a character followed by an integer. E.g., `a1 c3`. (Hint: You can use `cin >> ... >> ...;` to read in two strings.)

3. Make the player move. When the move is invalid, warn the player and prompt the same player to enter again until a valid move is entered.
4. Swap the players.
5. If the game is not over, go back to step 2.
6. When the game is over, print the messages “B wins!”, “W wins!”, and “Draw game!” accordingly.

### Some Points to Note

- You cannot declare any global variables in all your source files (except const ones).
- You can define extra functions in any source files if necessary. However, extra *member* functions (instance methods), no matter `private` or `public`, are not allowed.
- Your `LinesOfAction` class should not contain any cin statements. All user inputs shall be done in the client program (`gameplay.cpp`) only.
- Your `LinesOfAction` class should not contain any cout statements except in the printGame() member function (for printing the game board).

### Sample Run

In the following sample run, the **blue** text is user input and the other text is the program output. 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.

```
  a b c d e f g h
0 . W W W W W W .
1 B . . . . . B
2 B . . . . . B
3 B . . . . . B
4 B . . . . . B
5 B . . . . . B
6 B . . . . . B
7 . W W W W W W .
B's move: a2 c4↵
  a b c d e f g h
0 . W W W W W W .
1 B . . . . . B
2 . . . . . B
3 B . . . . . B
4 B . B . . . . B
5 B . . . . . B
6 B . . . . . B
7 . W W W W W W .
W's move: C0 b1↵
Invalid move. Try again!
W's move: j0 b1↵
Invalid move. Try again!
WW's move: c0 c8↵
```

Invalid move. Try again!

W's move: c0 d1↵

Invalid move. Try again!

W's move: c0 d2↵

Invalid move. Try again!

W's move: c0 b1↵

a b c d e f g h

0 . W . W W W W .

1 B W . . . . . B

2 . . . . . . . B

3 B . . . . . . B

4 B . B . . . . . B

5 B . . . . . . B

6 B . . . . . . B

7 . W W W W W W .

B's move: a1 d1↵

Invalid move. Try again!

B's move: a6 a1↵

Invalid move. Try again!

B's move: h1 h7↵

a b c d e f g h

0 . W . W W W W .

1 B W . . . . . .

2 . . . . . . . B

3 B . . . . . . B

4 B . B . . . . . B

5 B . . . . . . B

6 B . . . . . . B

7 . W W W W W W B

W's move: d7 b5↵

a b c d e f g h

0 . W . W W W W .

1 B W . . . . . .

2 . . . . . . . B

3 B . . . . . . B

4 B . B . . . . . B

5 B W . . . . . B

6 B . . . . . . B

7 . W W . W W W B

B's move: c4 g0↵

```
  a b c d e f g h
0 . W . W W W B .
1 B W . . . . .
2 . . . . . B
3 B . . . . . B
4 B . . . . . B
5 B W . . . . . B
6 B . . . . . B
7 . W W . W W W B
  ⋮      (Many moves skipped. See Blackboard for full version.)
  a b c d e f g h
0 . . . . .
1 . . . . . W
2 . . . . . W .
3 . . . B B . .
4 . . . . . B .
5 . . . . . B B .
6 B W W . . . .
7 . . . . .
B's move: a6 c4↵
  a b c d e f g h
0 . . . . .
1 . . . . . W
2 . . . . . W .
3 . . . B B . .
4 . . B . . B .
5 . . . . . B B .
6 . W W . . . .
7 . . . . .
B wins!
```

## Submission and Marking

- Your program file names should be LinesOfAction.cpp and gameplay.cpp. Submit the two files in Blackboard (<https://blackboard.cuhk.edu.hk/>). You do not have to submit LinesOfAction.h.
- Insert your name, student ID, and e-mail as comments at the beginning of all your source files.
- Besides the above information, your program should include suitable comments as documentation in all your files.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be free of compilation errors and warnings.
- **Do NOT plagiarize**. Sending your work to others is subjected to the same penalty as the copier.