CS 1110 Assignment 1

Gomoku

Name: `Gomoku.java`

Gomoku is a game played by two players on a 19 by 19 board. One player uses black stones and the other player uses white stones. The two players alternate putting down stones on the board with the goal to be the first to get five of their stones in a row either vertically, horizontally, or diagonally. Your task is to make a text-based implementation of this game.

This assignment entails several steps to arrive at the completed game. For each step, add on to what you already have. When you've completed all the parts, you should have only a single .java file to submit that contains a functional Gomoku program.

1. Display the Board (4 points)

To start out with, we want to simply be able to display the game board. The board can be represented as a 2D array of characters. In particular, we can use . to represent blanks, and there's a couple unicode characters that should work to represent the stones: ● (\u25cf) and ○ (\u25cb). We'll make a method to display the board. Here is the method signature to get you started:

```
public static void displayBoard(char[][] board)
```

Sample Run (Empty Board):

```
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
```

2. Place a Stone (4 points)

Once we can display the board, the next step will be to allow the player to place a stone. For the player to place a stone, we'll simply ask them to enter the row and column for where they would like to place their stone and then put their stone at that location and then display the board again.

Sample Run (Placing a Stone):

```
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .

Player 1's turn!
Enter row: 3
Enter column: 3

 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . ● . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
```

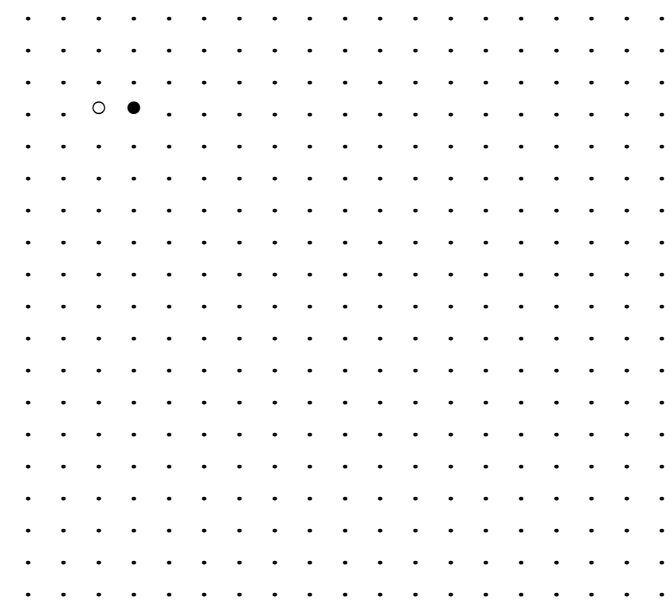
3. Alternate Turns Between Players (4 points)

Now that we can allow a player to place a stone, we would like to allow the next player to place a stone, and alternate back and forth between players placing stones.

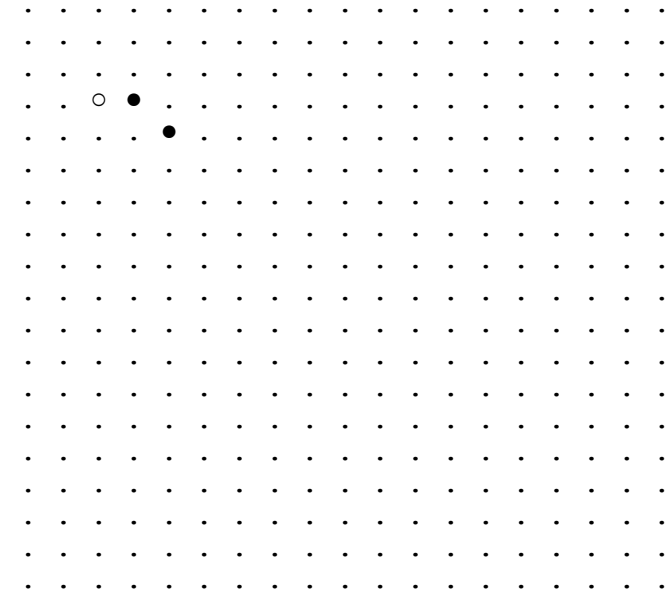
Sample Run (Alternating Between Players – Abbreviated):

```
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .

Player 1's turn!
Enter row: 3
Enter column: 3

 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . ● . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
```

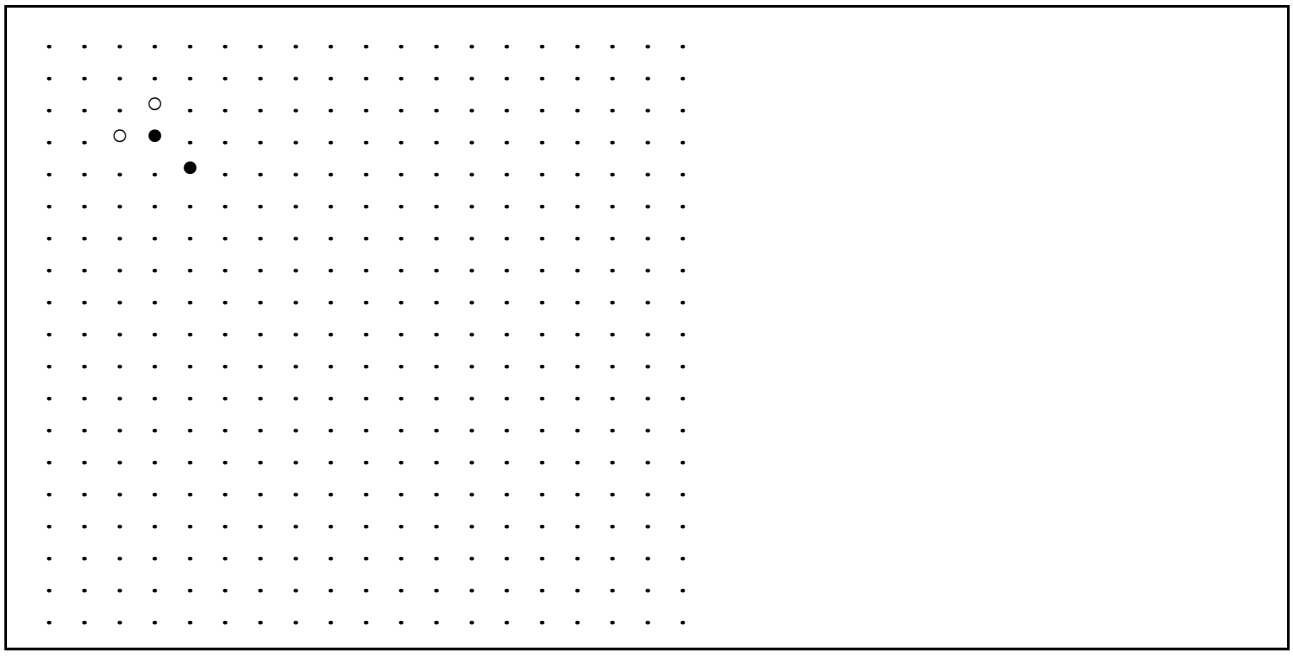
```
Player 2's turn!
Enter row: 3
Enter column: 2
```



```
Player 1's turn!
Enter row: 4
Enter column: 4
```



```
Player 2's turn!
Enter row: 2
Enter column: 3
```



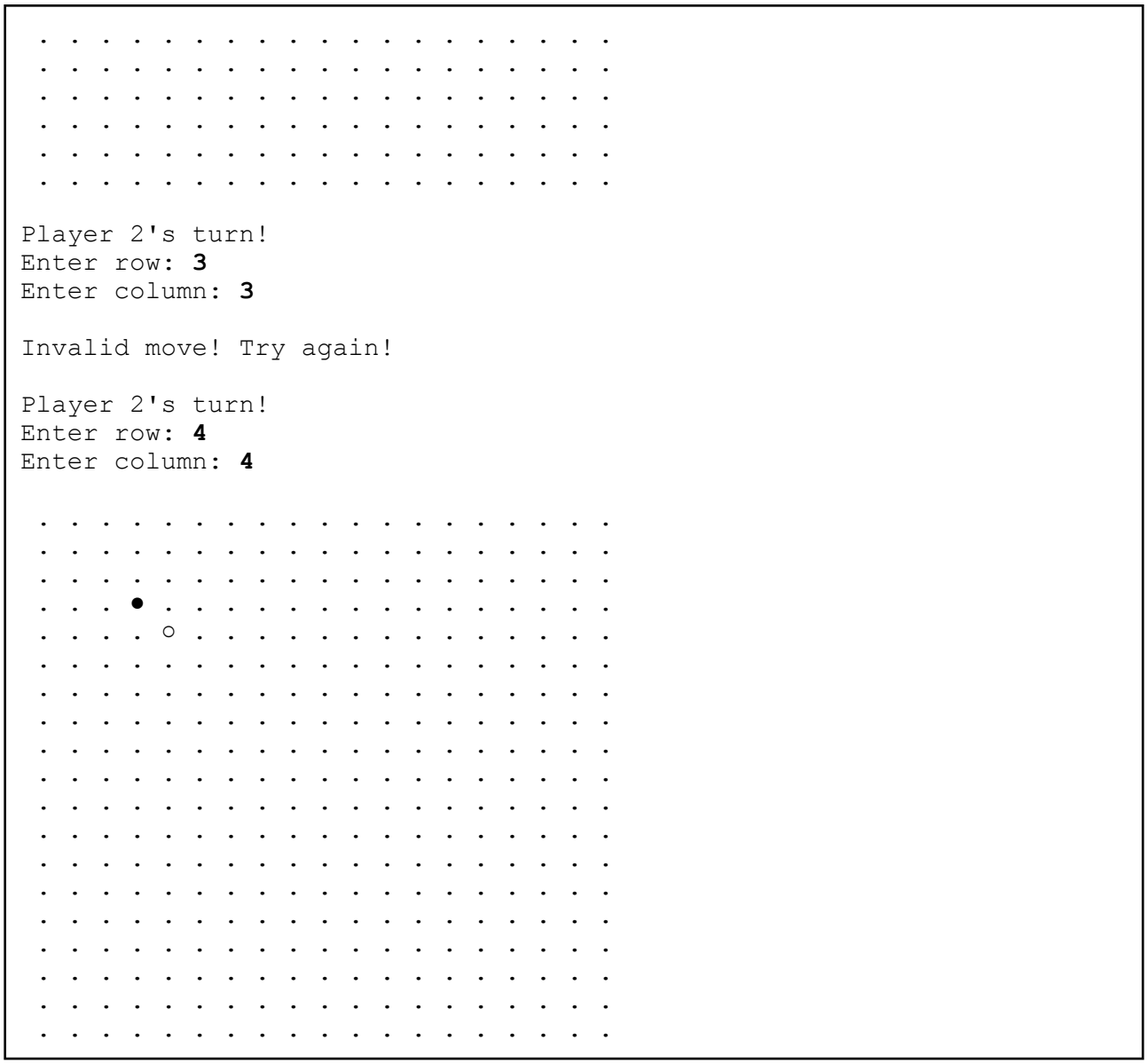
4. Validate Move (4 points)

Now that we can get moves from the players, we should really do some validation to make sure the moves are valid. In particular, we need to make sure the user entered a valid row and column (otherwise it could crash the program). Also, we need to make sure the user isn't putting a stone in a place where there's already a stone. If the player tries to make an invalid move, display a message saying so, and allow them to try again. To simplify checking to see whether the move is valid or not, create a method that perform the check and returns a boolean. Here's a method signature to get you started:

```
public static boolean isValidMove(char[][] board, int row, int column)
```

Sample Run:

```
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .

Player 1's turn!
Enter row: 3
Enter column: 3

 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . ● . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
```


Player 2's turn!
Enter row: 3
Enter column: 3
Invalid move! Try again!
Player 2's turn!
Enter row: 4
Enter column: 4

5. Detect End of Game (4 points)

There's basically two conditions in which the game can end: a player gets five of their stones in a row or the board is full and neither player has five stones in a row. First, create a method to detect if the board is full. Here's a method signature to get you started:

```
public static boolean isBoardFull(char[][] board)
```

Similarly, make a method to detect if a player has won:

```
public static boolean hasPlayerWon(char[][] board, int player)
```

There's really three possibilities for a player winning: horizontal win, vertical win, diagonal win. To make hasPlayerWon() easier to implement, create a method for checking if there's a horizontal win and another method for checking if there's a vertical win and use them in hasPlayerWon().

```
public static boolean isHorizontalWin(char[][] board, int player)
```

```
public static boolean isVerticalWin(char[][] board, int player)
```

Detecting diagonal wins is more difficult, so your program doesn't need to be able to detect diagonal wins, but you can implement detecting diagonal wins if you want for extra credit (5 points).

Sample Run (At the End of a Game – Abbreviated):

```
Player 1's turn!
Enter row: 3
Enter column: 4

 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . ● ● ● ● ● . . . . . . . . . . . . .
 . . . ○ ○ ○ . . . . . . . . . . . . .
 . . . . . ○ . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . .

Player 1 wins!
```