Introduction

In this assignment, you will create a program that simulates a 2-player turn-based strategy board game called Endodoi**.** Endodoi is a game played in Kenya and Tanzania and is a version of the popular Mancala game. The game is played with small stones, beans, or seeds and the playing surface are rows of holes or pits in the ground or a game board. There are no "formal rules" for the game and there are different published rules for the game. This assignment adds some additional variants to commonly found rules. The rules for the game that you need to implement for this assignment are provided below.

https://www.flickr.com/photos/revgriddler/2736706261/in/photostream/	By Víktor Bautista i Roca - www.wikimanqala.org, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=1613947

The rules of the game:

1. The board consists of two rows of **pits** or holes. Common layouts for the pits in Endodoi are 2 × 8 (2 rows and 8 pits per row for 16 pits total), 2 x 10, and 2 x 12 (shown in the second image on the previous page.) For testing of your program, pit layouts will range from 2 x 1 through 2 x 12 inclusive (always two rows though).
2. Players begin by placing a certain number of pieces called **seeds**, in each of the pits on the game board. Common starting amounts are 3 to 6 seeds per pit. For testing your program, the number of starting seeds per pit will range between 1 and 6 seeds inclusive.
3. There are two players. One player plays from the bottom row of pits and the other plays from the top row of pits.
4. There are also two **stores**, one for each player. The stores are only used for storing captured seeds. In the game board layout for our version, both stores are on the far left, the top one for player 1 and the bottom one for player 2.
5. At his or her turn, the player takes all seeds from one of their pits and **sows** them counterclockwise (examples will be given when it is time to implement these rules);
    a. **Sowing** seeds means placing one seed in a pit and then placing one seed in each additional pit in a counter-clockwise sequence. Sowing includes placing seeds in the other player's pit.
    b. Seeds are not sowed into either store, meaning that stores are only used for storing captured seeds not for placing seeds during a turn.
    c. Let's call the player who is sowing seeds Player A, and the opponent Player B. When the last seed of Player A is sown into an empty pit,
        i. If the empty pit belongs to Player A's row, he or she will capture any seed in Player B's neighboring pit (including the last seed dropped by Player A in their pit, if applicable). Then Player A's turn ends.
        ii. If the last seed is either 1) dropped into Player B's pit (that is empty), or 2) in an empty pit of Player A that is opposite an empty pit belonging to Player B (also empty), then Player A's turn ends without any captures.
    d. The player may only select from one of their own pits that currently has seeds in it, meaning a player is not allowed to select an empty pit for their turn.
6. This is a **relay sowing** game which means if sowing ends in an occupied pit of either player, the seeds in that hole are picked up and a new sowing begins as part of the same turn. This process continues until the sowing ends into an empty pit.
7. When one of the players cannot move anymore, the game is over. The seeds that are still on the board, are captured by the player who moved last.
8. The winner is the player who captured the most seeds. If the players each capture half the seeds, a draw is declared.

To help guide you through the process, this assignment will be broken into smaller parts where one part builds on the next. You will initially write a program that utilizes methods to control the display and execution of the game. This submission will represent a significant milestone in the assignment. The next submission due the following week will involve developing methods related to game play including sowing. The final submission due the following week will involve developing some additional methods to help control actual play of the game.

This is a challenging but doable assignment if you follow the process. It will require you to draw heavily on selection control, looping structures and 2D arrays. To help, a lab will be dedicated to providing assistance with getting started with the assignment in the beginning. You are encouraged to meet with TAs through the assignment to review and refactor your code. They can help to guide you in making your code manageable and identify areas of confusion in your code. This will serve to make you a better programmer and better understand computational thinking.

To help with your planning, the overall structure and assessment points of the assignment are provided below.

- Assignment Part I: Game Board Set-Up (due first week) 50 points
- Assignment Part II: Sow Methods (due following week) 40 points
- Assignment Part III: Game Control (due following week) 50 points

Assignment

Part I: Game Board Set-Up

For the first submission, you will work on building methods necessary for the game to run. In this phase of the assignment, we are going to focus on creating a visual display (ASCII art) of the board and setting the board state. Recall that state means the values of variables in the program.

1. Create a class called **Endodoi**
2. **Write stubs for all of your methods before submitting to MIMIR**
3. The next step is to implement the **createNewPits** method described below
4. Implement **playerPitLabels** method described below
5. Implement the **drawBoard** method described below

The **createNewPits** Method

This method will set up the initial starting seeds in a pit. Note this method does not print anything, only returns a 2D array.

Method signature

```
public static int[][] createNewPits(int pitsPerRow, int numSeeds)
```

- **pitsPerRow** is the number of pits for each row. For example if the argument to this parameter is 6, the board is 2 x 6 (2 rows of 6 pits).
- **numSeeds** is the number of seeds to place in each pit at the start of the game

Sample Output

```
int [][] pits = createNewPits(3,2);
// pits value is {{2, 2, 2},{2, 2, 2}}
```

The **playerPitLabels** Method

This method returns a 2D array with the character labels in the pit. You may find this method helpful for printing out the labels for the drawBoard method (you could call playerPitLabels from drawBoard).

Method signature

```
public static char[][] playerPitLabels(int[][] playerPits)
```

- **playerPits** is a 2d array of seeds in pits

Sample output

```
int [][] pits = {{2, 2, 2},{2, 2, 2}};
char [][] pitLabels = playerPitLabels(pits);
// pitLabels value is {'a', 'b', 'c'}, {'d', 'e', 'f'}}
```

The **drawBoard** Method

This method creates an ascii art representation of the board and returns it.

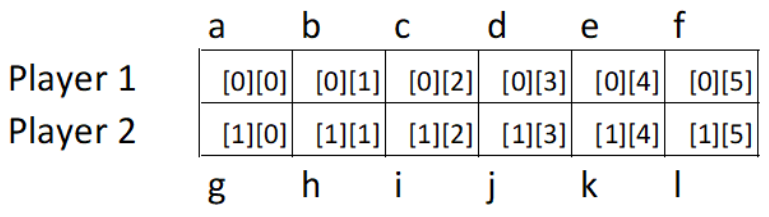
Method signature

```
public static String drawBoard(int[][] gamePits, int storePlayer1,
                               int storePlayer2, char selectedPit,
                               int indicatePlayerTurn)
```

This method will need to perform the following actions:

- Create a representation of the playing board as shown in figures below that show the players, stores, rows, and seeds in pits
- On the board insert character labels for the *identifier* of each pit
- Use the parameter **selectedPit** to highlight a specific pit using an asterix
- Use the parameter **indicatePlayerTurn** to alter the display of the board to indicate which player's turn it is, or to indicate no turn.

The 2D int array **gamePits** represents all of the pits on the playing board, with one half of the array being Player 1's pits and the other half of the array will be Player 2's pits. The diagram below shows an array mapping for a 2 X 6 playing board (with shown indexes inside each cell).



	a	b	c	d	e	f
Player 1	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]
Player 2	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]
	g	h	i	j	k	l

The 0 index of the first dimension (row 0) has the pits for Player 1 and the 1 index of the first dimension (row 1) has the pits for Player 2. The index of the second dimension of the array records the number of seeds in each player's pits from left to right.

Below is an example of an initialization of a 2D array, along with a figure showing how many seeds are in each pit. As an example, the number of seeds in pits[0][2] is 5 and corresponds to the label c.

```
int [][] pits = new int[][] { { 0, 3, 5, 2, 1, 0 }, { 1, 2, 0, 0, 1, 4 } };
```

	a	b	c	d	e	f
Player 1	0	3	5	2	1	0
Player 2	1	2	0	0	1	4
	g	h	i	j	k	l

The figure below shows the result of the drawBoard function for a 2 X 8 implementation with 4 seeds per pit. The parenthesis represent the pit and the character below the pit is the identifier (way to refer to) that specific pit. The top part of the board includes the store and pits for Player 1 and the bottom part of the board includes the store and pits for Player 2. The **st** on the far left indicates the store for each player, i.e. Player 1's store is the upper far-left.

An empty pit or store has no number between the parenthesis. A pit with seeds has an integer representing the number of seeds in the pit.

Figure 1: Example of drawBoard that shows 2 x 8 board with 4 seeds per pit and no captured seeds in either player store

```
 +====++==++----+----+----+----+----+----+----+----+
1|(  )||  ||( 4)|( 4)|( 4)|( 4)|( 4)|( 4)|( 4)|( 4)|
 | st ||  ||  a |  b |  c |  d |  e |  f |  g |  h |
 +====++==++----+----+----+----+----+----+----+----+
2|(  )||  ||( 4)|( 4)|( 4)|( 4)|( 4)|( 4)|( 4)|( 4)|
 | st ||  ||  i |  j |  k |  l |  m |  n |  o |  p |
 +====++==++----+----+----+----+----+----+----+----+
```

The arguments to create the above drawBoard method are:

```
// Creates an array of ints representing seeds in each pit on the
board
int [][] pits = new int[][]{{ 4, 4, 4, 4, 4, 4, 4, 4 },
                            { 4, 4, 4, 4, 4, 4, 4, 4 } };
```

```
System.out.println(drawBoard(pits, 0, 0, '!', -1));
```

The parameter **selectedPit** is used to highlight a specific pit by replacing the alphabetic character identifying the pit with an asterix. If the passed value is **any character** that is not a label on the playing board, the method does not highlight any pits, otherwise it indicates the pit to highlight. The figure below shows an example of a 2 x 4 game board with captured seeds, varying number of seeds in a pit, and a highlighted pit.

```
 +====++==++----+----+----+----+
1|( 3)||  ||( 3)|( 2)|( 1)|(  )|
 | st ||  ||  a |  b |  c |  d |
 +====++==++----+----+----+----+
2|( 2)||  ||( 4)|( 4)|( 4)|( 1)|
 | st ||  ||  e |  * |  g |  h |
 +====++==++----+----+----+----+
```

The arguments to create the above drawBoard method are:

```
int [][] pits = {{ 3, 2, 1, 0 },{ 4, 4, 4, 1 }};
System.out.println(drawBoard(pits, 3, 2, 'f', -1));
```

The parameter **indicatePlayerTurn** takes an int which alters the gameboard presentation to indicate which player's turn it is. If the passed value is **-1**, the game board is not altered. The other valid values are **1** or **2**, indicating Player 1 or Player 2 respectively. The figure below shows the same gameboard as above but with player 1 highlighted.

The behavior is to place in X in the box between the store and pits of the player whose turn it is and to turn off display of the ids of the player who does not have currently have a turn. Note that the asterix highlighting still displays even if the character ids of a row are not supposed to display.

```
 +====++==++----+----+----+----+
1|( 3)||\/||( 3)|( 2)|( 1)|(  )|
 | st ||/\||  a |  b |  c |  d |
 +====++==++----+----+----+----+
2|( 2)||  ||( 4)|( 4)|( 4)|( 1)|
 | st ||  ||    |  * |    |    |
 +====++==++----+----+----+----+
```

The arguments to create the above drawBoard method are:

```
int [][] pits = {{ 3, 2, 1, 0 },{ 4, 4, 4, 1 }};
System.out.println(drawBoard(pits, 3, 2, 'f', 1 ));
```

Getting Started/Helpful Hints for Part I

Below are some suggestions on how to get started/work through the assignment.

1. Build on the work from the Endodoi starter lab. Use the setBoard method to create different board configurations.
2. Create Unit tests of your code. When grading, we will use a similar method to test functionality.

Submission for Part I

Submit your **Endodoi.java** file to **Mimir** before the deadline. You may submit more than one time, the last attempt before the deadline is the one that is graded.

Before uploading your program, be sure to complete the following steps carefully:

- Your input and output should match the input and output test cases exactly, character for character.

Part II: Sow Methods

For the second submission, you will work on building two of the methods necessary for a player pit selection to change how the seeds are distributed on the board. This part of the assignment is challenging as it requires you to think about how to implement these methods with the data structures that you have to work with. There are many, many different ways to approach how you develop these methods, there is no one “right” answer. Also if you have not written helper methods for other assignments, you are highly encouraged for this one to reduce redundant code.

1. Continue to work on the class **Endodoi** you submitted for the first part of the assignment. If you were unable to have your first version of the Endodoi class perform all requirements correctly, or want to have help refactoring the code from your last submission, set up a time to meet with a TA to review your last submission.
2. **Write stubs for all of your methods before submitting to MIMIR**
3. The next step is to implement the **pit2Coordinates** method.

4. Implement the **getOpposingPit** method described below
5. Implement the **isPitInPlayerRow** method described below
6. Implement the **getPitSeedCount** method described below
7. Implement the **nextPit** method described below
8. Implement **singleSow** method described below

pit2Coordinates Method

This method returns the position of a pit label for a giving pit 2d array. The method returns an integer array of length 2 where the first element of the returned array is the index of the pit in the first dimension of passed pits 2d array and the second element of the returned array is the index of the pit in the second dimension of passed pits 2d array. If the parameter for the character is not present return null

Method signature

```
public static int[] pit2Coordinates(char pitLabel, int [][] playerPits)
```

Examples

```
int [] coordinates = pit2Coordinates('b', new int[][]{ {0, 0},
                                                       {0, 0} });
// the value of coordinates is {0, 1};

int [] coordinates = pit2Coordinates('d', new int[][]{ {1, 5, 7, 4},
                                                       {8, 2, 1, 3 }});
// the value of coordinates is {0, 3};
int [] coordinates = pit2Coordinates('e', new int[][]{ {1, 5, 7, 4},
                                                       {8, 2, 1, 3 }});
// the value of coordinates is {1, 0};
```

getOpposingPit Method

This method returns the label of an opposing player's pit when passed a pit label and an array of playerPits. The method returns a character for the pit label.

Method signature

```
public static char getOpposingPit(char pit, int[][] playerPits)
```

Examples

```
char label = getOpposingPit('c', new int[][]{ {0, 0}, {0, 0} });
// the value of label is 'a'

char label = getOpposingPit('d', new int[][]{ {1, 5, 7, 4},
                                              {8, 2, 1, 3 }});
// the value of label is 'h'
```

isPitInPlayerRow Method

This method determines if a labeled pit is in a player row.

Method signature

```
   public static boolean isPitInPlayerRow(int player, char pit,
                                          int[][] playersPits)
```

- int **player** is either 1 or 2 and represents the player that you want to know if the labeled pit is in their row.
- char **selectedPIt** is the label for the pit
- int [][] **playerPits** is a 2d array that represents the game pits layout

For example in a 2 X 6 board. Pit 'b' would be in Player 1's pits. Pit 'g' would be in Player's 2 pits.

```
isPitInPlayerRow(1, 'b', new int[2][2]);  // returns true
isPitInPlayerRow(2,'b', new int[2][2]);  // returns false
isPitInPlayerRow(1,'g', new int[2][6]);  // returns false
isPitInPlayerRow(2, 'g', new int[2][4]);  // returns true
```

nextPit Method

Determines the label of the next pit that a sown seed will be placed in. Reminder the movement of the board for sowing seeds is counter clockwise.

Method signature

```
public static char nextPit(char currentPit, int[][] playersPits)
```

For example in a 2 X 6 board. The following code provides examples of output

```
//Player 1  a,b,c,d,e,f
//Player 2  g,h,i,j,k,l

nextPit('h', new int[2][6]);      // 'i'
nextPit('e', new int[2][6]);     // 'd'
nextPit('l', new int[2][6]);     // 'f'
nextPit('a', new int[2][6]);     //'g'
```

getPitSeedCount Method

Given a valid pit label, returns a count of the seeds in the pit for a passed in 2d array of pits.

Method signature

```
public static int getPitSeedCount(char pit, int[][] playersPits)
```

For example in a 2 X 6 board. The following code provides examples of output

```
//Player 1  a,b,c,d,e
//Player 2  f,g,h,i,j

getPitSeedCount('a',new int[][]{{1,0,2,1,6},{5,4,7,11,8}}); // 1
getPitSeedCount('g',new int[][]{{1,0,2,1,6},{5,4,7,11,8}}); // 4
getPitSeedCount('j',new int[][]{{1,0,2,1,6},{5,4,7,11,8}}); // 8
getPitSeedCount('c',new int[][]{{1,0,2,1,6},{5,4,7,11,8}}); // 2
```

singleSow Method

This method performs one pick from a pit and then sows (put in a pit) the picked seeds one per pit in counter clockwise fashion until all picked seeds are sown. Do not perform relay sows with this method. The method returns the label of the pit that the last seed is sown into. Note this method has a side effect, where it updates the array object. Normally side effects are not a good programming practice, but not always, in this case, it is helpful for us.

Method signature

```
public static char singleSow(char pickPit, int[][] playersPits)
```

For example in a 2 X 3 board. The following code provides examples of output

```
//Player 1  a,b,c
//Player 2  d,e,f

int[][] pits = { {4,4,4},{4,4,4} };
char lastPit = singleSow('c', pits);
// the value of lastPit is 'e'
// the value of the pits array is { {5,5,0}, {5,5,4} }

//another example
int[][] pits = { {0,3,1},{2,3,5} };
char lastPit = singleSow('f', pits);
// the value of lastPit is 'e'
// the value of the pits array is { {1,4,2}, {3,4,0} }
```

Getting Started/Helpful Hints for Part II

Below are some suggestions on how to get started/work through the assignment.

1. As you develop methods, feel free to use those methods in subsequent methods that you develop. In some cases, you may call a previously developed method frequently.

2. Refactor your code, you may find that a method you've written has become redundant or some of its work is better done in another method. Look for any opportunity to reduce the amount of code you have, as it creates less opportunity for bugs. Also keep your use of global variables to as little as possible. If you can write a method without needing to use the global variables, do so.

Part III: Game Control

For the final submission, you will work on building some additional methods to support the play of the game and a method called that starts and controls the execution of the game. If you look at the point allocation, most of the points go towards implementing methods that perform small part of the program. The playGame method is meant to integrate everything together so it actually plays, the methods developed throughout the assignment can help you put the game together. If you are unable to get the game output to match/perform exactly as the test cases, understand that it is only a small part of the overall grade. Be sure to submit all that you have working by the program deadline.

1. Utilize your previous class called **Endodoi**. If you were unable to have your second version of the Endodoi class perform all requirements correctly, or want to make sure that you implemented all of your code correctly, meet with a TA to review your code.
2. **Write stubs for all of your methods before submitting to MIMIR**
3. The next step is to implement the **checkEndGame** method described below
4. Implement the **totalSeedsOnBoard** method described below
5. Implement the **clearBoard** method
6. Implement the **playGame** method

checkEndGame Method

This method checks to see if the game is over. Review the rules above for the conditions for ending the game. All the information that you need to make the decision of whether or not to end the game is in this method's parameters. The method returns true if the game is over and false if is not.

Method signature

```
public static boolean checkEndGame(int[][] playersPits, boolean isPlayer1Turn)
```

- int [][] **playerPits** is a 2d array that represents the game pits with seed counts
- boolean **isPlayer1Turn** is true if it's player 1's turn and false if it's player's 2 turn.

For example in a 2 X 6 board. The following code provides examples of output

```
//Player 1 a,b,c
//Player 2 g,h,i

int [][] pits = new int[][]{{0,1,0},{0,0,0}};
checkEndGame(pits,true)     // player 1s turn returns false
checkEndGame(pits,false)    // player 2s turn returns true


//Player 1 a,b,c
//Player 2 g,h,i

int [][] pits = new int[][]{{2,1,4},{3,1,5}};
checkEndGame(pits,true)     // player 1s turn returns false
checkEndGame(pits,false)    // player 2s turn returns false
```

totalSeedsOnBoard Method

This method returns a count of the total seeds on the board.

Method signature

```
public static int totalSeedsOnBoard(int [][] playersPits)
```

- int [][] **playerPits** is a 2d array that represents the game pits with seed counts

For example in a 2 X 3 board. The following code provides examples of output

```
//Player 1 a,b,c
//Player 2 g,h,i

int [][] pits = new int[][]{{1,1,0},{5,2,1}};
totalSeedsOnBoard(pits);  // returns value of 10
```

clearBoard Method

This method sets all arrays in the pits array to 0. It is meant to represent removing all seeds from the board.

Method signature

```
public static void clearBoard(int [][] playersPits)
```

- int [][] **playerPits** is a 2d array that represents the game pits with seed counts

For example in a 2 X 3 board. The following code provides examples of output

```
//Player 1 a,b,c
//Player 2 g,h,i

int [][] pits = new int[][]{{0,1,2},{3,1,3}};
clearBoard(pits);

// the value of pits is {0,0,0},{0,0,0}}
```

playGame Method

This method controls the entire game and calls the other methods created during this assignment to run the game. It prompts for and then receives input from the user and outputs the results of each play of the game.

Method signature

```
public static void playGame(int[][] playersPits,int player1Store,
                            int player2Store, boolean isPlayer1Turn)
```

- int [][] **playerPits** is a 2d array that represents the game pits with seed counts at the start of the game.
- int **player1Store** keep track of the seeds in player 1's store
- int **player2Store** keep track of the seeds in player 2's store
- boolean **isPlayer1Turn** is true if it's player 1's turn and false if it's player's 2 turn.

Requirements

- At each turn, the current state of the board is displayed with the player's turn indicated, the player whose turn it is enters a character
    - If the pick is valid for the player, then the game relay sows their selection. The state of the game board is displayed with the messages shown in output for each single sow until the player's turn ends
    - If an invalid pit is picked or an empty pit is picked an appropriate message is displayed and the player is prompted to re-enter a selection
    - The game can be ended early by entering the '!' character.
    - The game automatically updates a player's store at the end of the turn if they captured seeds
    - The game automatically handles determining when the game is over and declaring the winner. The labels to print out are:

```
//If player 1 is the winner (player 1 has more seeds in their store)
Player 1 is the winner!

//If player 2 is the winner (player 2 has more seeds in their store)
Player 2 is the winner!

//If it is a tie (meaning both player's stores were equal)
It was a draw!
```

    - The asterix for a label is meant to indicate the ending pit of sowing seeds

Full Game Example

```
    public static void main(String[] args) {
         int [][] playerPits = createNewPits(3,2);
         playGame(playerPits,3,2,true);
    }
```

```
Starting game of Endodoi
 +====++==++----+----+----+
1|( 3)||\/||( 2)|( 2)|( 2)|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||( 2)|( 2)|( 2)|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1 choose a pit:
c
 +====++==++----+----+----+
1|( 3)||\/||( 3)|( 3)|(  )|
 | st ||/\||  * |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||( 2)|( 2)|( 2)|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 1. Resowing seeds...
 +====++==++----+----+----+
1|( 3)||\/||(  )|( 3)|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||( 3)|( 3)|( 3)|
 | st ||  ||    |    |  * |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 2. Resowing seeds...
 +====++==++----+----+----+
1|( 3)||\/||( 1)|( 4)|( 1)|
 | st ||/\||  * |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||( 3)|( 3)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1's turn ended.
Player 1 captured 4 seeds
 +====++==++----+----+----+
1|( 7)||  ||(  )|( 4)|( 1)|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|( 3)|(  )|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Player 2 choose a pit:
e
 +====++==++----+----+----+
1|( 7)||  ||(  )|( 5)|( 2)|
 | st ||  ||    |  * |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|(  )|( 1)|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 1. Resowing seeds...
 +====++==++----+----+----+
1|( 7)||  ||( 1)|(  )|( 3)|
 | st ||  ||    |    |  * |
 +====++==++----+----+----+
2|( 2)||\/||( 1)|( 1)|( 2)|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 2. Resowing seeds...
 +====++==++----+----+----+
1|( 7)||  ||( 2)|( 1)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||( 2)|( 1)|( 2)|
 | st ||/\||  * |  e |  f |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 3. Resowing seeds...
 +====++==++----+----+----+
1|( 7)||  ||( 2)|( 1)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|( 2)|( 3)|
 | st ||/\||  d |  e |  * |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 4. Resowing seeds...
 +====++==++----+----+----+
1|( 7)||  ||( 3)|( 2)|( 1)|
 | st ||  ||  * |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|( 2)|(  )|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 5. Resowing seeds...
 +====++==++----+----+----+
1|( 7)||  ||(  )|( 2)|( 1)|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||( 1)|( 3)|( 1)|
 | st ||/\||  d |  e |  * |
 +====++==++----+----+----+
Player 2's turn ended.
Player 2 captured 2 seeds
 +====++==++----+----+----+
1|( 7)||\/||(  )|( 2)|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 4)||  ||( 1)|( 3)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1 choose a pit:
b
 +====++==++----+----+----+
1|( 7)||\/||( 1)|(  )|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 4)||  ||( 2)|( 3)|(  )|
 | st ||  ||  * |    |    |
 +====++==++----+----+----+
Sowing ended in non-empty pit. Multi-lap 1. Resowing seeds...
 +====++==++----+----+----+
1|( 7)||\/||( 1)|(  )|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 4)||  ||(  )|( 4)|( 1)|
 | st ||  ||    |    |  * |
 +====++==++----+----+----+
Player 1's turn ended.
 +====++==++----+----+----+
1|( 7)||  ||( 1)|(  )|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 4)||\/||(  )|( 4)|( 1)|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Player 2 choose a pit:
f
 +====++==++----+----+----+
1|( 7)||  ||( 1)|(  )|( 1)|
 | st ||  ||    |    |  * |
 +====++==++----+----+----+
2|( 4)||\/||(  )|( 4)|(  )|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Player 2's turn ended.
 +====++==++----+----+----+
1|( 7)||\/||( 1)|(  )|( 1)|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 4)||  ||(  )|( 4)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1 choose a pit:
c
 +====++==++----+----+----+
1|( 7)||\/||( 1)|( 1)|(  )|
 | st ||/\||  a |  * |  c |
 +====++==++----+----+----+
2|( 4)||  ||(  )|( 4)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1's turn ended.
Player 1 captured 5 seeds
 +====++==++----+----+----+
1|(12)||  ||( 1)|(  )|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 4)||\/||(  )|(  )|(  )|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Player 2 can't move.
The remaining 1 seeds go to Player 1
 +====++==++----+----+----+
1|(13)||  ||(  )|(  )|(  )|
 | st ||  ||  a |  b |  c |
 +====++==++----+----+----+
2|( 4)||  ||(  )|(  )|(  )|
 | st ||  ||  d |  e |  f |
 +====++==++----+----+----+
Player 1 is the winner!
```

The following code provides examples of output for a game that isn't from a starting state, this is one way that you can see if your game is working without having to implement an entire game.

```
   public static void main(String[] args) {
        int [][] playerPits = new int[][]{{0,1,0},{1,1,0}};
        playGame(playerPits,3,2,true);
    }
```

```
Starting game of Endodoi
 +====++==++----+----+----+
1|( 3)||\/||(  )|( 1)|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||( 1)|( 1)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1 choose a pit:
b
 +====++==++----+----+----+
1|( 3)||\/||( 1)|(  )|(  )|
 | st ||/\||  * |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||( 1)|( 1)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1's turn ended.
Player 1 captured 2 seeds
 +====++==++----+----+----+
1|( 5)||  ||(  )|(  )|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|( 1)|(  )|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Player 2 choose a pit:
e
 +====++==++----+----+----+
1|( 5)||  ||(  )|(  )|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|(  )|( 1)|
 | st ||/\||  d |  e |  * |
 +====++==++----+----+----+
Player 2's turn ended.
 +====++==++----+----+----+
1|( 5)||\/||(  )|(  )|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||(  )|(  )|( 1)|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1 can't move.
The remaining 1 seeds go to Player 2
 +====++==++----+----+----+
1|( 5)||  ||(  )|(  )|(  )|
 | st ||  ||  a |  b |  c |
 +====++==++----+----+----+
2|( 3)||  ||(  )|(  )|(  )|
 | st ||  ||  d |  e |  f |
 +====++==++----+----+----+
Player 1 is the winner!
```

Example of Other Commands

```
   public static void main(String[] args) {
        int [][] playerPits = new int[][]{{1,1,0},{0,1,0}};
        playGame(playerPits,3,2,false);
    }
```

```
Starting game of Endodoi
 +====++==++----+----+----+
1|( 3)||  ||( 1)|( 1)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|( 1)|(  )|
 | st ||/\||  d |  e |  f |
 +====++==++----+----+----+
Player 2 choose a pit:
a
Not a valid pit. Please pick again.
Player 2 choose a pit:
w
Not a valid pit. Please pick again.
```

```
Player 2 choose a pit:
d
Can't pick from an empty pit. Please pick again.
Player 2 choose a pit:
e
 +====++==++----+----+----+
1|( 3)||  ||( 1)|( 1)|(  )|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
2|( 2)||\/||(  )|(  )|( 1)|
 | st ||/\||  d |  e |  * |
 +====++==++----+----+----+
Player 2's turn ended.
 +====++==++----+----+----+
1|( 3)||\/||( 1)|( 1)|(  )|
 | st ||/\||  a |  b |  c |
 +====++==++----+----+----+
2|( 2)||  ||(  )|(  )|( 1)|
 | st ||  ||    |    |    |
 +====++==++----+----+----+
Player 1 choose a pit:
!
Game ended
```

Getting Started/Helpful Hints for Part III

Below are some suggestions on how to get started/work through the assignment.

1. Build on the work from the Endodoi starter lab, use the setBoard method to create different board configurations.
2. Create Unit tests of your code for grading will use a similar method to test functionality.

Submission for Part III

Submit your **Endodoi.java** file to **Mimir** before the deadline. You may submit more than one time, the last attempt before the deadline is the one that is graded.

Before uploading your program, be sure to complete the following steps carefully:

- Your input and output should match the input and output test cases exactly, character for character.

Assessment

Criteria	**Points**
Endodoi.java submitted with at least 50 lines of code	1
Program compiles without error	2
Program passes Checkstyle	2
Passes checkEndGame tests	6
Passes totalSeedsOnBoard tests	6
Passes clearBoard tests	6
Passes playGame tests	18
Passes methods in Part I	3
Passes methods in Part II	6
Total	**50**