Assignment 4, Part 2

Program Structure Hints
public boolean makeMove()

- makeMove doesn't receive any arguments, but recall that since it extends TicTacToe, we have access to:
  - TicTacToeArray
  - step
  - winner
  - player

  And all the methods of TicTacToe, including:
    updateTTT(char sym, int row, int col)
public boolean makeMove()

- First, create the two main arrays, and fill them with zeros.

```
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0

  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
```

defensiveOppsArray  offensiveOppsArray

- Our goal is to populate these arrays, (one element at a time) sum them together, and select the max.
public boolean makeMove()

- Recall: the cells of defensiveOppsArray and offensiveOppsArray correspond to the defensive and offensive value of playing that position for the next move.
- We consider each cell (row,col) independently, which means we need a double loop.

![TicTacToeArray Diagram]
public boolean makeMove()

• For each cell (inside the double loop):
  1. If the cell is already played, skip it and continue on to the next cell.

Skip these. Their values in the OppsArrays should be left at 0.
public boolean makeMove()

• For each cell (still in the double loop):
  
  2. If the cell is on left-leaning diagonal, do this:
    
    • Copy the diagonal into a path array
    • Call assessDefensiveOpps and assessOffensiveOpps on the path.
    • Add the return values to the OppsArrays.

char[] path = { *, O, *, O }

int v = assessDef...(path,'X')

...
public boolean makeMove()

• For each cell (still in the double loop):

  3. If the cell is on right-leaning diagonal, do this:
  • Copy the diagonal into a path array
  • Call assessDefensiveOpps and assessOffensiveOpps on the path.
  • Add the return values to the OppsArrays.

```
char[] path = { *, X, *, X }
int v = assessDef...(path,'X')
```

Hint, might need a loop to build the path.
public boolean makeMove()

• For each cell (still in the double loop):

  4. Always do this:
  • Copy the cell's row into a path array
  • Call assessDefensiveOpps and assessOffensiveOpps on the path.
  • Add the return values to the OppsArrays.

Hint, might need a loop to build the path.

TicTacToeArray

<table>
<thead>
<tr>
<th>*</th>
<th>X</th>
<th>O</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>O</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>X</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>O</td>
</tr>
</tbody>
</table>

char[] path = { *, *, *, O }

int v = assessDef...(path,'X')

v

...
public boolean makeMove()

- For each cell (still in the double loop):
  5. Always do this:
     - Copy the cell's column into a path array
     - Call assessDefensiveOpps and assessOffensiveOpps on the path.
     - Add the return values to the OppsArrays.

```java
char[] path = { *, *, *, * };
int v = assessDef...(path,'X');
v = assessDef...(path,'X');
```

Hint, might need a loop to build the path.
public boolean makeMove()

• Now you should have fully populated OppsArrays.

• Here would be a good place to print out your OppsArrays to make sure they match the web simulator.
  – If you do this, be sure to comment it out before submitting, or you'll lose points! makeMove shouldn't print anything in your final submission.
public boolean makeMove()

• Walk over the two arrays. The best move is the (row,col) value where the sum of defensiveOppsArray[row][col] and offensiveOppsArray[row][col] is maximized.
  – (Hint: requires a double loop, and some state variables to keep track of the max value and coordinates).
• If both arrays are full of zeros, then there is no best move. Return false.
• Otherwise, call updateTTT to play the move and return true. If two moves tie, play the 1st occurrence of the tie in a row-major scan of the array.
  – Hint: row major means your outer loop walks over the rows, the inner loop walks over the columns, as shown on slide 4.
public void int assessDefensiveOpps(char[] path, char sym)

• Count the number of opponents in the path by walking over the path array. (requires a loop)
  – If at any point you encounter your own piece (sym), then return 0 because the path is already blocked.

• Now that you know how many opponents are in the path, a simple if statement will determine if this is a critical move.

• At this point, a simple one-line mathematical expression should give you your return value.
public void int
assessOffensiveOpps(char[] path, char sym)

• Very similar to assessDefensiveOpps.
Assignment 4, Part 3
Using ArrayList to implement a memory
Goal

- Computing the `defensiveOppsArray` and `offensiveOppsArray` takes time.
- Perhaps we can optimize our code by adding the concept of memory to our program.
- If a board state has been encountered before in a previous game, we don't need to recompute the OppsArrays if we saved them somewhere (ie. “memory”).
- The goal of Part 3 is to implement such a memory.
ArrayList

- Recall that an ArrayList is an array that can grow dynamically.
- We add items to an ArrayList by calling the add method.
- The type of data an ArrayList can hold is specified in <>.
  - Example:
    
    ```java
    ArrayList<String> al = new ArrayList<String>();
    al.add("hello");
    al.add("world");
    System.out.println(al.get(1)); // prints world
    ```
ArrayList for Memory

• How can an ArrayList be used to implement a memory?
  – Store all the previously encountered board states, and their corresponding computed OppsArrays.

• First we need to create a data type.

  ```java
  public class BoardState{
    public String TTTState;
    public int[][] defensiveOppsArray;
    public int[][] offensiveOppsArray;
  }
  ```

• Now we can do: `new ArrayList<BoardState>()`
Saving to Memory

- To store an item in memory, just create an instance of BoardState and add it.
  
  ```java
  BoardState addMe = new BoardState();
  addMe.offensiveOppsArray = offensiveOppsArray;
  addMe.offensiveOppsArray = offensiveOppsArray;
  ```

- But wait, TTTState is a String, and the game board is a char[][][]. Why?

- Solution is to create a method that converts the char[][][] into a String array.

- Now we can do:
  
  ```java
  addMe.TTTState = convertTTTArrayToString();
  ```
Using the Memory

• To check for a board state in memory, just walk over all the elements in the ArrayList and compare their TTTState to the current board's TTTState.
  – This check should be done before building the OppsArrays.

• If you do have to build the OppsArrays, make sure to save them to memory immediately after.
Example:

This program asks a user to enter a string, and keeps track of how many times they enter each string.

```java
import java.util.ArrayList;

public class ArrayListDemo{
    public static void main(String[] args){
        ArrayList<StringEncounter> al =
            new ArrayList<StringEncounter>();
        for(;;){
            //query the user to enter a string
            System.out.print("Enter a string: ");
            String x = UserInput.readString();
            System.out.println("\n");
            //check memory to see if we've typed that before
            boolean found = false;
            for(int i=0; i < al.size(); i++){
                StringEncounter test = al.get(i);
                if(test.str.equals(x)){
                    System.out.println("You've typed that "+
                        test.count+" times.");
                    test.count++;
                    found = true;
                }
            }
            //if this was the first time we typed that string, add it to the list
            if(!found){
                System.out.println("Looks like the"+
                    " first time you've typed that.");
                StringEncounter se = new StringEncounter();
                se.str = x;
                se.count = 1;
                al.add(se); //add it to the memory
            }
        }
    }
}

public class StringEncounter{
    public String str;
    public int count;
}
```