Tic-Tac-Toe

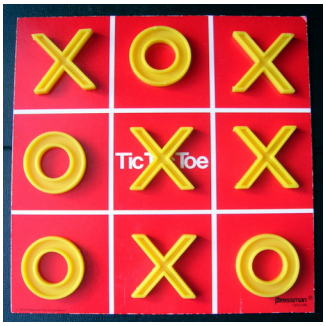
Team Name:

Manager:

Recorder:

Presenter:

Analyst:



This is a Process Oriented Guided Inquiry Learning (POGIL) activity. You and your team will examine a working program. A series of questions will guide you through a cycle of exploration, concept invention, and application. There is strong evidence that this is more effective (and less boring) than a traditional lecture.

By the time you are done with this activity, you and your team should be able to:

- decompose a program into static methods with Javadoc comments.
- use multidimensional arrays.
- manage your team more effectively.

Your team's recorder is responsible for writing your team's answers to the numbered questions on this form.

The image above is from boardgamegeek.com.

Playing the game

Open the Tic-Tac-Toe module. Run TicTacToe.java to play the game. This is a two-player game, so your manager will have to divide your team into pairs to play.

1. Which members of your team were paired to play together?

2. After these games are done, form different pairs and play a second time. Which members of your team were paired?

3. Is everyone done playing and ready to pay attention to the team?

You may need to go back and play the game again to answer some of the questions to come, but you should do so *deliberately*, because your team's manager assigned one or more people to find something out, not merely because you got bored with the conversation or thought you could answer a question better on your own.

4. What happens if you try to make an illegal move, e.g., click on an occupied square?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

5. How else might a program react to an illegal move? Do you think one of the other options is better? Why?

Static methods and Javadoc comments

Examine TicTacToe.java. This program is broken down into multiple *methods* (functions), each of which starts with the keywords `public static`. Find the `main` method.

6. What do you think the line below does?

```
draw(board, currentPlayer);
```

7. If you place the cursor in the word `draw` on the line above in the program and hit control-j (Mac) or control-q (Windows or Linux), quick documentation pops up. From where (in this program) did the text describing the draw method come?

8. Does editing that text change what appears in the quick documentation?

9. Does the quick documentation still use that text if the comment starts with `/*` instead of `/**`?

10. What causes the loop below to end?

```
while (!gameOver(board)) {
    handleMouseClick(board, currentPlayer);
    currentPlayer = opposite(currentPlayer);
    draw(board, currentPlayer);
}
```

11. What do you think the first statement inside the loop does?

12. What do you think the second statement inside the loop does?

13. How does the behavior of the game change if you remove that second statement? (Hint: in IntelliJ IDEA, you can use command-/ (Mac) or control-/ (Widows or Linux) to comment or uncomment one or more selected lines.)

14. How did your team experiment to answer the previous question? Did just one person modify the program or did everyone do it? Why did you do it that way?

15. Do you feel that you understand how this program works at a high level?

16. Do you feel that you understand the details of how things like `handleMouseClick` and `opposite` work?

Let's examine the behavior of one of the other methods. Add the following line at the beginning of `main` (right after the line that begins `public static void main`):

```
StdOut.println(opposite('X'));
```

17. What is printed when you run the program with this modification? (Note that `StdOut.println` prints to the console, not in the graphic window.)

18. What is the value of `opposite('O')`?

Now examine the code for the `opposite` method.

19. Within the method, `opposite` uses an if statement to determine its answer. What does it do with the answer? Does it store it in a variable? Print it? Something else?

20. If you temporarily change the first line of the definition of `opposite` from

    ```
    public static char opposite(char player) {
    ```

    to

    ```
    public static void opposite(char player) {
    ```

    several red underlines appear in the code, indicating that it no longer compiles. If you hold your mouse over the first red underline within `opposite`, a pop-up window appears. What does the first line in that window say?

21. In general, how does a method indicate that it is giving back an answer?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

22. What are the advantages and disadvantages of breaking a program down into multiple methods (instead of having everything in the `main` method)?

Multidimensional arrays

23. Has your team been staying on task?

24. After the line

```
char[][] board = new char[3][3];
```

    (in the `main` method) what is the value of `board.length`?

25. What is the value of `board[0].length`? Does this change if the two numbers on the line in the previous question are different?

26. What do the nested loops below accomplish?

```
for (int x = 0; x < board.length; x++) {
    for (int y = 0; y < board[x].length; y++) {
        board[x][y] = ' ';
    }
}
```

The `winner` method contains the following statement:

```
int[][][] lines = { { { 0, 0 }, { 0, 1 }, { 0, 2 } },
                    { { 1, 0 }, { 1, 1 }, { 1, 2 } },
                    { { 2, 0 }, { 2, 1 }, { 2, 2 } },
                    { { 0, 0 }, { 1, 0 }, { 2, 0 } },
                    { { 0, 1 }, { 1, 1 }, { 2, 1 } },
                    { { 0, 2 }, { 1, 2 }, { 2, 2 } },
                    { { 0, 0 }, { 1, 1 }, { 2, 2 } },
                    { { 0, 2 }, { 1, 1 }, { 2, 0 } } };
```

27. After this statement, what is `lines.length`?

28. What is `lines[0].length`?

29. Which part of the statement (some of the curly braces, commas, and numbers) represents `lines[0]`?

30. What is `lines[0][0].length`?

31. Which part of the statement represents `lines[0][0]`?

32. What does each pair of numbers represent?

33. What does each group of three pairs represent?

34. Has your team been managing time effectively? Have you been rushing through things, lingering too long on some questions, or getting distracted?

The `winner` method continues with the following for loop:

```
for (int i = 0; i < lines.length; i++) {
    int[][] line = lines[i];
    char a = board[line[0][0]][line[0][1]];
    char b = board[line[1][0]][line[1][1]];
    char c = board[line[2][0]][line[2][1]];
    if (a != ' ' && a == b && b == c) {
        return a;
    }
}
```

35. What does each pass through the loop accomplish?

36. Under what circumstances does the loop exit early?

37. Under what circumstances does the loop complete all of its passes, allowing the program to proceed to the last line of `winner`?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have. If your team is done before other teams, discuss the following open-ended question:

38. Would the code be more or less clear if it had a separate if statement for each of the eight possible winning lines? Would it be longer or shorter? Would your answer be different for a game that had a hundred ways to win instead of eight?

This work was supported by the Google Education and Relations Fund's CS Engagement Small Grant Program grant #TFR15-00411.

The principal investigator, Peter Drake, wishes to thank the following for their useful comments: the members of the CS-POGIL project, specifically Clif Kussmaul and Helen Hu; Maggie Dreyer; and various anonymous reviewers.