

## CSSE 290 Exam 3 February 18-19, 2015 (Session 29)

You may use your textbook, programs that you have written (not programs written by other students in this course from any term), reference websites such as w3schools.com, php.net, webstepbook.com. You may not search for programs that solve something like the specific problems on this exam. You may not talk with anyone else about web programming until after you have submitted your final solution. After that you may not talk to other students in the course about web programming unless they have also submitted the exam.

By submitting the exam electronically on **wwwuser**, you indicate your agreement to the following statement:

I have not communicated (electronically or in person) with anyone other than the instructor about the problems on this exam or about techniques needed to solve the problems.

**Due Thursday at 8 AM.** If you come to Thursday's class meeting on time, you get an automatic extension until Friday at 8 AM.

**How long should it take to do this problem?** I recommend that you attempt you attempt to get Part 1 working on Wednesday. If you don't succeed, you can have an idea of how much time you might need to set aside on Thursday. My expectation is that most students will need 4-6 hours for part 1, and 30-60 minutes for part 2, but there will be a lot of variation; some good programmers are simply a lot slower than others (I think I am a good programmer, but I am not very fast!). Perhaps a good estimate is "about the same amount of time as you needed for HW5". The level of complexity of this problem is similar to that one.

**To submit:** Copy the entire **Exam3** folder to your **webProgramming** folder on **wwwuser**. If you do the extra credit problem, you will need to add a table to your database on the **wwwuser** server. The easiest way to do that is to export the table using **phpmyadmin** on your server, then import it using **phpmyadmin** on **wwwuser**.

**If you have questions about the specifications**, post them to Piazza. I will try to answer quickly. Because I was up very late Tuesday night and up again early this morning putting this exam together, I am unlikely to answer any questions between 9:15 PM Wednesday and 5:00 AM Thursday. If I need to make announcements related to the exam, I will use Piazza.

## Dots and Boxes game [http://en.wikipedia.org/wiki/Dots\\_and\\_Boxes](http://en.wikipedia.org/wiki/Dots_and_Boxes)

The first few paragraphs of the above web page explain the simple rules of this game.

Next you should watch my video, also linked from the schedule page. The first part of the video demonstrates the program, and the second part is a quick tour of the provided code. The rest of this document will assume that you have seen the video. The video was done without a lot of editing, since it is "one time use." Hopefully it is clear. If not, there is Piazza!

"Looking pretty" is not a major goal for this program; the emphasis is on functionality. There is one annoying part: not being able to click directly between two dots (see the video). Here I chose "ease of implementation" over "ease of use," since you have limited time to do the implementation.

**Part 1** (35 points). JavaScript program. It is only necessary to modify the **dots.html** or **dots.css** files, although you are allowed to modify any of the files if you wish. Be very careful about changing the rules for square, dot, and row in the **dots.css** file; it took a lot of tweaking to get things to line up nicely.

The first three lines of **dots.js** contain parameters for the number of players, rows, and columns. Your program should work for two-to-four players, three-to- six rows, and three-to-six columns. When we test your code, we will try different values.

The first thing you'll need to do is to "fix" the user color key at the top of the page and to generate the dot and square spans that make up the board. For each square span, I recommend giving it an id that indicates where it is on the board. For example, you might give the square in row 2, column 3 might have id s2\_3.

Perhaps the trickiest part of the code is figuring out which border (top, bottom, left, or right) to enable after a user click. And then if that border is between two squares, the corresponding border of the adjacent square must also be added. I illustrate this early in the video. Using some of the variables and functions that I provided may help you do this elegantly.

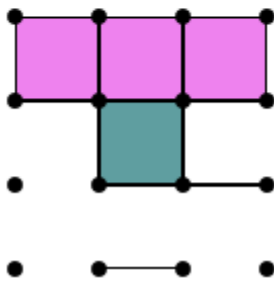
You'll need to get the x and y coordinates of a click relative to the top left corner of the square where the click happens. The following code will get you the correct values in most browsers:

```
var x = e.hasOwnProperty('offsetX') ? e.offsetX : e.layerX;  
var y = e.hasOwnProperty('offsetY') ? e.offsetY : e.layerY;
```

## Dots game

Mary 3 John 1

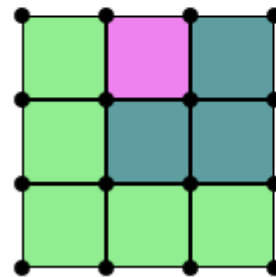
John's turn



## Dots game

Mary 1 John 3 Alicia 5

Alicia IS THE WINNER!



The **sample.html** file is there to illustrate the kinds of things your Javascript code will need to set in order to get border elements and filled-in squares. Compare the code to the output you get when you double-click it. This file is for illustration only, and is not really part of the program.

**Part 2** (10 points). Use a PHP program to allow the user to specify the three size parameters. I have provided **index.html** which displays the form and commits to **dots.php**.



## Dots Game Setup

Number of players: ☒ 2 ☐ 3 ☐ 4

Number of rows of squares: ☐ 3 ☒ 4 ☐ 5 ☐ 6

Number of columns of squares: ☐ 3 ☒ 4 ☐ 5 ☐ 6

The **dots.php** file should return to the browser mostly the **same code** as in dots.html in part 1. **Dots.php** from part 2 should be only slightly different from B in part 1. The main difference is that it must include the values of the three GET parameters, so that the slightly modified **dots.js** program can use those parameters to initialize **numPlayers**, **ROWS**, and **COLS**.

After the user selects radio buttons and submits the form in index.html, the program should behave just as in part 1.

**Part 3 – extra credit** (10 points). This optional part allows the user to save an “in-progress” game in a database. The **index.html** file has a second form that allows the user to enter the name of a saved game. The program then retrieves that game from the database and displays it. The page that displays the board should have a form that allows for naming and saving a game’s state. You can design the interface and the database table. No login is required.