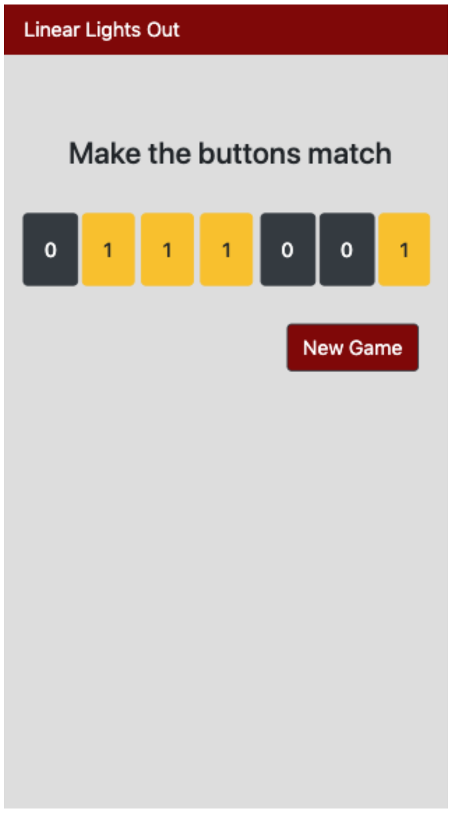
Linear Lights Out Homework

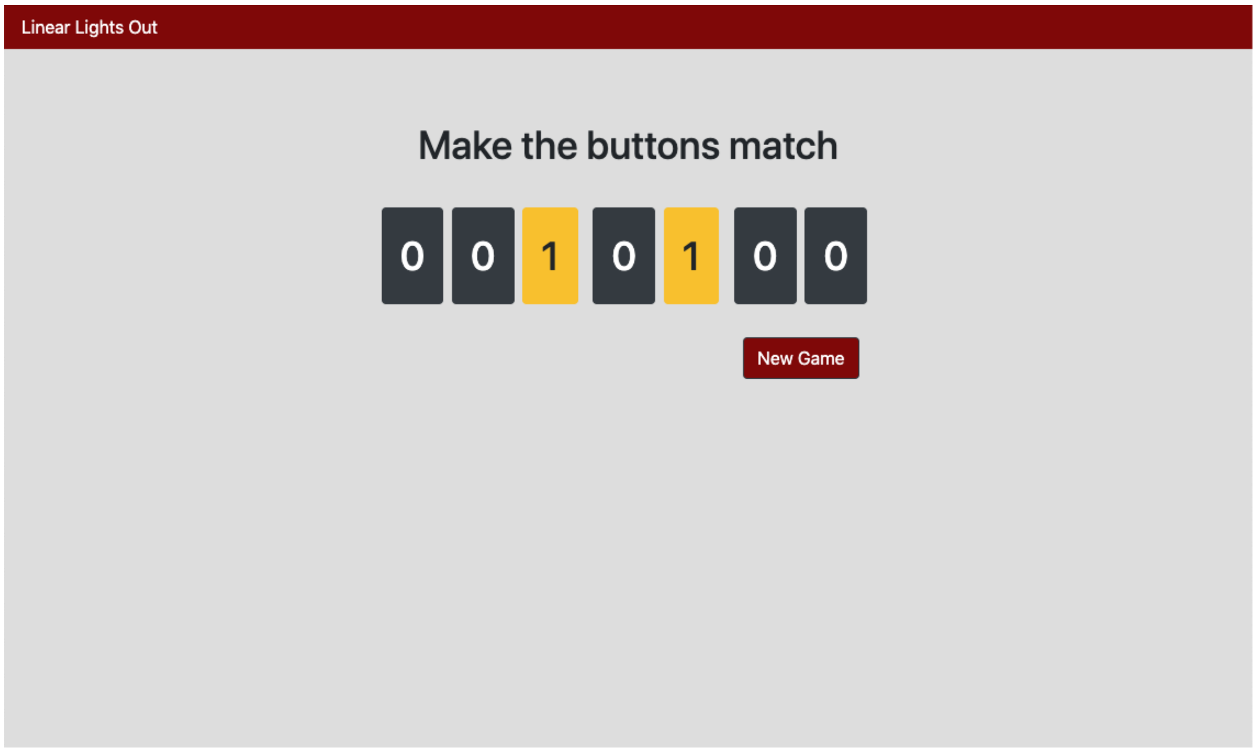
Goal

In this homework you'll practice making a game on your own that is similar to the skills you learned for Tic-Tac-Toe. This game is called Linear Lights Out. Much like Tic-Tac-Toe it will have some CSS + HTML challenges and then some JavaScript challenges to make the model object and controller. You should use 2 classes, just like Tic-Tac-Toe, called rhit.Game (or rhit.LightsOutGame) and rhit.PageController, following the pattern you learned in the lecture. The game should look like this:

Mobile



Desktop:



If you look **super** close and think about it, the fonts are bigger on the desktop. It's hard to see in the images since the mobile image is scaled bigger, but the fonts changed! I did this to make you implement some media queries. Here is what you need to do with fonts.

- @ 576 pixels make the fonts (game state and buttons) roughly 50% bigger
- @ 992 pixels make the fonts (game state and buttons) roughly 100% bigger

In the video lecture we showed you how to use Bootstrap, flexbox, and grid. This homework is not a grid, so you need to use flexbox. You are not allowed to use Bootstrap for the main page layout. You must use flexbox (for practice).

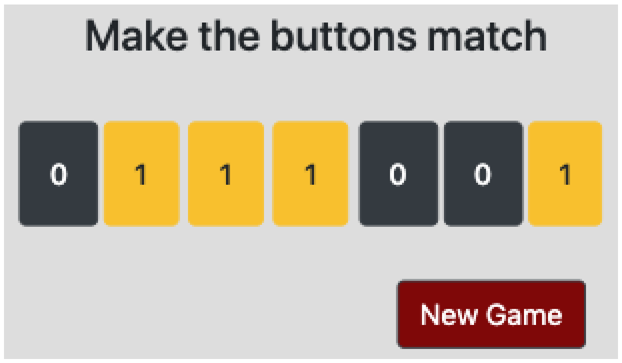
You should also use the colors as shown, notice that when a box is 1 it is yellow (light on) and when it is 0 it is dark (light off).

Here is how the game Linear Lights Out works…

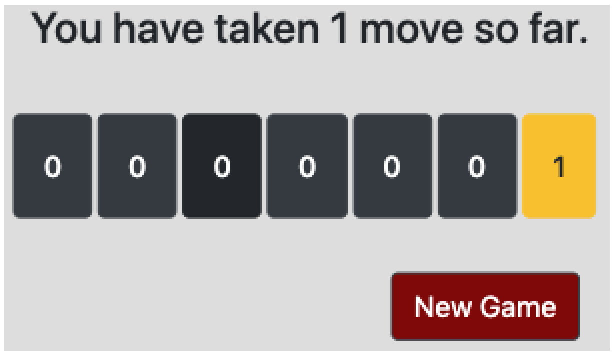
- Players are trying to make all the buttons be the same (in a real game it's "lights out", but we've decided to count all on or all off as a win to make it easier).
- Clicking a button changes that button **and both neighbors** (or just itself and 1 neighbor for the edge buttons).

Example:

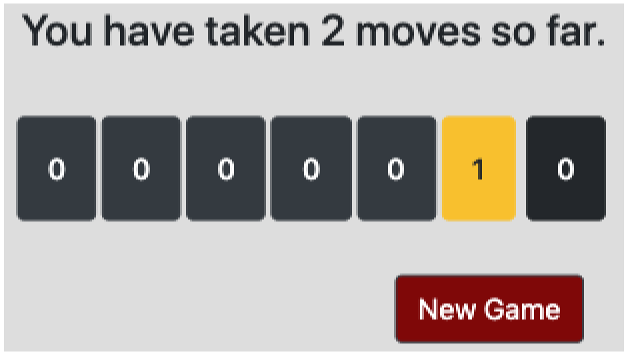
If you started the game with this board (you will need to figure out how to randomly create a winnable game):



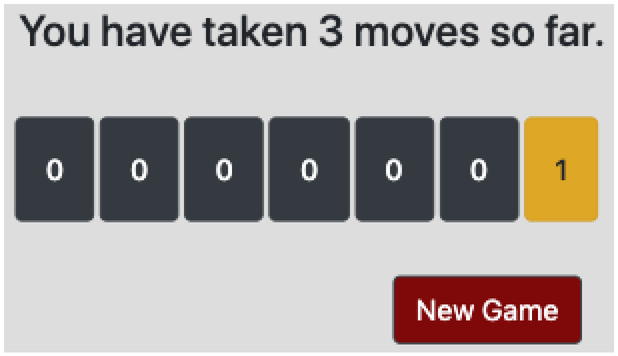
Then you click the 3rd button (index=2, the middle 1 in the group of 3), it changes buttons 2, 3, and 4 (indexes 1, 2, 3). Notice how all 3 lights changed states. Also notice that the text changed AND the button colors changed (see below).



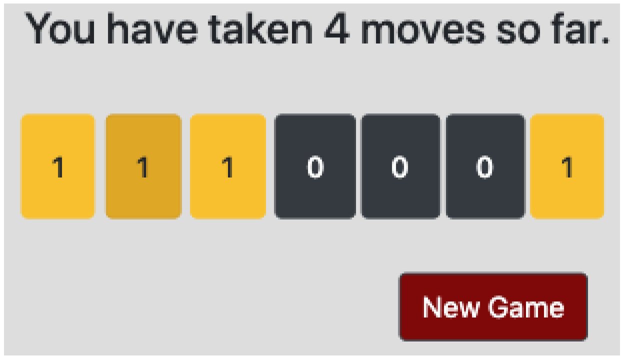
Then being a new player you decide to click the single 1 that is showing. Unfortunately if you click the 1, this will happen...



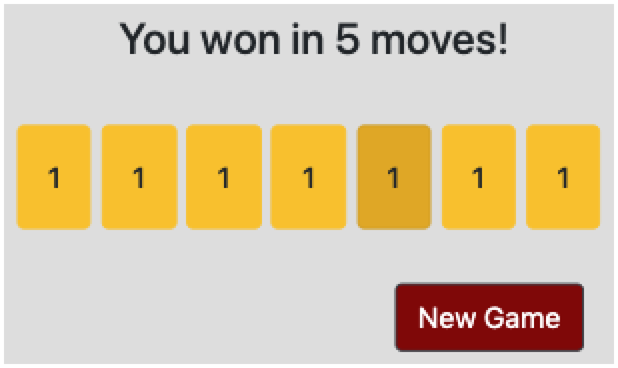
Ugh! So to try to win this game I'll undo that move by clicking the far right button again, which is an easy way to fix a mistake (at the cost of a move).



Instead I decide to click button 2 (index = 1). Seems odd at first, but a much better plan.



then button 5 (index = 4) for the win!



You win if they all match! All on or all off.

Notice how the game state shows players the number of moves at all times and then shows the number of moves you used to win the game.

When you setup this project in firebase feel free to use the format “yourusername-lightsout” which will make your url become:

yourusername-lightsout.web.app

When you complete this homework, you need to deploy it to firebase and submit the .zip to Moodle.

That is really all you need to know to implement this homework. You can stop reading now. The rest of this document is just hints and advice that you can optionally look at (or not) if you get stuck.

Good luck!

Free Hints for the CSS / Layout

In Tac-Tac-Toe we first did the layout with Bootstrap, then flexbox, and then grid. To be honest, this layout using flexbox is easier. Look at your commented out flexbox solution from the video. Use that as motivation and don't forget the CSS flexbox rules. Rewatch that video if you like. Also this url is a great flexbox "cheat sheet"

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Free Hints for JavaScript

Most of the challenges here are in the JavaScript classes. Follow the format of Tic-Tac-Toe. For example I made a game class with an enum for the different button states and created instance variables for tracking the game state (via a counter not an enum). Really the enum could just be a bool, but I like enums.

```
rh.LightsOutGame = class {
    static NUM_BUTTONS = 7;
    static LIGHT_STATE = {
        ON: "1",
        OFF: "0"
    }

    constructor() {
        this.buttonValues = [];
        this.numPresses = 0;
```

(Optional item) One thing that can be tricky about Linear Lights Out is that not all game states are winnable (interesting fact, but annoying to you really). So if you want to **make sure** a game is 100% winnable, it works well to start from a win state then randomly click buttons to scramble the board. For example I did this....

```
    constructor() {
        this.buttonValues = [];
        this.numPresses = 0;
        for (let k = 0; k < rhit.LightsOutGame.NUM_BUTTONS; k++) {
            this.buttonValues.push(rhit.LightsOutGame.LIGHT_STATE.OFF);
        }
        this.randomizeButtons();
    }

    randomizeButtons() {
        // Start with a win and randomly press buttons
        for (let k = 0; k < rhit.LightsOutGame.NUM_BUTTONS * 10; k++) {
            this.pressButtonAtIndex(Math.floor(Math.random() *
this.buttonValues.length), true)
        }
        if (this.isGameWon()) {
            this.pressButtonAtIndex(Math.floor(Math.random() *
this.buttonValues.length), true)
        }
        this.numPresses = 0;
    }
```

I passed in a value of true to my pressButtonAtIndex function so that he would know I'm doing the setup and shouldn't be checking for a win (that argument name is **isSetup**). If you want to learn something I made that argument be an optional argument…

```
    pressButtonAtIndex(buttonIndex, isSetup = false) {
```

Again there are PLENTY of ways to do it. Making them just random works fine too, but I wanted to be 100% sure I didn't make an unwinnable game. Making the rest of the class is a good programming challenge so I'll stop here with the free hints. Try to follow the format of Tic-Tac-Toe to help you out for things like the PageController and updateView. You might have to do some research on the internet for things in this homework. That is good! For example, I have two CSS classes in my solution called .light-on and .light-off that set the button colors. My updateView changes the classes of the buttons as appropriate and the CSS rules apply the color based on the current class. We have NOT done that in the lectures. If you do it that way, you should research the internet to figure out how to add or remove a class from an HTML element. Yes, you have to learn things on your own and that's good! ;)

Checklist before you submit

Copied from the TA grading rubric:

Functionality:

- deployed to firebase at username-lightsout.web.app
- lights start out in a random configuration
- Pressing "New Game" starts a new random game
- Game text/state is accurate: Before first move: "Make the buttons match" Moves 1..n: `You have taken ${n} moves so far.` After winning: `You won in ${n} moves.`
- Clicking on a light causes that button and its neighboring lights to toggle state
- The game stops allowing moves once you have won

Style/Layout:

- font sizes change with screen size
- layout was created using flexbox (not grid, bootstrap, etc), is reasonably close to the pictures
- button color changes in tandem with the number (1=yellow/orange, 0=graphite/grey/black). The specifics of the exact color do not matter here.

Code:

- js was implemented with a separate model and controller with appropriate methods for each class.

If you know exactly what the TAs are looking for when they grade your work, it should be easy to get a 100% on this assignment.