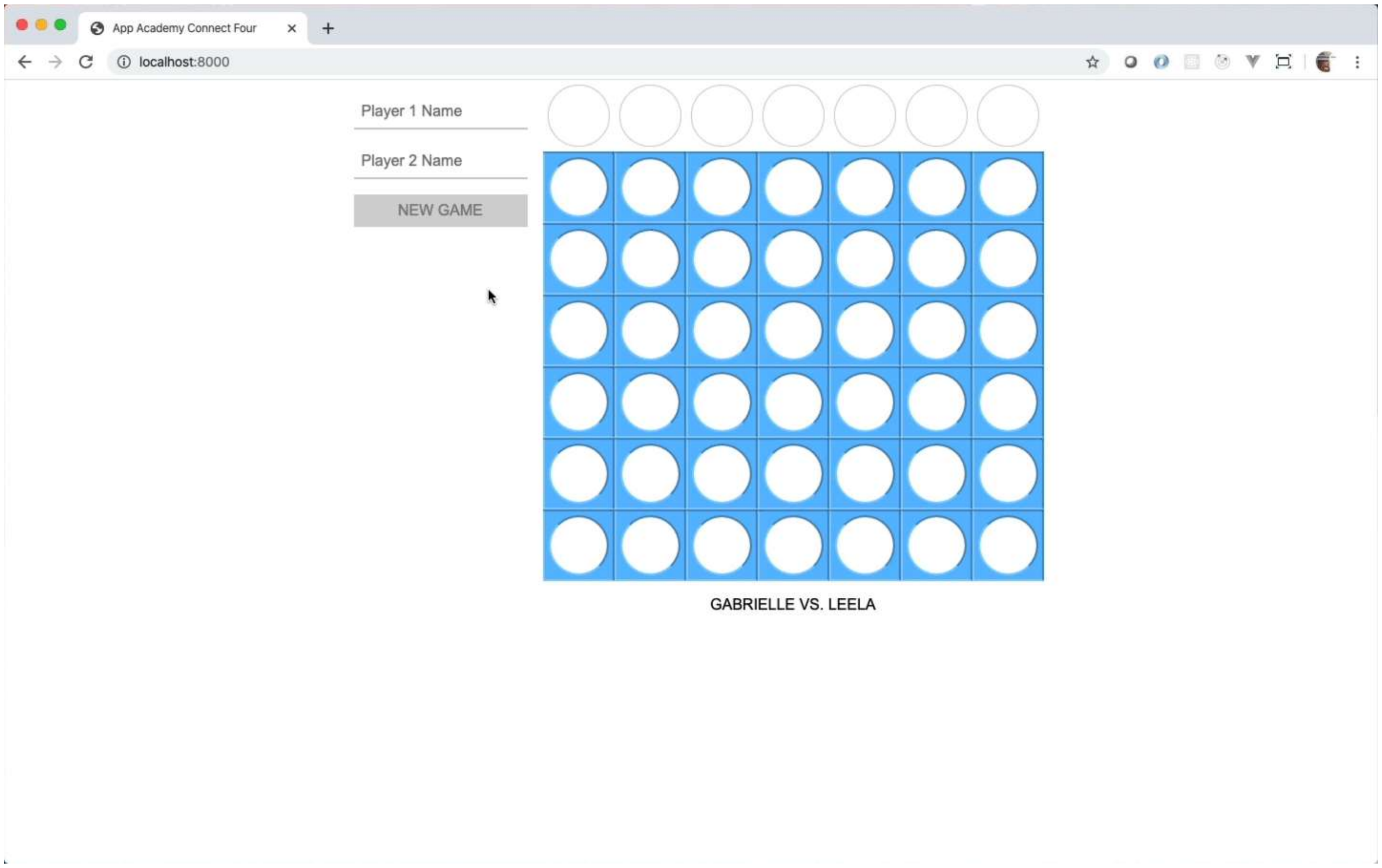
Placing A Played Token

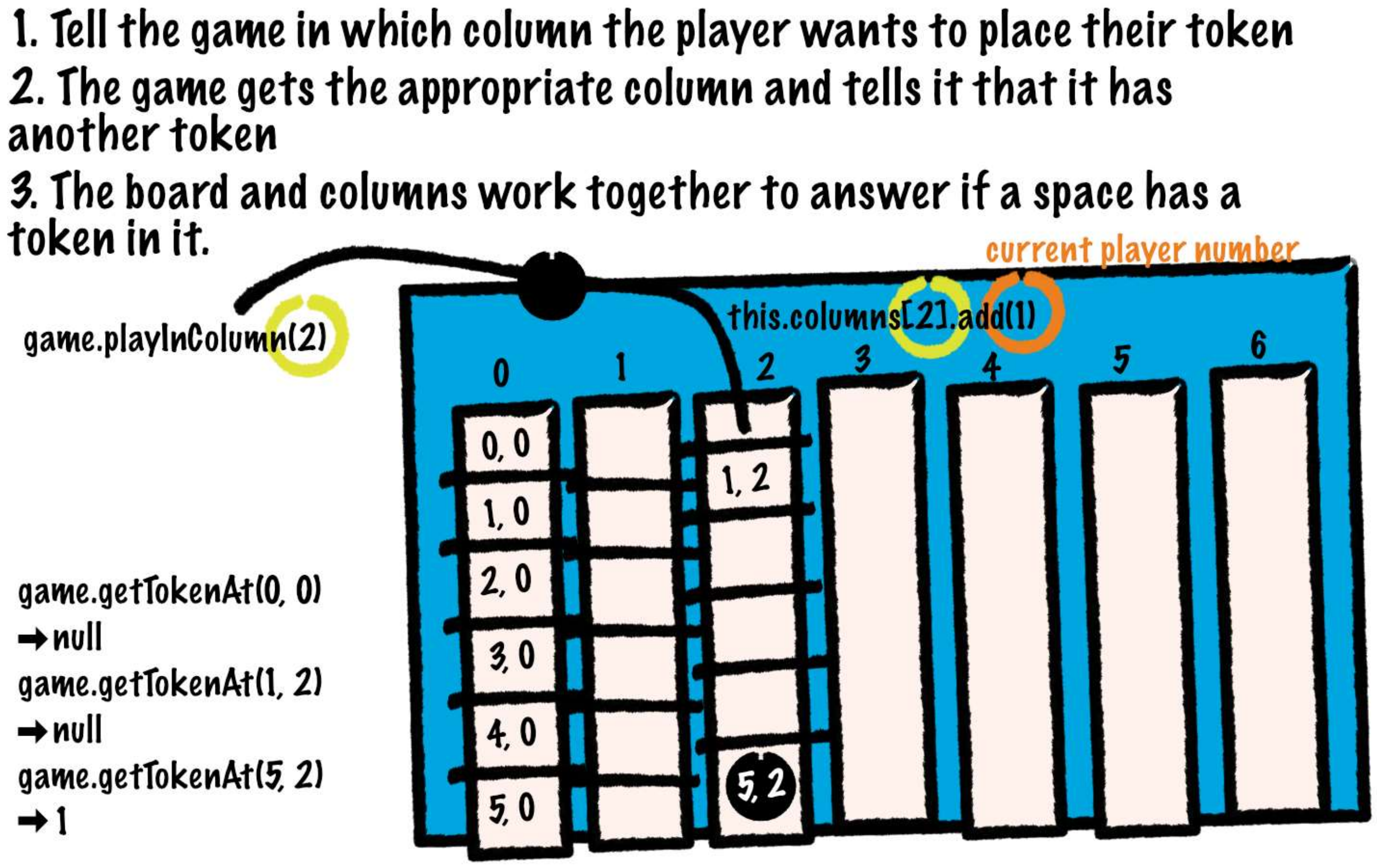


The player has clicked a click target. You have captured it and responded by changing the color of the hover on the click targets. Now, it's time to put that token on the board.

Here's a think. The board fills from the bottom-up. That's different than games like tic-tac-toe where you can just handle where the player clicks and put something in an array. In this case, your code needs to determine *where*the token is in each column because if fills from the "bottom up". Your code will have to answer, "where's the next available square for this next token to fall into?"

That seems like a lot of behavior to add to the `Game`class. It seems that there's some complexity, here, that goes beyond the single responsibility of the`Game`class. This is *token management*which could be delegated to some other concept....

Since each column behaves the same way, it would make sense to create a new class, a `Column`class that handles the unique behavior of a column in the board. The game will have an array that will contain seven instances of the`Column`class. Those objects will work together to make the board work like a Connect Four board!



1. Tell the game in which column the player wants to place their token
2. The game gets the appropriate column and tells it that it has another token
3. The board and columns work together to answer if a space has a token in it.

The design shows you that you will adapt the already existing `playInColumn()`class to take a number. The `Game`object will use that number to get the appropriate `Column`object. Then, it will call the `add()`method with the value of the current player, `1` or `2`.

Later, when your code needs to update the board, it will pass in the row and column that it's interested in querying to the `getTokenAt`method. That method will return `null`if the square is empty, `1` if player one's token is there, or `2` if player two's token is there.

Creating an array of custom objects

Say you wanted to create an array of strings of some street names. You could do that similarly to the following code snippet.

```
const streetNames = ['9th Ave', 'M. G. Road', 'Calle de las Huertas'];
```

That's an array of strings. You can create strings by just typing them as their literal representations. Now, answer this question before going further: ***How do you create objects from your classes?***

If you thought to yourself, "Self, that's with the 'new' keyword.", then you got it! If you have a class named `Person`and you want to create an instance of it, then you would type `new Person()`in your code and pass in whatever arguments the constructor needed.

Now, here's the big question: ***How would you create a "small" array of objects created from your classes?***

You just combine the array literal with the `new`that you use to create objects. If you want an array of three `Person`objects, you could just write:

```
const people = [new Person('Dima'), new Person('Siân'), new Person('Bob')];
```

If you needed to create an array of 100 `Person`objects, that's where a loop could come in handy.

```
const people = [];
for (let i = 0; i < 100; i += 1) {
  people.push(new Person());
}
```

Create the "Column" class

In the same directory as your **connect-four.js**and **game.js**files, create a new file named **column.js**. (Note that you don't *have* to do this, this one class per file thing. It is generally considered best practice to do it so that you know where to go look for a specific class rather than having to search through a long JavaScript file to see if a class definition is in there.)

The following instructions will cause you to think hard about how you can and want to do this. If you struggle with this, see the hints in the next task for two different ways to implement this, each with their own trade offs.

In the **column.js**file:

- Create and export a class named `Column`.
- Create a constructor that will create a way to manage the tokens stored in the column.
- Create a method named `add`that takes a player number and stores it in the "bottom-most" entry in the column.
- Create a method named `getTokenAt`which takes a row index number between `0`and `5`and returns `null`if there's no token there, `1`if player one's token is there, or `2`if player two's token is there.

Create "Column" objects in the "Game" class

In **game.js**:

- Add to the `Game`constructor a new instance variable named `columns`and initialize it to an array of seven `Column`objects.
- Add a method named `playInColumn`that takes the index of the column in which to play, uses that index to select the correct column from the array of`columns`, and calls the `add`method on that column object passing in the number of the current player. Make sure that you leave the toggling of the

current player from one to two and back, again, in the method at the end of it.
- Add a method named `getTokenAt`that takes the row index and the column index. Use the column index to get the correct column from the `columns`array. Then, call the `getTokenAt`method *on the column object*passing in just the row number. Return the return value of that function.

Add a click handler for the click targets

Now that you have the `Game`and `Column`classes ready to go, you can hook them up to the events that you're already capturing. In **connect-four.js**, in the event handler for click targets that you already have, before the call to the`playInColumn`method, you need to get parse the number of the click target that the player clicked on. Make sure your event handler is getting the event object in its parameter list. Then, access the "id" property of the "target" property of the click event. If it's a click that you want to handle, make sure that id value starts with the string "column-". If it does, then use `Number.parseInt`to convert the last character of the id into a number. Pass that number into the`playInColumn`method.

Update the tokens in the board

In the `updateUI`method, it's now time to show the tokens in the board. Create a `for`loop that will loop through the values from zero to five, inclusive; that will be the row index. Then, inside that **for-block**, create another`for`loop that loops from the values zero to six, inclusive; that will be the column index. Now, you have a row index and a column index to use to update the board. Inside the inner loop:

- Select the element `#square-«row»-«column»`using the row and column indexes that you have.
- Use the `getTokenAt`method on the `Game`object stored in the global `game`variable. The value that gets returned from `getTokenAt`will determine what you should do:
    - First, clear out the inner HTML of the square you selected in the previous step by setting it to an empty string
    - If the value returned by `getTokenAt`is `1`, then create a "div" element, make sure it has both the "token" and "black" classes, and add it as the child to the square.
    - If the value returned by `getTokenAt`is `2`, then create a "div" element, make sure it has both the "token" and "red" classes, and add it as the child to the square.