Homework #6

One Hand of Blackjack (Client Side JavaScript)

Overview

Repository Creation

👀 Click on this link: https://classroom.github.com/a/EjzlqWxN (https://classroom.github.com/a/EjzlqWxN) to accept this assignment in GitHub classroom.

- This will create your homework repository
- Clone it once you've created it.

Goals / Topics Covered

You'll be using the following concepts:

- manipulating the DOM by creating and adding elements
- setting DOM element attributes
- handling events with `addEventListener`
- stopping form submission with `preventDefault`
- using CSS to arrange elements with `position` and `display`

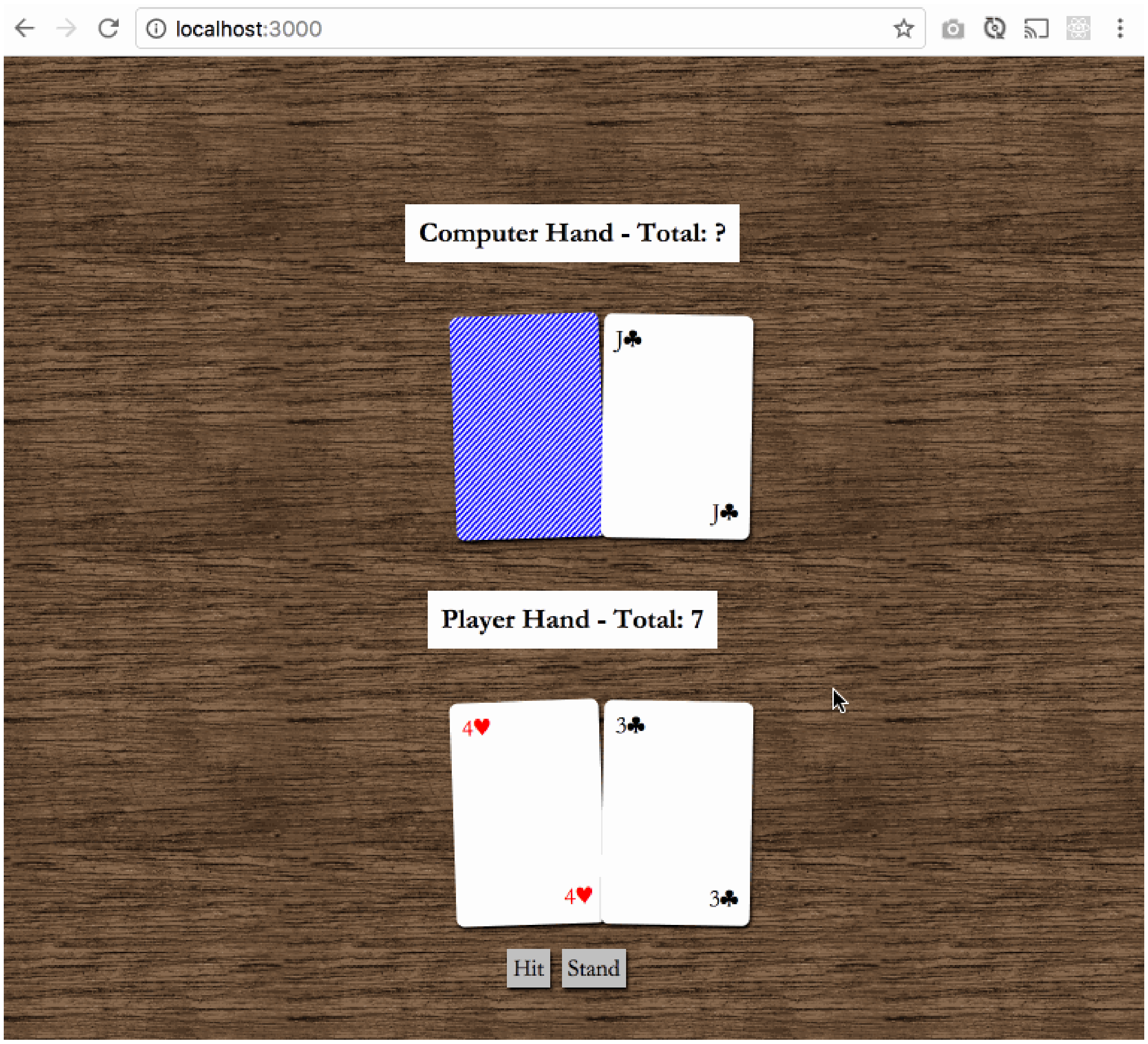
Description

Create a two-player (user vs computer) client-side card game. The card game will be **a single hand** of blackjack. If you're unfamiliar with the rules:

- each player will try to construct a hand of cards that's equal to 21 or as close to 21 as possible, without going over
  - the sum of the numeric values of the cards determine the value of a hand
  - face cards are worth 10
  - aces are worth 1 or 11
  - the player with the hand closest to (or equal to) 21 wins
  - ties are possible
- each player is dealt 2 cards from a 52 card deck, with each card representing some numeric value
- in our version, the initial cards are dealt to the computer and user in an alternating fashion, with the computer being dealt the 1st card:
  - the computer is dealt one card, and the user is dealt another card
  - this repeats one more time so that both the user and computer have 2 cards each
- once the initial two cards are dealt, the user can choose to be dealt more cards ("hit") or stop being dealt cards ("stand")
  - if the user's hand ends up exceeding 21, then the user automatically loses
  - if the user chooses to "stand" (to stop being dealt cards), then the computer can choose to continually "hit" or "stand"
- once both players have either chosen to stand or have a hand with a total that's more than 21 ("bust"), the hands are compared
  - the player with the hand that's closest to 21 without going over wins

- again ties are possible (either same total, or both player "bust")

Here's an example of what your game may look like:



Submission Process

You will be given access to a private repository on GitHub. The final version of your assignment should be in GitHub

- **Push** your changes to the homework repository on GitHub.

Make at Least 4 Commits

- Commit multiple times throughout your development process.
- Make at least 4 separate commits

Blackjack Game Requirements

There are three main features required for the game:

1. An initial screen where the user can enter a series of comma separated face values (2 - 10, J, Q, K, A) that will set the cards of the top of the deck to those values
   - Consequently, if a user puts in 2,3,4,5 …. 2 will be on the top of the deck, 3 next, etc.

    - So when hands are dealt, the computer will be dealt a 2 and a 4, and the player will be dealt a 3 and a 5
2. A client-side JavaScript implementation of one hand of blackjack
3. A small amount of styling:
    - the cards in a hand should appear adjacent to each other
    - cards should have a height and a width
    - *some effort* should be made to make the game appear visually polished

Setup and Initial Screen

Create an express application and setup some directories

1. You don't *really* need an Express application to do this homework, but start with one anyway, in case you decide to do the extra credit…
2. Start a new express project that uses express-static, using the following directory structure:
    - `package.json`
    - `.eslintrc.js`
    - `README.md`
    - `src`
        - `app.mjs`
        - `public`
            - `index.html`
            - `stylesheets`
                - `style.css`
            - `javascripts`
                - `main.js`
3. (there's no need to create any route handlers for this homework, you can do the whole thing with static files)
4. In your public folder, create an `index.html` file.
5. In your public folder, create a `stylesheets` folder and a file, `style.css` , within it
6. In your public folder, create a `javascripts` folder and a file, `main.js` , within it

Create a form

1. Add the following code to your `index.html` (it will be the only markup in your project, everything else will be generated with JavaScript):

```
<!doctype html>
<html>
<head>
<title>Cards!</title>
<link rel="stylesheet" href="stylesheets/style.css" type="text/css" media="screen" title="no title" char
set="utf-8">
<meta charset="utf-8" />
</head>
<body>
<script type="module" src="javascripts/main.js"></script>
<div class="start">
  <form method="POST" action="">
 <label for="startValues">Start Values:</label> <input type="text" name="startValues" id="startValues">
 <input class="playBtn" type="submit" value="Play">
  </form>
</div>
<div class="game">
</div>
</body>
</html>
```

2. Test that your form shows up when you go to `http://localhost:3000`
3. **You are not allowed to use any additional markup, and you cannot modify the markup above**; you must generate any additional elements you'll need with JavaScript

Initialize your JavaScript

Note that `main.js` is included at the top of the body. This means that the code within it may run before the rest of the page (DOM) is actually available to manipulate. This problem will result in:

1. a failure to find an element that *should* exist (when using `document.getElementById`, `querySelector`, etc.)
2. which most likely will manifest itself as an undefined object

So, **to delay the running of your JavaScript until the page has completely loaded** (and without changing the markup, use `document.addEventListener` to call the main line of execution for your program only when the DOM has completely loaded. Follow these steps to do this:

1. in `main.js`, create a function called `main` - this will be where your game code resides (of course, you can create other functions, but you *should* have at least a main function)
2. after that, add the following line to make sure that the `main` function is only run when the DOM is completely loaded: `document.addEventListener('DOMContentLoaded', main);` (note that modifying the script tag by adding `defer` would work as well if changing the script tags were allowed)

Handle the Form Submission and Modify the User Interface

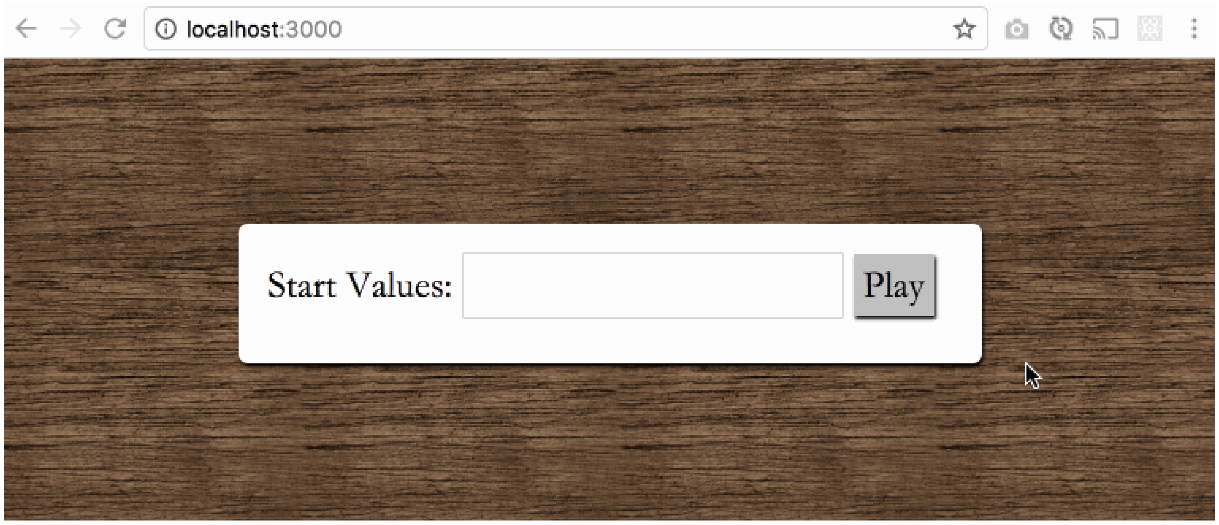
Submitting the form hides the original form and distributes (deals) cards to the computer and user. If the user entered a list of card faces in the form before submitting, then those cards will be dealt first from the top of the deck.

Start off by handling the submit button on the form and making the form disappear. Use a combination of `document.querySelector` and `addEventListener` to allow the submit button on the form to be pressed:

- see the slides on `document.querySelector` (../slides/19/js-css.html#/6) or the mdn documentation (https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector)
- check out the slides on events (../slides/19/events.html#/)
- …along with mdn's documentation on addEventListener (https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener), click (https://developer.mozilla.org/en-

US/docs/Web/Events/click), and DOMContentLoaded (https://developer.mozilla.org/en-US/docs/Web/Events/DOMContentLoaded)

- because the button in the form is a submit button, it'll *actually* make a GET or POST request when pressed; we don't want this to happen, so we'll use preventDefault (../slides/19/events.html#/15) to stop the default action from occurring on a click event
- and lastly, the slides (../slides/19/css.html#/56) on classList (../slides/19/css.html#/7) will show you how to add, remove and toggle css classes on elements
- Remember, you'll need to put all of your DOM dependant JavaScript in a `DOMContentLoaded` listener (which you already did above)
- and, of course, you'll need to add a `click` event listener for your submit button
- within the callback to your `click` event listener:
    1. make sure to call `preventDefault` (see the slides on events (slides/19/events.html#/15)) to stop the form from submitting
    2. create and apply the appropriate CSS classes to get rid of the *form* (do this with styles, there's no need to remove the element) to make room for displaying cards (in addition to the slides above, here's the mdn docs on classList (https://developer.mozilla.org/en-US/docs/Web/API/Element/classList))
- Note that `this` within the callback passed to `addEventListener` will refer to the element that was clicked on, as long as the function you use is not an arrow function
    - For example….
    - Using this markup: `<div id="clicker">Click Me</div>`
    - Using this JavaScript: `document.querySelector('#clicker').addEventListener('click', myCallback)`
    - `this` within `function myCallback` will refer to the original div element clicked on!



Generate a Deck of Cards and Deal Cards

Once you've gotten rid of the form, generate a deck of cards and distribute a hand to the user and the computer (optionally setting the top cards to the values set in the form above).

- Note that there was a text field in the original form - this can be used to set the cards at the top of the deck
    - If the player enters a value in this field, then the cards on top of the deck are set to the sequence inputted
    - The input should be a comma separated list of characters and/or numbers (for example `K,2,3,7`), the suit of the cards can be any suit (in the example below, all of the specified cards were given a

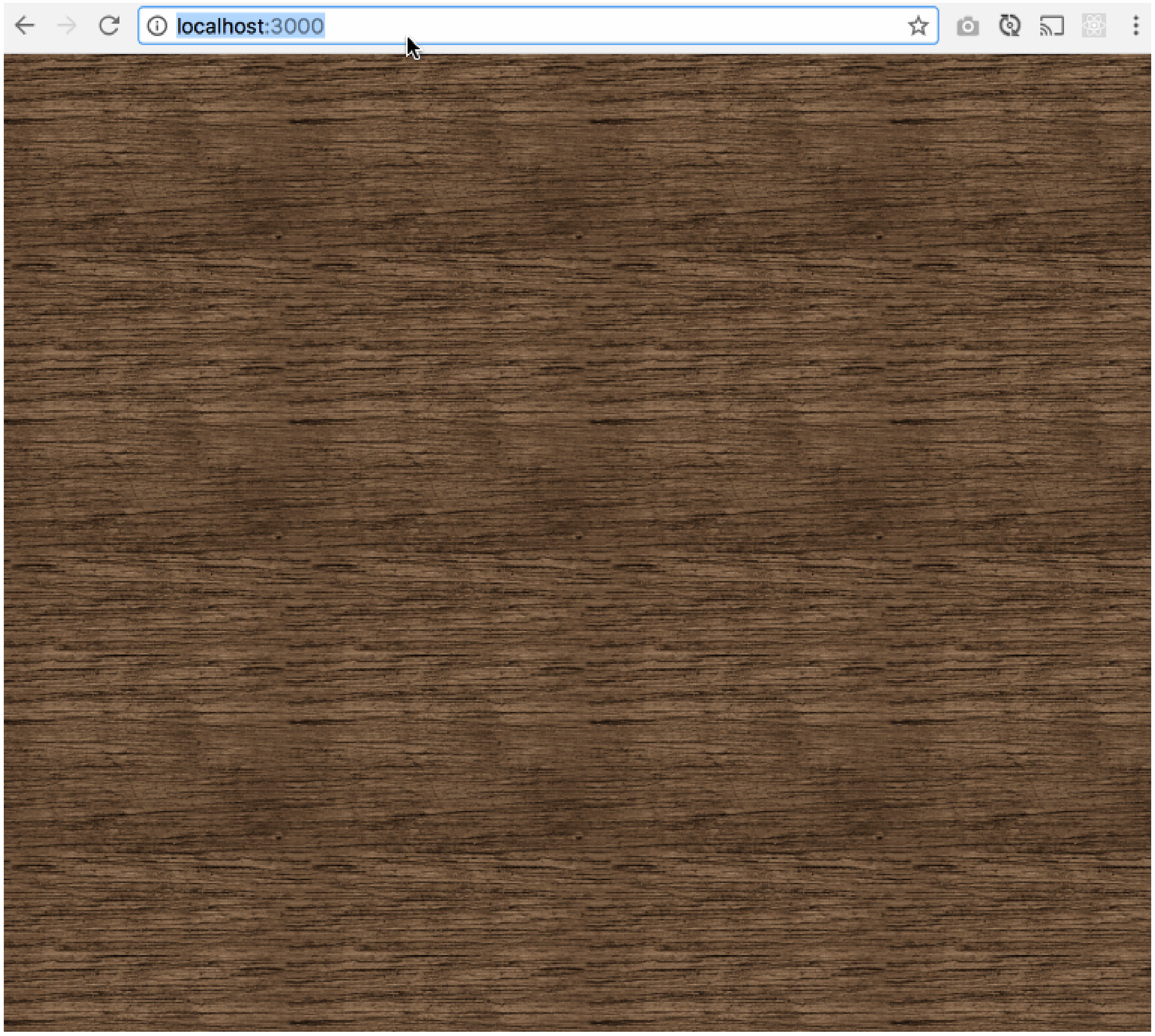
    default suit of diamonds)
  - No validation is required (assume that the user puts in valid input or no input)
  - You can retrieve the user input from the text field by using the `value` property on the form element that contains the user input - see the mdn docs on value under HTML Input Element (https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement)
- **The following should done programmatically via client side JavaScript** (the graphical part will be specified in the next section)
- Generate a deck of 52 shuffled cards (perhaps an Array of objects?)
- If there were card faces that were entered in the form above…
  - add those cards to the top of the deck (again, you can just pick a suit; it's the faces that matter)
  - these cards will be dealt first, alternating between the computer and the user
  - for example, if the cards 2, 5, 2, 5, J, Q, K, then:
    - the computer would get a 2 and 2
    - the user would get a 5 and 5
    - the next cards dealth (from pressing the "Hit" button) would be J, Q, and K
- Deal the cards - alternate between the computer and player
  - the computer gets the first and third card
  - the player gets the second card and fourth card

Display the Cards and User Interface

Based on the cards dealt, add elements to the user interface to represent both the user's and computer's hands (set of cards).

- Create DOM elements to represent cards
  - Check out the slides and mdn documentation on modifying and creating elements
    - slides on adding and removing elements (../slides/18/modifying-creating.html#/1)
    - slides on changing an elements text (../slides/18/modifying-creating.html#/5)
    - slides on creating elements (../slides/18/modifying-creating.html#/7)
    - mdn docs on `createElement` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement)
    - mdn docs on `createTextNode` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createTextNode)
    - mdn docs on `appendChild` (https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild)
  - You should consider writing some helper functions for creating and adding elements
  - For example, one of our books introduces a function to create an element and add children immediately (http://eloquentjavascript.net/13_dom.html#c_Mnkp5ioh9C)
  - Programmatically create elements to represent cards
  - All of the computer's cards should appear next to each other horizontally - one of the cards should not be revealed
  - All of the user's cards should appear next to each other horizontally
  - All of the cards should have a width and height
  - Hint: to lay out elements adjacent to each other and still maintain a width and height, you can use …
    - `display: inline-block`
    - `display: flex`
    - a `table`
    - or `float` your elements

- Calculate the hand total for both the computer and the user
    - Remember that aces can be 1 or 11
    - Make sure that your algorithm optimizes the values of aces so that the hand total is as close to 21 as possible without going over
- Create two elements to display the computer and user total
    - the computer total should be displayed as ?
    - the user total should reflect the total in the user's hand
- Create two buttons, `Hit` and `Stand`
- See an example interaction below:
- THE STYLING IN THE ANIMATED GIFS IS JUST FOR EMBELLISHMENT; YOUR STYLING DOES NOT HAVE TO MATCH
    - (but you should put in some effort to make the cards *card-like*)



Hit and Stand

Once user's hand and the computer's hand are displayed after the initial deal, the user can now decide whether they want more cards or stay with the cards that they have. The previous section added buttons (Hit and Stand), but now you'll have to add event listeners to those buttons.

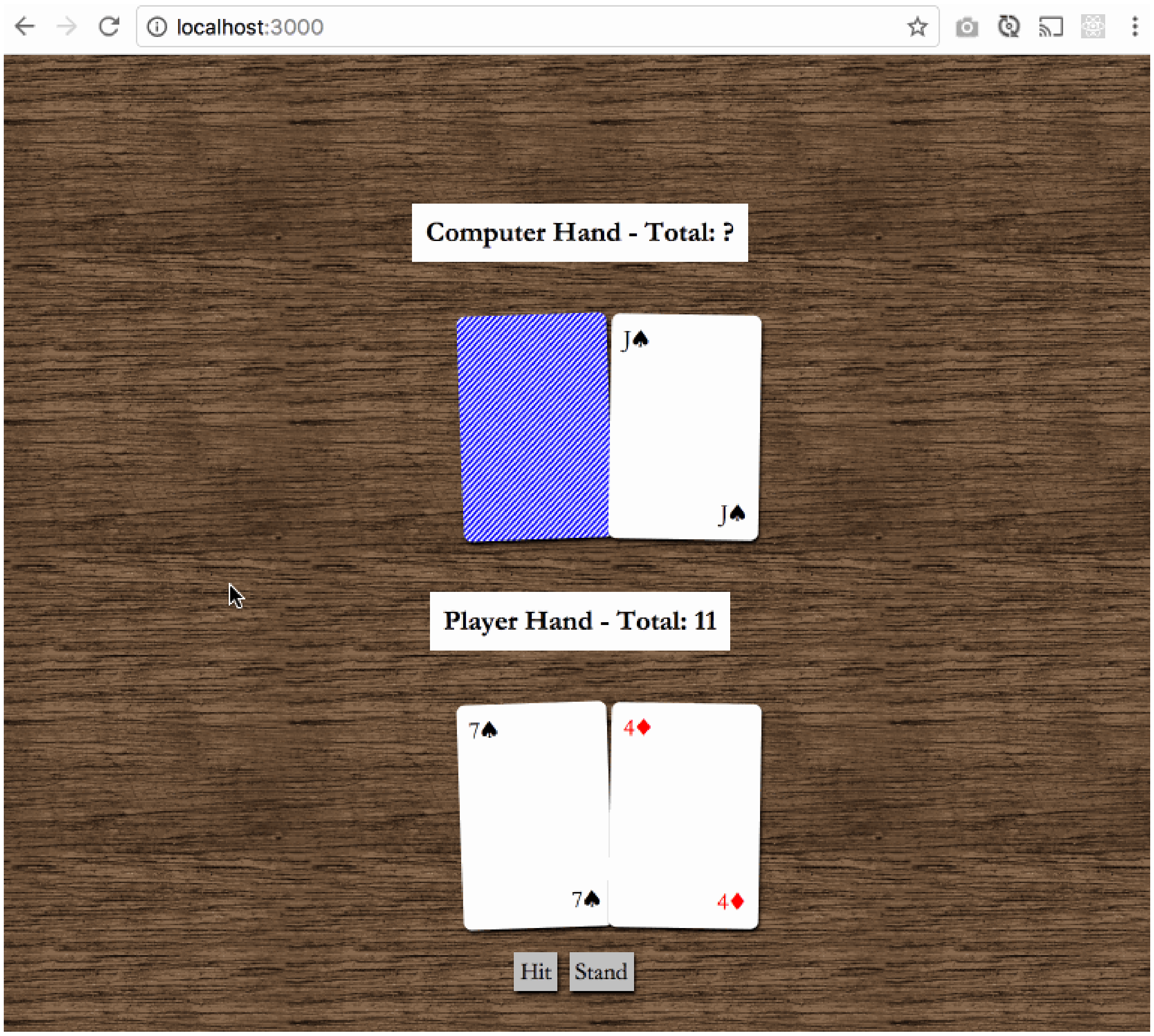
1. pressing `Hit` should deal the next card from the deck
    - the next card should be moved from the deck to the user's hand
    - an element should be created to add the card to the user's hand in the user interface

- if hitting makes a user's total go over 21:
  - then the user's turn ends immediately
  - … and they lose the hand
  - (the computer does not even need to decided to hit or stand)

2. pressing `Stand` should end the user's turn and allow the computer to Hit or Stand
   - the computer's strategy is your discretion
   - though the computer must hit in some situations or stand in others
   - an easy strategy is to always hit if a hand total is underneath a threshold, but stand if it's eqaul to or above that threshold

The following is an example of the user:

1. hitting twice
2. then going over 21 (busting)
3. (and consequently, immediately losing the hand)



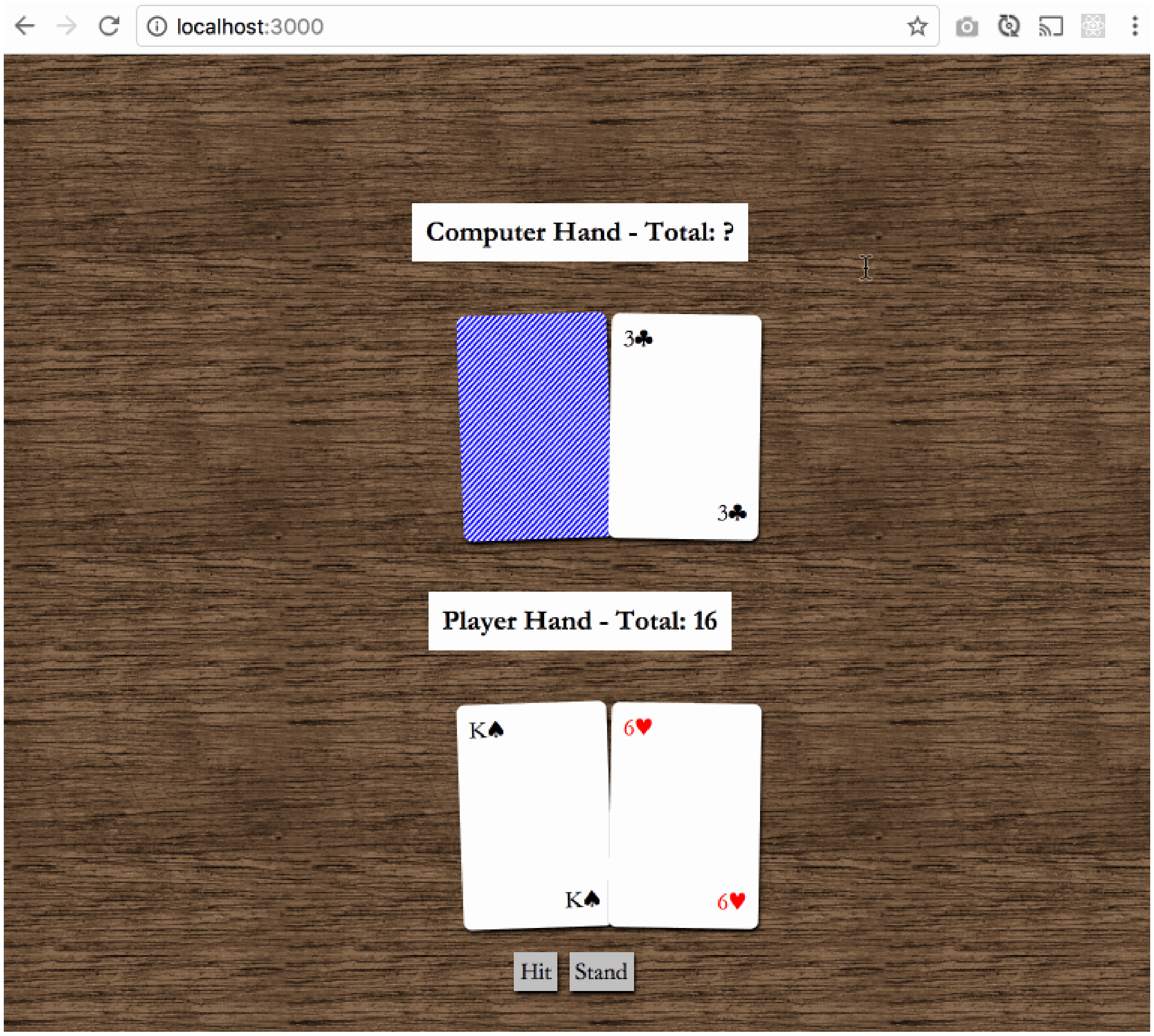
Determine the winner

Once the user and computer have both decided to "stand" or have a hand total over 21…

1. Compare the computer's score with the user's score
2. The player with the score closer to or equal to 21 wins
3. Add text to indicate of who won

4. If the player went over 21, they lose automatically
5. **A tie is possible**

Below is an example of the user winning a hand:



Optional Features (Extra Credit)

Implement any of the following features for extra credit.

If you are implementing extra credit:

1. add a README.md to the root of your project
2. add notes specifying which extra credit you did

(5 points) **Reset hand**

- when a hand ends, remove the `Hit` and `Stand` buttons and add a `Restart` button along the bottom row
- when the `Restart` button is pressed, it resets the game so that:
  - the computer and player hands are cleared
  - new cards are dealt
  - the `Hit` and `Stand` button are added back

(10 points) **Save results in database**

- when the game ends, allow the user to enter their initials in a form

- when the form is submitted:
    - save the results of this hand (the computer scores and the user score) in the database, along with the user's initials
    - display the results of the last 5 hands
- if this is combined with the reset hand, then place the restart button after the form has been submitted