**Program is like a card game often called “fight” or “war”. V2**

[1] In separate js file, define an object constructor that builds card objects. This is a constructor function as we need to make 52 of these, not an object definition.

Card objects have 2 properties,

- suit which can be one of 1,2,3,4 and

- rank, which can be 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, or 14

The constructor should take in 2 values to set these 2 properties for each card object you create.

[2] In separate js file, create an **object** called deck. This is an object definition, not a constructor, as we only need one of these.

Deck should define an array called cardArray and it should also have a load method you can call that fills that array with 52 “card objects”, one of each of the possible unique combinations of suit and rank. Note that when the code in the load method references the cardArray (in the same object), you must say this.cardArray to refer to the cardArray in the object you are currently executing in. There are many different ways of building some number of loops, or nested loops to achieve this, any way is fine as long as it uses loops and it not just 52 lines to define each array member.

[3] In separate js file, your primary js file, in conjunction with your index.html file and your css file

- define 2 arrays (each which will be loaded to a size of 26), one is playerArray and the other is computerArray

- Display a button called DEAL

**When the button called DEAL is clicked, in its click event method you should**

- 1st call your deck.load() method which should load the deck’s cardArray with the 52 unique cards.

- copy the 52 cards from your deck object array into the 2 arrays ( playerArray and computerArray ), doing so randomly . I used this line in my code

random = Math.floor(Math.random() \* (52));

make sure not to use the same card twice. See my idea at the end of the document for how I did this. You may use other schemes if they work.

- At this point make the DEAL button hidden, and a new button, NEXT should appear

**When the button called NEXT is clicked, in its click event method you should**

- Displays 2 cards, the [0] card from each of the two arrays ( And then each subsequent click of NEXT walks down the 2 arrays, [1], [2], etc up to [25] turns. See below for how “cards” should be displayed

- Under or over each card display it should display “player” or “computer”

- It should display a message that the player either “won” or “lost” the round

- It should display scores for each, starting at 0, and incremented based on who won

- decide who won. You win by having a higher value card rank, and if the ranks are equal, than the winner is based on suit value

- somewhere display which turn it is, counting from 1 to 26. I suggest using that value in this event method as your array pointer for the 2 arrays as well.

- when the 26th turn is complete, it should write out “WON”, “LOST” or “TIE*”, and then OPTIONALLY, the NEXT button should disappear, and the DEAL button should reappear so they can play again.* Being able to restart is not required.

[4] As you begin writing the game, I suggest for the card display, you just write out simple numbers, such as R: 5 S: 2 in a large font. (See first YouTube video below)

After you have everything else working, then finish the project by have the game instead display the rank, without the R:, but changing the 11 to a “JACK”, the 12 to a “QUEEN”, the 13 to “KING” and the 14 to “ACE” I did this be creating a function to which I passed in a rank number, then using a switch statement, I returned the correct string.

And also instead of writing S:, write out “SPADE” for a 1, “CLUB” for a 2, “DIAMOND” for a 3, and “HEART” for 4. I did this be creating another function to which I passed in a suit number, then using a switch statement, I returned the correct string.

And the rank and suit should be black if the rank is 1 or 2, and red if 3 or 4. I did this with some if statements, changing the class tag of the elements between a class with black or one with red colot.

(See the second YouTube video)

============================

**One idea for not giving out the same card twice.**

Add another property to the card object, make it a bool that is called .inuse

In your card object constructor function, set this value to false (meaning, you have not used this card yet)

Then in your loop that picks random cards from the array of 52 to place into the 2 arrays of 26, check each card that the random number points to, if its inuse property is true, then don’t use it, and pick another random card. When you find a random card thats property is still false, then use it AND set the property to true so it will not be used again.

If you decide to take the challenge and allow the game to be restarted, then you should do a

deck.cardArray.length = 0; // empty array to start over

as the first instruction of your DEAL button event, just before you call deck.load(). This will make sure all the inuse property values are set back to false.

Here is a YouTube of my program running in an early form. I have not yet made the card displays look better yet. I have not enabled a re-play yet, and I have my turn counter set to 5 instead of 26 to make debugging easier.

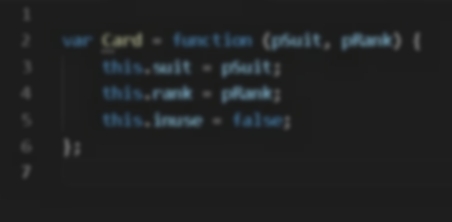
<https://youtu.be/w3jLy6OKyGE>

Here is a YouTube of my program running in an final form, going thru a full game of 26 cards, and allowing the user to re-click DEAL at the end of the game to play again (which is not required).

<https://youtu.be/8n164Ktg1FQ>

to give you an idea about how much code is involved, here are pics of my code.

Here is my file with the Card constructor



Here is my file with the Deck object definition including the method to load the 52 cards



Here is my file with the HTML



Here is my main js file in 2 pictures, the top part is the “deal” button event code and the 2nd one is the bottom part, the Next button event code.



