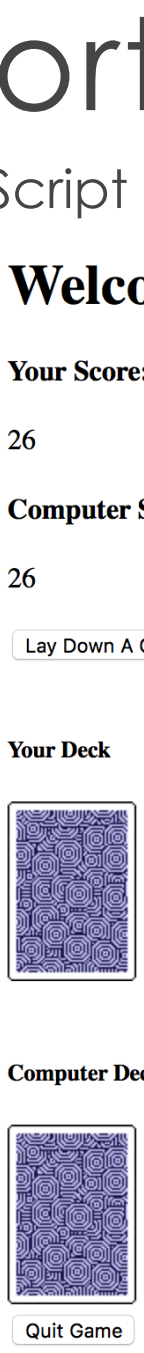
Portfolio I

JavaScript War Card Game



Jacob Neyens and Hope Scheffert (Group 12)

Table of Contents

Overview

The goal of this portfolio was to experiment with JavaScript, HTML, and the Model-View-Controller software architectural procedure to implement the popular card game known as "War." After learning about JavaScript in our lectures and labs and all of the awesome things you can do to a webpage, we decided it would be a great platform for our project. Creating the card game War sounded like a interesting and fun way to display what we have learned.

Our inspiration for the card game actually derived from the matching game presented to us in class. We noticed how the MVC design pattern worked and that implementing a game wasn't so out of reach or as intimidating as it looks and sounds.

The most challenging part about our game was figuring out the use cases according to the game rules. First we had to basically Google the game rules for a refresher, and then decide how we wanted to set it up. Once we got a better understanding of how the game is played, we were able to come up with basic use cases, such as two cards having the same rank, and therefore the players "go to war". Also, what if they went to war, and then their cards matched again?

We also had trouble dealing with the DOM. It was difficult to lay down the cards correctly so that the game is as real as possible. We would have liked to implement a more "fancy" view, if we were given more time. It was also difficult to stay in a strict MVC design pattern. We found ourselves making elements in the HTML file instead of actually breaking them down the way MVC does. A few times, we tried using the MVC style, and it wasn't working the way we thought, and we had to make changes accordingly in order to make our code actually function

correctly. Again, given more time, we could have gotten some clarification and fixed those issues.

Finally, it would have made a huge difference in our game if we had more knowledge of CSS, as this would have led to a more appealing web page and UI. If we would have had more time, we would have been able to learn and understand CSS better and implement it into our game. Yet, we are satisfied with the project we have created because we met our goal which was to become knowledgeable of JavaScript and the Model-View-Controller software design pattern and in result have a fun and accurate game to play.

New and Complex Issues

I. MVC Design Pattern

The MVC Design Pattern was implemented into our project. We had not used this pattern yet in our javascript assignment, so it was difficult to understand how to use it at first. After working with the MVC design pattern for a few hours it became a much clearer concept and we were able to implement it better. The MVC pattern ended up cleaning up our code quite a bit and making it more logically engineered. The Model contained the initialization of the Card objects and the Deck objects. The View contained all of the code that was needed to generate text, buttons, and the photos for the face and back of the cards. The Controller contained all of the code that performed actions on the Deck and Card objects.

II. Going to War!

This functionality of our game is without a doubt what gave us our biggest issue. The player and computer would go to war when the card thrown by the player would be equal in value with the card thrown by the computer. When this would happen the player and computer would each lay a card face down and after that was done they would each flip another card over. Whoever's card was higher would then win all six cards (three cards each for player and computer) on the "table," and if the new cards flipped were equal to each other the action was started all over again.

We had several issues with this operation throughout our project. At first, we had several processes performed at once when the initial call to war was made. It would go through and carry out these actions without showing them on the webpage, which also made it hard to debug. It would show that a call to war was made and would then say that the player or computer won based on the card that was used to be the tiebreaker, but it would not show what the tiebreaking cards were. Once we fixed this issue, another one popped up where it would skip over the card that would be put face down. We went into debugging mode and identified that it was actually laying it face down but immediately carried out the rest of the operations and played the next card face up. This is not how we intended the game to work and tried several different methods of implementing our intention before resolving the issue. For example, we tried to change the onclick method of the "Lay Down a Card" button to the warLayDownButtonHandler() if we were in warMode, but this (for whatever reason) wasn't working.

After struggling for a while with this, we decided to be creative and get the job done by any means. Our final resolution was to spawn a new

button called "War" between the player and computer hands that handled laying the middle card face down, then going back to using the normal "Lay Down a Card" button.

Bloom's Taxonomy

I. Analysis

The first thing we did was analyze the game rules and situations, and determine what functions, buttons, button handlers, etc. that we needed to implement in a Model, View, Controller design pattern similar to that of the matching game example from class. It is probably true that we got veered away from the MVC design pattern...but this was due to time constraints and needing a fully functional program.

II. Evaluation

After first making two separate functions for the main areas of play: "layDownACardHandler()", and "war()", we found that this led to issues with the DOM. Our problem was not in our logic, because we self tested by printing to the console at significant parts of the game and making sure the state of each player and their "hand" were correct.

We found that we were trying to do too many things in one function. This led to manipulating the DOM all at once, and we weren't able to show the case where there is a war, the two players lay their first card face down, followed by their second card face up. Our code was doing both things back to back, resulting in only the second action being shown on the page...so it appeared to be skipping a step.

In order to fix this, we created a variable "warMode" which was false unless two cards had matched, and therefore the players went to war. We then sent the necessary information along with this boolean "inWarMode" to the original layDownACardHandler(). Here, we were able

to check if we were currently at war with our opponent, and therefore our next cards laid must be *face down*. After that, we returned and waited for the user to click the Lay Down a Card button to battle the computer.

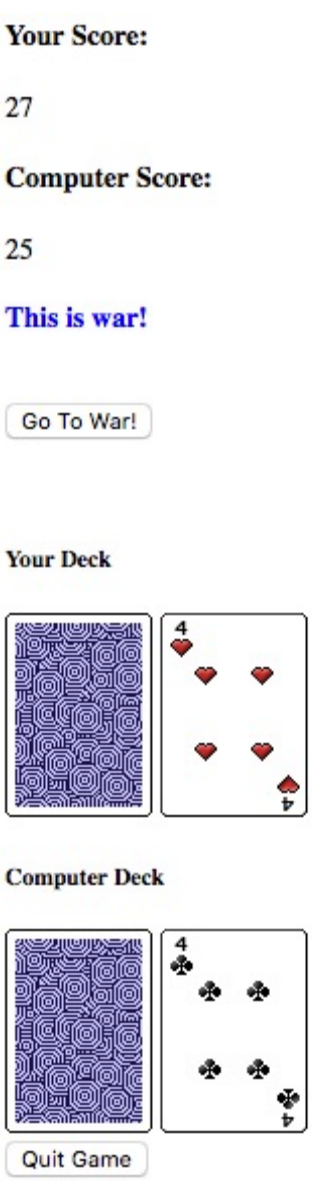


Figure 1

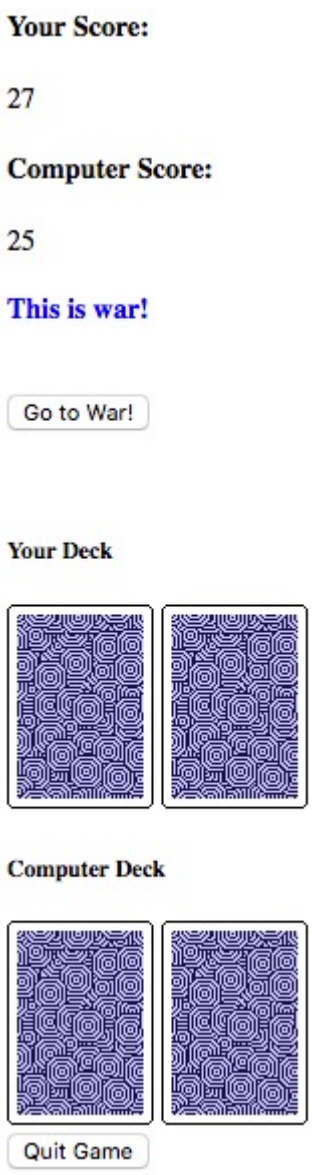


Figure 2

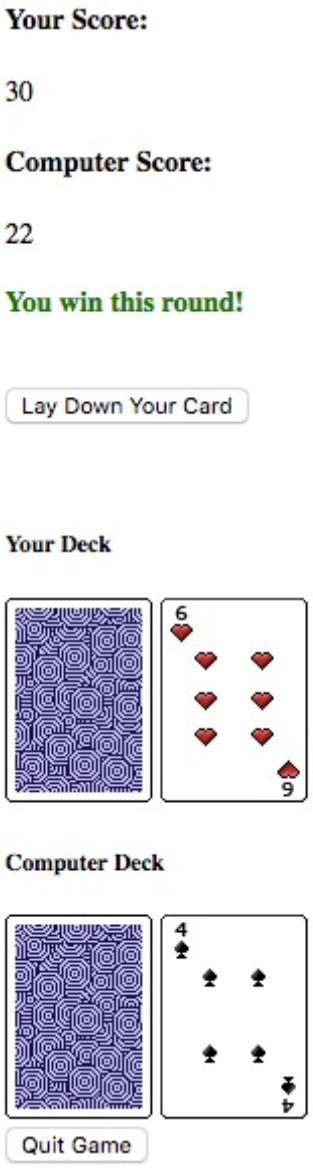


Figure 3

Figure 1 demonstrates what happens when two matching cards are flipped up, Figure 2 shows that a card is placed face down during war, and Figure 3 displays that the next card is flipped for both player and computer

III. Creation

Our game is a War Game that was constructed from scratch but possesses all of the original rules of the well-known card game of War. It consists of a user who is the "Player" who battles against the computer.

There is a button that lays down the card on the top of your deck and if that is equal to what the computer lays down, another button is generated to commence War on the computer. There is a third button that is always on the screen that allows you to quit the game.

Developed a handler for taking a typical turn in which the boolean "warMode" is always false and if the cards are equal in value it changes the boolean to true and calls the warLayDownButtonHandler function. The reason we assembled this warLayDownButtonHandler is to take care of the issue of where we had the game skipping over laying down a card face down before flipping the war card.

If we could go back, I think we would definitely make a bunch of helper methods to clean up this code. Because we started off making two methods that were so similar, then changed our minds and used only one method with a "mode" variable--we ended up working against ourselves in the long run. We will keep this in mind for our following portfolios or other projects in general. We should have taken more time with the use cases before starting the implementation.

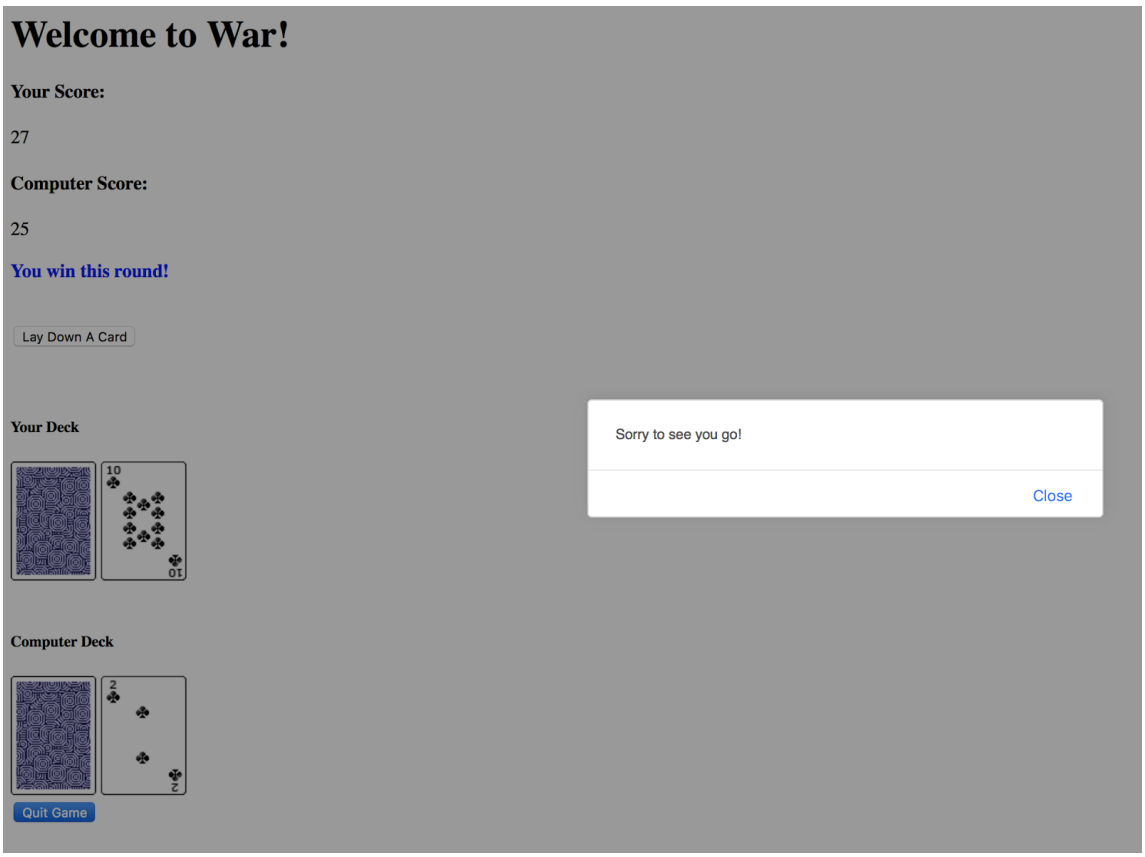


Figure 4: Demonstrates what happens when "Quit Game" is clicked