



Week 6: Final Project

URL to GitHub Repository:

<https://github.com/Rcruzn33/War-Card-Game-JS.git>

URL to Your Coding Assignment Video:

<https://youtu.be/5ivUp5XM9vk>

Instructions:

- In Visual Studio Code, write the code that accomplishes the objectives listed below and ensures that the code compiles and runs as directed.
- Create a new repository on GitHub for this week's assignments and push this document, with your project code, to the repository.
- Include the URLs for this week's repository and video where instructed.
- Submit this document as a .PDF file in the LMS.

Coding Steps:

- For the final project you will be creating an automated version of the classic card game *WAR*! There are many versions of the game *WAR*. In this version there are only 2 players.
 - You do not need to do anything special when there is a tie in a round.
- Think about how you would build this project and write your plan down. Consider classes such as: **Card**, **Deck**, **Player**, as well as what **properties** and **methods** they may include.
 - You do not need to accept any user input, when you run your code, the entire game should play out instantly without any user input inside of your browser's console.

The completed project should, when executed, do the following:

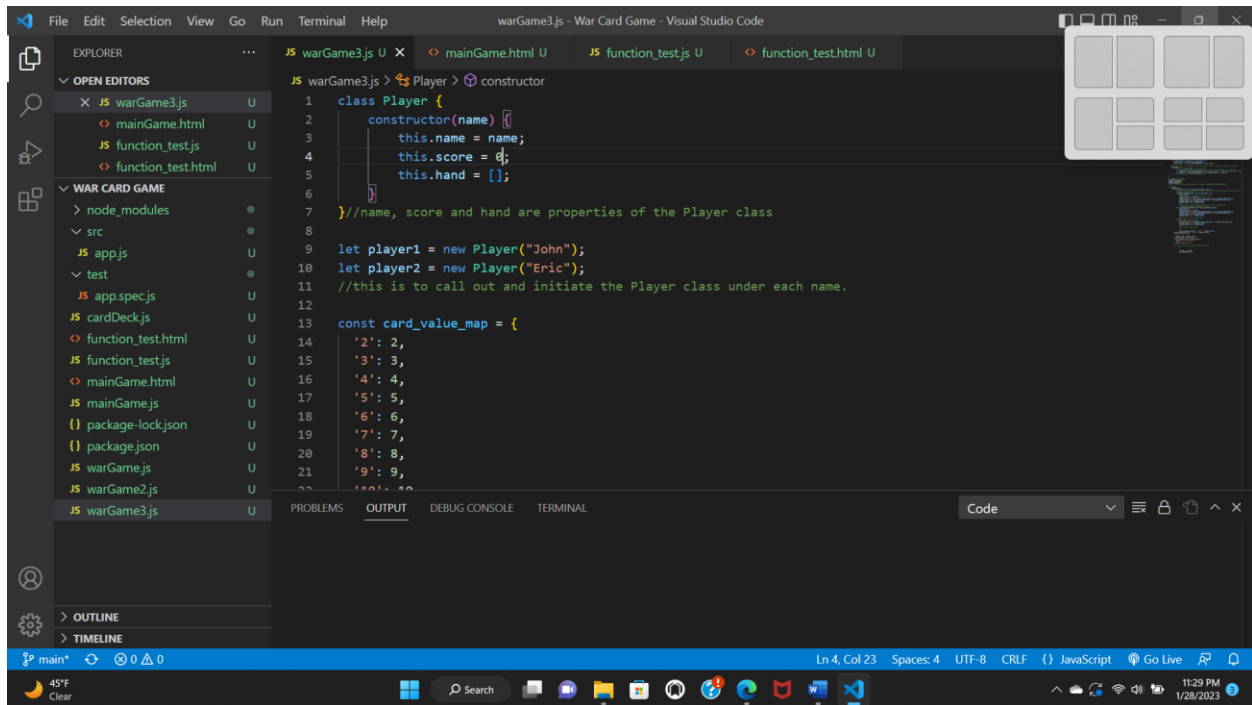
- Deal 26 Cards to each Player from a Deck of 52 cards.
- Iterate through the turns where each Player plays a Card.
- The Player who played the higher card is awarded a point
 - Ties result in zero points for both Players
- After all cards have been played, display the score and declare the winner.
- Write a Unit Test using Mocha and Chai for at least one of the functions you write.



Week 6: Final Project

Video Steps:

- Create a video, up to five minutes max, showing and explaining how your project works with an emphasis on the portions you contributed.
- This video should be done using screen share and voice over.
- This can easily be done using Zoom, although you don't have to use Zoom, it's just what we recommend.
 - You can create a new meeting, start screen sharing, and start recording.
 - This will create a video recording on your computer.
- This should then be uploaded to a publicly accessible site, such as YouTube.
 - Ensure the link you share is **PUBLIC** or **UNLISTED**!
 - If it is not accessible by your grader, your project will be graded based on what they can access.





PROMINEO TECH

Week 6: Final Project

The image displays two screenshots of a Visual Studio Code editor window titled "warGame3.js - War Card Game - Visual Studio Code". The editor shows the development of a War Card Game in JavaScript.

Top Screenshot: The Explorer pane on the left shows the project structure, including files like `warGame3.js`, `mainGame.html`, `function_test.js`, and `function_test.html`. The main editor displays the `warGame3.js` file, showing the `Card` and `Deck` classes. The `Card` class has a constructor that takes `suit` and `value` as arguments and assigns them to `this.suit`, `this.value`, and `this.rank` (using a `card_value_map` object). The `Deck` class has a constructor that initializes `this.deck` as an empty array.

```
class Card {
  constructor(suit, value) {
    this.suit = suit;
    this.value = value;
    this.rank = card_value_map[value];
  }
}
//value, suit, and rank are properties of the Card class
//this is where rank and value are combined via the card_value_map

class Deck {
  constructor() {
    this.deck = [];
  }
}
```

Bottom Screenshot: The Explorer pane shows the same project structure. The main editor displays the `warGame3.js` file, showing the `Deck` class with additional methods: `createDeck` and `dealDeck`. The `createDeck` method initializes the `suits` array and the `values` array (using `Object.keys(card_value_map)`). It then uses a `for` loop to push new `Card` objects into the `deck` array. The `dealDeck` method calculates the `deckMidpoint` and assigns the first half of the deck to `player1.hand` and the second half to `player2.hand`. A comment indicates that the `shuffleDeck` method is called next.

```
class Deck {
  constructor() {
    this.deck = [];
  }

  createDeck() {
    let suits = ['hearts', 'diamonds', 'clubs', 'spades'];
    let values = Object.keys(card_value_map);
    //this is to call out and initiate values, suits and ranks of the Card class
    //Object.keys is used to get the keys (card values) from the card_value_map Object
    for(let valueCount=0;valueCount<values.length;valueCount++) {
      for(let suitCount=0;suitCount<suits.length;suitCount++) {
        this.deck.push(new Card (suits[suitCount], values[valueCount]))
      }
    }
  }

  dealDeck() {
    let deckMidpoint = Math.ceil(this.deck.length / 2);
    player1.hand = this.deck.slice(0,deckMidpoint);
    player2.hand = this.deck.slice(deckMidpoint,this.deck.length);
  }
}
//function to split the card deck in two for each player, calls and establishes a midpoint in the deck
shuffleDeck();
```



Week 6: Final Project

The image displays two screenshots of a Visual Studio Code editor window, showing the development of a War Card Game. The editor is open to the file `warGame3.js`, which contains the game logic.

Top Screenshot: The code shows the initial setup of the game. It includes a `Player` constructor, a `Deck` class, and a `Game` class. The `Game` class has a `compareCards` method that is currently empty. The code is written in JavaScript and uses ES6 syntax (e.g., `let`, `class`, `constructor`).

Bottom Screenshot: The code shows the implementation of the `compareCards` method. It uses a `for` loop to iterate through the cards in each player's hand. The method compares the rank of the cards, and if they are equal, it compares the suit. The winner of each round is determined, and the score is updated. The code is written in JavaScript and uses ES6 syntax (e.g., `let`, `class`, `constructor`).



Week 6: Final Project

The image displays two screenshots of a Visual Studio Code editor window, showing the development of a War Card Game. The top screenshot shows the game logic for comparing cards and updating scores. The bottom screenshot shows the final game initialization and the alert function to announce the winner.

Top Screenshot: The code editor shows the `warGame3.js` file. The `compareCards` function is being implemented. It uses `if` and `else if` statements to determine the winner based on the rank of the cards. The function updates the scores of the players and logs the results to the console.

```
Player > constructor
//conditional initiates if player 1 is the winner
}else if(player2.hand[round].rank > player1.hand[round].rank){
    player2.score++;
    console.log(`Round: ${round+1}
    ${player2.name} wins with ${player2.hand[round].value} of ${player2.hand[round].suit}
    ${player1.name} has ${player1.hand[round].value} of ${player1.hand[round].suit}
    ${player1.name} score: ${player1.score}
    ${player2.name} score: ${player2.score}
    `);
    //initiates if player2 is the winner
}else {
    console.log(`Round: ${round+1}
    Draw! Both players have ${player1.hand[round].value} of ${player1.hand[round].suit}
    ${player1.name} score: ${player1.score}
    ${player2.name} score: ${player2.score}
    `);
    //initiates if there is a tie;
}
console.log(player1.name + " score: " + player1.score);
console.log(player2.name + " score: " + player2.score);
//displays final scores
```

Bottom Screenshot: The code editor shows the `warGame3.js` file. The `Game` class is being implemented. The `compareCards` method is called to determine the winner. The `alert` function is used to display the winner's name.

```

    console.log(player1.name + " score: " + player1.score);
    console.log(player2.name + " score: " + player2.score);
    //displays final scores

    if(player1.score > player2.score) {
        alert("John is the winner!");
    }else if (player2.score > player1.score) {
        alert("Eric is the winner!");
    } else {
        alert("Game is a tie!");
    }
    //alerts the winner of the Game by displaying a pop up
}

let game = new Game();
game.compareCards();
```