BlackJack v1.0

Contenido

1. Introducción 1
1.1. Punto inicial 2
2. Primera fase. Funciones del juego 2
2.1. Definición de tipos de cartas 2
2.2. Variable game 2
2.3. Crear mazo 2
2.4. Mezclar cartas 3
2.5. Añadir array de cartas de jugadores 4
2.6. Crear jugada inicial 4
2.7. Calcular puntuación 4
2.8. Obtener nueva carta 6
2.9. Jugada del dealer 6
2.10. Jugada del jugador 7
2.11. Funciones auxiliares 7
2.12. Jugar una partida 8
3. Segunda fase. Crear el HTML 9
3.1. Objetos DOM 11
3.2. Limpiar datos 11
3.3. Mostrar y ocultar los botones del jugador 12
3.4. Mostrar el estado 12
3.5. Mostrar ganadores 12
4. Manejadores de evento 12
4.1. Nueva partida 12
4.2. Nueva carta 13
4.3. Me planto 13
5. Ejercicios 13

1. Introducción

El BlackJack es un juego de cartas que consiste en con las cartas a tu disposición, acercarte lo más posible a 21 sin pasarte. Compites contra el crupier o dealer en inglés. Su mecánica es realmente sencilla lo que lo hace ideal para práctica los diferentes elementos que tenemos en el lenguaje de forma inicial. Para más información de cómo se juega, puede consultar la el siguiente artículo de Wikipedia https://es.wikipedia.org/wiki/Blackjack.

1.1. Punto inicial

Creamos una página inicial sencilla, ya que vamos en un primer momento vamos a utilizar la consola del navegador.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>BlackJack</title>
</head>
<body>
    <h1>BlackJack</h1>
    <script src="js/blackjack.js"></script>
</body>
</html>
```

Como podemos ver tenemos un primer archivo **blackjack.js** situado en el directorio **js**. Puedes crear un directorio **css** porque lo utilizaremos más adelante.

La idea de la implementación será crear funciones que sean independientes de la página en la que estemos trabajando, para en una segunda fase podamos integrar estas funciones en un interfaz de página en concreto. Siempre debemos tener en cuenta en separar funciones reutilizables, de funciones que dependan de una página específica.

2. Primera fase. Funciones del juego

2.1. Definición de tipos de cartas

Creamos dos constantes para cada palo de la baraja francesa, y el valor que tiene cada una de ellas. Sabemos que estos arrays nunca van a cambiar.

```
const suits = [
  "Corazon", "Trebol", "Diamante", "Picas"];
const values = ["A","K","Q","J","10","9","8","7","6","5","4","3","2"];
```

2.2. Variable game

Vamos a almacenar en un objeto literal el estado del juego. En primera instancia solo va a contener el mazo de cartas para repartir. Este objeto lo ampliaremos posteriormente.

```
let game = {
  deck: []
}
```

2.3. Crear mazo

Vamos a añadir cada carta de la baraja a nuestro mazo. Cada carta será un objeto literal que tiene como propiedades `suit` (el palo) y el `value` (el valor). Recorremos constantes para crear el mazo.

```
function createDeck() {
```

```
  for (let suitIdx = 0; suitIdx < suits.length; suitIdx++) {
    for (let valueIdx = 0; valueIdx < values.length; valueIdx++) {
      let card = {
        suit: suits[suitIdx],
        value: values[valueIdx]
      };
      game.deck.push(card);
    }
  }
}
```

Vamos a crear una segunda función para chequear los avances hasta el momento y la invocamos para consultar los datos por la consola.

```
function playGame(){
  createDeck();
  console.log (game.deck);
}
playGame();
```

2.4. Mezclar cartas

Las cartas no están mezcladas, por lo que tenemos que crear una función. Básicamente vamos a recorrer el array, y generando un número aleatorio entre 0 y el número total de cartas, hagamos un intercambio en los valores de las posiciones utilizando una variable auxiliar.

La clase `Math` nos permite generar un número aleatorio decimal entre 0 y 1 con `random`. Si lo multiplicamos por el límite del número de cartas y lo truncamos, obtendremos un número entero entre 0 y el número de cartas menos 1. Revisar el siguiente artículo https://www.w3schools.com/js/js_random.asp para una explicación más detallada.

```
function shuffleDeck() {
  for (let i = 0; i < game.deck.length; i++) {
    let swapIdx = Math.trunc(Math.random() * game.deck.length);
    let tmp = game.deck[swapIdx];
    game.deck[swapIdx] = game.deck[i];
    game.deck[i] = tmp;
  }
}
```

Invocamos la función `playGame` para chequear los resultados.

```
function playGame(){
  createDeck();
  shuffleDeck();
  console.log (game.deck);
}
playGame();
```

2.5. Añadir array de cartas de jugadores

Vamos a empezar a crear el estado del juego. Añadimos a `game` los arrays para contener las cartas que ha obtenido el jugados y las cartas que ha obtenido el crupier, así como su puntuación.

```
let game = {
  dealerCards: [],
  playerCards: [],
  dealerScore: 0,
  playerScore: 0,
  deck: []
}
```

2.6. Crear jugada inicial

En la primera jugada se reparten dos cartas de forma alterna, que iremos consumiendo del taco de cartas en el objeto `game`, y asignándoselas a cada jugador.

```
function initialTurn() {
  for (let i = 0; i < 2; i++) {
    game.playerCards.push(game.deck.shift());
    game.dealerCards.push(game.deck.shift());
  }
}

function playGame() {
  createDeck();
  shuffleDeck();
  initialTurn();
  console.log(game.playerCards);
  console.log(game.dealerCards);
}
```

2.7. Calcular puntuación

Cada carta está ponderada según la siguiente tabla:

Carta	Puntuación
As	11 o 1 si la suma total supera 21.
Figuras, K, Q y J	10
Resto de cartas numéricas	Su valor numérico

Tabla 1 Ponderación de cartas

Vamos a crear una función que mediante un *switch* obtenga el valor de una carta recibida por parámetro.

```
function getCardNumericValue(card){
  switch (card.value){
    case "A": return 11;
    case "K": case "Q": case "J": return 10;
    default: return +card.value;
```

```
  }
}
```

Debemos recordad que la carta está representada por un *string*, por lo que la ponderación debe transformarlo en *number*, lo que hacemos con el operador unario +.

Para calcular la puntuación debemos recorrer el array de cartas de cada jugador recibido como parámetro, e ir sumando el valor de cada carta. Si nos fijamos en el bucle, vemos que también tenemos en cuenta el número de ases que hemos obtenido, ya que su valor cambiará en función de si nos hemos pasado de 21 o no.

El segundo bucle de esta función tiene como objetivo determinar el valor de los ases que disponemos. Por cada as que tengamos y nos hallamos pasado de 21, le restamos 10 a la puntuación para que el as tenga un valor de 1.

```
function getScore(cards) {
  let score = 0;
  let hasAce = 0;

  for (let i = 0; i < cards.length; i++) {
    score += getCardNumericValue(cards[i]);
    if (cards[i].value === "A") {
      hasAce++;
    }
  }

  while (hasAce > 0 && score > 21){
    score -= 10;
    hasAce--;
  }

  return score;
}
```

Una vez implementada la función de calcular puntuación, debemos actualizar `initTurn` con la puntuación de cada jugador. Debemos poner atención a las dos primeras sentencias de la función, utilizadas para reinicializar un array.

```
function initialTurn() {
  game.playerCards.length = 0;
  game.dealerCards.length = 0;
  for (let i = 0; i < 2; i++) {
    game.playerCards.push(game.deck.shift());
    game.dealerCards.push(game.deck.shift());
  }
  game.dealerScore = getScore(game.dealerCards);
  game.playerScore = getScore(game.playerCards);
}
```

Realizamos una el chequeo de las puntuaciones.

```
function playGame (){
  createDeck();
  suffleDeck();
  console.log(game.deck);

  initialTurn();
  console.log(game.playerCards);
  console.log(game.dealerCards);

  console.log(getScore(game.playerCards));
  console.log(getScore(game.dealerCards));
}
playGame ();
```

2.8. Obtener nueva carta

Creamos la acción de conseguir una nueva carta para el jugador, sacándola del mazo de cartas.

```
function getNextCard(cards) {
  cards.push(game.deck.shift());
}
```

Vamos a emular una partida para comprobar los resultados. Creamos un bucle que vaya obteniendo nuevas cartas mientras la puntuación sea menor de 21.

```
function playGame() {
  createDeck();
  shuffleDeck();
  initialTurn();
  console.log(game.playerCards);
  console.log(game.dealerCards);

  while (!(getScore(game.playerCards) > 21)){
    getNextCard(game.playerCards);
    console.log(getScore(game.playerCards));
  }
  console.log(game.playerCards);
}
```

2.9. Jugada del dealer

El dealer debe obtener una carta siguiendo las siguientes premisas:

- La puntuación del jugador sea menor o igual que 21, de lo contrario, se habría pasado y el dealer no necesita seguir jugando.
- Su puntuación sea menor que la del jugador. El dealer juego mientras vaya perdiendo. Esta es la opción ganadora para el dealer.
- Su puntuación sea menor de 21. El dealer juega hasta que se pase. Esta es la opción perdedora para el dealer.

```
function playDealer(){
  game.dealerScore = getScore(game.dealerCards);
  while (game.playerScore <= 21 &&
    game.dealerScore < game.playerScore &&
    game.dealerScore < 21){
    getNextCard(game.dealerCards);
    game.dealerScore = getScore(game.dealerCards);
  }
}
```

Testeamos la función asignando previamente una puntuación alta al jugador para ver las posibles opciones que tiene el dealer.

```
function playGame() {
  createDeck();
  shuffleDeck();
  initialTurn();
  console.log(game.playerCards);
  console.log(game.dealerCards);
  console.log(getScore(game.playerCards));
  console.log(getScore(game.dealerCards));

  game.playerScore = 18;
  playDealer();
  console.log("-------");
  console.log(game.dealerCards);
  console.log(game.dealerScore);
}
```

2.10. Jugada del jugador

El jugador tan solo tiene que esperar a que el usuario de la aplicación decida si quiere una nueva carta o se planta. Esta función obtiene una nueva carta del mazo y calcula la puntuación en base a ella.

```
function playPlayer(){
  getNextCard(game.playerCards);
  game.playerScore = getScore(game.playerCards);
}
```

2.11. Funciones auxiliares

Otras funciones que tendremos que utilizar para integrar el juego con la interfaz son las siguientes. En primer lugar, tenemos una función para transformar una carta en un string.

```
function cardsToString(cards){
  let str = "";
  for (let i = 0; i < cards.length; i++) {
    str += cards[i].suit + "-" + cards[i].value + "; ";
```

```
  }
  return str;
}
```

En el juego real, una jugada “blackjack” se da cuando obtenemos 21 con las cartas iniciales. La siguiente función nos indica si tenemos blackjack mediante un operador ternario.

```
function checkBlackJack(cards){
  return (getScore(cards) === 21)? true : false;
}
```

Esta función nos dice si el jugador se ha pasado al obtener una nueva carta, con lo que no puede seguir jugando.

```
function isPlayerScoreUpperLimit(){
  return (game.playerScore >= 21);
}
```

Las dos últimas funciones nos chequean si el jugador o el dealer a ganado comparando las puntuaciones. Tenemos que tener en cuenta que ambos pueden ganar si tiene el mismo resultado, por ejemplo, ambos tienen 21.

```
function checkPlayerWinner(){
  return (game.playerScore <= 21 &&
    (game.dealerScore > 21 ||
    game.playerScore >= game.dealerScore))? true : false;
}

function checkDealerWinner(){
  return (game.dealerScore <= 21 &&
    (game.playerScore > 21 ||
    game.dealerScore >= game.playerScore))? true : false;
}
```

2.12. Jugar una partida

Actualizamos la función para que solamente cree el mazo de cartas, las mezcle y cree la jugada inicial. A partir de ese momento tenemos que interactuar con el usuario para saber si se planta o pide una nueva carta.

```
function playGame() {
  createDeck();
  shuffleDeck();
  initialTurn();
}
```

3. Segunda fase. Crear el HTML

Creamos un nuevo fichero **balckjackDOM.js** para contener las funciones dependientes de la página.

- Creamos el esqueleto HTML.
- Estará hecho con Bootstrap.

Modificamos el HTML por el siguiente.

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <title>BlackJack</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/boots
trap/4.5.2/css/bootstrap.min.css"
        integrity="sha384-
JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z" crossor
igin="anonymous">
    <!-- Google Fonts -->
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:300,400
,600,700" rel="stylesheet">
    <!-- Font Awesome -->
    <link href="fontawesome/css/all.min.css" rel="stylesheet">

    <!-- Estilos locales -->
    <link href="css/main.css" rel="stylesheet">
</head>

<body>

    <!-- Header -->
    <header>
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-
top">
            <div class="container">
                <a class="navbar-brand" href="#">BlackJack V1.0</a>
            </div>
        </nav>
    </header>

    <!-- CATEGORIES -->
    <main class="container p-5">
        <h1>Juego del BlackJack</h1>
        <div class="row">
            <div class="col-md-6 col-12">
                <h3 class="display-5 py-3">Crupier</h3>
```

```
                <div id="cards-dealer" class="py-2">cartas</div>
                <div id="score-dealer" class="py-2">Puntuación</div>
                <div id="result-dealer" class="py-2">Resultado</div>
            </div>
            <div class="col-md-6 col-12">
                <h3 class="display-5 py-3">Jugador</h3>
                <div id="cards-player" class="py-2">cartas</div>
                <div id="score-player" class="py-2">Puntuación</div>
                <div id="result-player" class="py-2">Resultado</div>
                <div id="player-buttons" class="py-2 d-none">
                    <button id="hit-button" type="button" class="btn btn-
primary m-2">Nueva carta</button>
                    <button id="stay-
button" type="button" class="btn btn-primary m-2">Me planto</button>
                </div>
            </div>
        </div>
        <div class="row">
            <button id="new-game-button" type="button" class="btn btn-
primary">Nueva partida</button>
        </div>
    </main>

    <footer class="bg-dark text-white text-center p-3" id="footer">
        Powered by <strong>Pablo Lizano Montalvo</strong>.
    </footer>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->

    <script src="https://code.jquery.com/jquery-3.5.1.min.js"
        integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
        crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/
umd/popper.min.js"
        integrity="sha384-
ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bo
otstrap.min.js"
        integrity="sha384-
B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV"
        crossorigin="anonymous"></script>

    <script src="./js/blackjack.js"></script>
    <script src="./js/blackjackDOM.js"></script>

</body>
```

```
</html>
```

También tenemos unos estilos personalizados en el fichero **main.css**.

```
*, ::after, ::before {
    box-sizing: border-box;
}
html {
    font-size: 16px;
    scroll-behavior: smooth;
}
body {
    font-family: 'Open Sans', sans-serif;
    line-height: 1.42857143;
    color: #364147;
    background-color: #ffffff;
    padding-top: 68px;
}
```

3.1. Objetos DOM

Primero creamos los objetos DOM que vamos a utilizar para interactuar con el usuario en la página. Como podemos observar, hemos agrupado en un objeto literal los elementos del dealer y del jugador respectivamente, para mostrarles información. El identificador *playersButtons* hace referencia al `div` que contiene los botones del jugador, y tenemos un objeto por cada tipo de botón, uno para una nueva partida, otro para que el jugador pida una carta, y por último, para que se plante.

```
// DOM variables
const dealerDOM = {
  cards:document.getElementById("cards-dealer"),
  score:document.getElementById("score-dealer"),
  result:document.getElementById("result-dealer")
}
const playerDOM = {
  cards:document.getElementById("cards-player"),
  score:document.getElementById("score-player"),
  result:document.getElementById("result-player")
}
const playerButtons = document.getElementById("player-buttons");
const newGameButton = document.getElementById("new-game-button");
const stayButton = document.getElementById("stay-button");
const hitButton = document.getElementById("hit-button");
```

3.2. Limpiar datos

Creamos una función para limpiar los datos para cada partida mostrados en las áreas de cada jugador.

```
function cleanGame(){
  dealerDOM.cards.innerHTML = "";
  dealerDOM.score.innerHTML = "";
```

```
  dealerDOM.result.innerHTML = "";
  playerDOM.cards.innerHTML = "";
  playerDOM.score.innerHTML = "";
  playerDOM.result.innerHTML = "";
}
```

3.3. Mostrar y ocultar los botones del jugador

Estas funciones muestran u ocultan los botones del usuario en función de si hemos empezado una nueva partida o no. Utilizan la clase *d-none* de Bootstrap para hacerlo.

```
function showGameButtons(){
  newGameButton.classList.add("d-none");
  playerButtons.classList.remove("d-none");
}

function hideGameButtons(){
  newGameButton.classList.remove("d-none");
  playerButtons.classList.add("d-none");
}
```

3.4. Mostrar el estado

Esta función muestra el estado del juego en cada petición de cartas del usuario.

```
function showStatus(){
  playerDOM.cards.innerHTML = cardsToString(game.playerCards);
  playerDOM.score.innerHTML = getScore(game.playerCards);
  dealerDOM.cards.innerHTML = cardsToString(game.dealerCards);
  dealerDOM.score.innerHTML = getScore(game.dealerCards);
}
```

3.5. Mostrar ganadores

Muestran si los jugadores han ganado o no en la página.

```
function showWinners(){
  playerDOM.result.innerHTML = checkPlayerWinner()? "Ganador":"Perdedor";
  dealerDOM.result.innerHTML = checkDealerWinner()? "Ganador":"Perdedor";
}
```

4. Manejadores de evento

Cada botón de la página tiene una funcionalidad asignada para interactuar con el usuario.

4.1. Nueva partida

Este manejador inicia una nueva partida al ser cliqueado, limpiando los resultados actuales de la página y mostrando los botones del jugador, para finalmente, mostrar el estado del turno inicial.

```
newGameButton.addEventListener("click", function() {
```

```
  playGame();
  cleanGame();
  showGameButtons();
  showStatus();
});
```

4.2. Nueva carta

Si lo cliqueamos el jugador solicita una nueva carta. Chequeamos que no nos hemos pasado, ya que si lo hemos hecho la partida termina y mostramos los ganadores.

```
hitButton.addEventListener("click", function() {
  playPlayer();
  if(isPlayerScoreUpperLimit()){
    playDealer();
    showWinners();
    hideGameButtons();
  }
  showStatus();
});
```

4.3. Me planto

El botón para plantase desencadena la jugada del dealer, para posteriormente mostrar los resultados y ocultar los botones ya que la partida ha finalizado.

```
stayButton.addEventListener("click", function() {
  playDealer();
  showStatus();
  showWinners();
  hideGameButtons();
});
```

5. Ejercicios

1. Utilizar más de un mazo para repartir las cartas. Ejemplo 4 o 6.
2. Solo renovar el mazo en una partida nueva cuando el número de cartas sea menor de 15.
3. En inicio, no muestres la segunda carta del dealer ni su puntuación total, hasta que el jugador haya terminado su partida.
4. Si el dealer o el jugador tienen blackjack, 21, en el inicio la partida termina.