PAC Actividad de desarrollo (UF2) Diseño de Interfaces web

Redactado por:

Daniel Arturo Berroterán

Profesora:

Laura Bermúdez

El juego de la oca

Introducción

Para el desarrollo de esta actividad se ha utilizado los lenguajes ,**HTML5**, **CSS3** y **JavaScript** sin librerías separados cada uno en sus respectivos ficheros, para la organización del código se planteo usar un esquema de clases y así mejorar en la medida de lo posible la legibilidad y el orden del código fuente.

Para facilitar la ejecución y revisión de la actividad se evito usar un compilador como por ejemplo **Nodejs** aunque las clases están separadas cada una en sus respectivos archivos, no se han importado en los sitios en los que van usarse si no que se han copiado y pegado, evidentemente esto no esta bien, pero se ha hecho de esta manera para evitar usar módulos y así evitar usar **Nodejs** o algún otro tipo de compilador, lo que permite ejecutar la aplicación simplemente arrastrando el archivo **index.html** al navegador.

Se ha planteado que para el juego de la oca hubiera un jugador principal y el resto de jugadores fueran escogidos de forma aleatoria por la aplicación, y se ha usado el **LocalStorage** para guardar todos los datos referentes a las preferencias de partida del jugador principal, la información de los jugadores secundarios, las posiciones de cada jugador y si han llegado ya a la meta o no.

En los siguientes apartados se explicará de forma resumida los puntos mas importantes del funcionamiento del juego acorde con lo pedido en la rubrica, para hacer un análisis exhaustivo del funcionamiento del juego se deberá revisar de forma minuciosa el código fuente.

1) :Organización de carpetas del repositorio:

Para la actividad se ha planteado el siguiente orden de carpeta:

-- assets/

- – css/
- – images/
- – music/

– src/

- – controllers/
- – entitys/

– views/

En la carpeta **assets** se guardara la hoja de estilos **css** junto con los archivos de imágenes y sonidos cada uno en su propia carpeta. En la carpeta **src** se guardará la lógica del juego separando el código que controla la lógica de las vistas y las clases que se guardan el la carpeta **entitys.**

Por ultimo tenemos un fichero de vistas donde tendremos solo dos vistas **index.html** y **board.html**

2) El array de jugadores:

Tras iniciar una nueva partida se crea un array llamado **matchReport** con la siguiente estructura:

[[1,0,false] , [2,0, false] , [3,0, false] , [4,0,false]]

donde cada array dentro de **matchReport** representa la información del los jugadores que van a participar en la partida, el primer valor es la ficha del jugador, el segundo valor es su posición dentro del tablero y el ultimo valor indica si el jugador ha concluido o no su recorrido dentro del tablero.

Este array llamado **matchReport** se crea usando el siguiente método dentro de la clase **Match**:

```
static createMatchReport() {
  let playerPreferences = localStorage.getItem("playerPreferences");
  let playerPreferencesObject = JSON.parse(playerPreferences);
  let numberOfPlayer = playerPreferencesObject.playerNumberPreference;
  let matchReportArray = [];
  for (let i = 1; i < numberOfPlayer + 1; i++) {
   let playerInfo = [i, 0, false];
   matchReportArray.push(playerInfo);
  }
  localStorage.setItem("matchReport", matchReportArray);
 }
```

3) Uso del array de jugadores:

El array **matchReport** se guarda en el **LocaStorage** y se utiliza para guardar los datos de la partida independientemente de si se ha hecho una recarga de la página y se borra o mas bien se reinicia cuando se le da al botón de Reiniciar partida.

Este array a su vez cambia de estado con la jugada de cada jugador actualizando su posición y si el jugador ha terminado su recorrido por el tablero: la variable booleana dentro del array permite que un jugador no siga avanzando hasta después de haber alcanzado la casilla 60.

Actualización de **matchReport:**

```
static updateMatchReport(newTokenPosition) {
  let tokenPosition = [];
  let matchReport = convertToArray();
  for (let i = 0; i < matchReport.length; i++) {
   if (i === newTokenPosition[0] - 1) {
    matchReport[i][1] = newTokenPosition[1];
    tokenPosition.push(matchReport[i]);
    break;
   }
  }
  localStorage.setItem("matchReport", matchReport);
 }
```

4) El tamaño y ubicación de las imágenes:

Cómo se ha dicho antes todas las imágenes usadas en el proyecto se guardan dentro de la carpeta **assets/images**, las imágenes proporcionadas por el enunciado de la tarea han sido retocadas con **Paint** y **https://www.photopea.com/** a la hora de añadir las imágenes dentro de las vistas **HTML** se les ha otorgado en cada caso un ancho y un alto conveniente al diseño dentro de la misma etiqueta **<img/>** ejemplo:

```
mainPlayer.innerHTML = `
 <img class="p-1" src="../assets/images/${playerCharacter}.jpg" width="100px"
height="100px" alt="imagen-jugador"/>
```

5) El desplegable de jugadores:

En la vista index.html tenemos una barra de navegación con las siguientes opciones:

¿Cómo se juega?, Elige tu personaje, Elige el numero de jugadores.

Al pulsar en alguna de estas opciones aparece un desplegable con las opciones disponibles

el funcionamiento de los desplegables se encuentra en el archivo **menuController.js**, ha continuación se muestra una parte del código:

```
menu.forEach((option) => {
 let childrenOptions = option.querySelectorAll("ul");
 option.addEventListener("click", function () {
  childrenOptions.forEach((childrenOption) => {
   childrenOption.classList.toggle("deploy");
   let childrenOptionOfDropdown = childrenOption.querySelectorAll("li");
   childrenOptionOfDropdown.forEach(function (item) {
    item.addEventListener("click", function () {
     let itemId = item.id;
     saveItemSelected(itemId, hasAllAttributes);
    });
   });
  });
 });
});
```

La función **saveItemSelected()** procesa los datos que el usuario a seleccionado en el menú

6) Los botones de dados:

Para tirar los dados cada jugador tendrá un botón que se activará cuando sea su turno, cuando el botón esta activado se vera resaltado el texto: **Tirar dado** y cuando se coloque el cursor encima del botón mostrará claramente que se resaltará el botón cambiando de color tanto el texto como los bordes del botón dando a entender que se puede pulsar.

Los botones del resto de jugadores estarán deshabilitados y se verán claramente diferente al botón que si esta habilitado.

7) Los botones de dados funcionamiento:

Los botónes de tirar dado funcionan con el evento **onClick** que activara el método **rollDiceButtonClick(button)** este método recibirá como parámetro el elemento **HTML** utilizando el parametro this en la vista:

```
button id="rollDiceOne"class="roll-dice-btn align-self reset-btn btn"
onclick="rollDiceButtonClick(this)">
         Tirar dado</button
        ><div class="dice-result d-flex justify-content-center"><p id="diceNumberOne"
class="dice-number align-self">0</p></div>
```

Una vez pulsado el botón se desencadenará una serie de métodos que se muestran a continuación:

```
function rollDiceButtonClick(button) {
 let rollDiceResult = Player.rollDice();
 renderRollDiceResult(button, rollDiceResult);
 changePlayerTurn();
 let playerToken = selectPlayerToken(button);
 let tokenPosition = catchPlayertokenPosition(playerToken);
 let newTokenPosition = Player.moveToken(tokenPosition, rollDiceResult);
 DrawNewTokenPosition(newTokenPosition, coordinates);
 Match.updateMatchReport(newTokenPosition);
 let winners = Match.listWinners(listWinners);
 DrawListWinners(winners);
 showListWinners();
}
```

La función **rollDice()** de la clase **Player** genera un numero aleatorio del 1 al 6, luego con ese parámetro y el elemento **HTML** que hace referencia el botón se activa el método **renderRollDiceResult(button, rollDiceResult)** este método mostrará al lado del botón de tirar dados de cada jugador el numero aleatorio que se ha creado.

Luego se activa el método **changePlayerTurn(),** que se encarga de deshabilitar el botón pulsado y habilitara el botón del siguiente jugador, posteriormente el método **selectPlayerToken(button);** buscará la ficha que le corresponde al jugador que ha tirado los dados y el método **catchPlayerTokenPosition(playerToken)** devolverá la posición actual en la que se encuentra la ficha del jugador.

Con esta información ya se puede mover ficha con el método de la clase **Player** llamado **moveToken(tokenPosition, rollDiceResult);** se pasa como parámetro la posición de la ficha y el resultado del dado, este método determinará donde se tiene que mover la ficha y establecerá las reglas de la partida para cada jugador, luego devuelve la nueva posición de la ficha movida.

El método **DrawNewTokenPosition(newTokenPosition)** dibuja la ficha en la nueva posición que tiene en el tablero y luego se actualiza el array de jugadores **matchReport** con el método de la clase **Match** llamado **updateMatchReport(newTokenPostion)**

Si un jugador a llegado a la casilla 60 se actualizará el array de ganadores que se ha creado en el fichero **boardController.js** con el método de la clase **Match** llamado **listWinners(listWinners)** para finalizar el método **DrawListWinners(winners)** crea una tabla con el orden de llegada de los jugadores que llegan a la casilla 60, si solo falta un jugador por alcanzar la casilla 60 el método **showListWinners()** mostrará la tabla de los jugadores por orden de llegada.

8) El funcionamiento de los dados:

El método **rollDice()** crea un numero aleatorio del 1 al 6 de la siguiente manera:

```
static rollDice() {
  const diceResult = Math.floor(Math.random() * 6) + 1;
  return diceResult;
 }
```

se usa el método **random()** de la clase **Math** dentro del método de la misma clase **floor()** para evitar números decimales, al resultado se le suma 1 para evitar que pueda dar 0 como numero aleatorio.

9) Creación de las ficha de los jugadores:

Como se ha dicho en la introducción las fichas de los jugadores se crean de forma aleatoria pero la ficha del **jugador principal(usuario)** se crea en función del personaje que el usuario ha elegido en sus preferencias de partida, cada jugador es una instancia de la clase **Player** y se guarda en el **LocalStorage**, el jugador principal se crea antes de iniciar la partida y se guarda en el **LocalStorage** con la siguiente estructura:

{"_playerId":1,"_playerCharacter":"pinguino","_playerToken":1,"playerNumberPreference":4}

Lo importante ha resaltar en esta estructura de datos es el valor de **_playerCharacter** en el ejemplo se a elegido el personaje **pingüino,** dato que luego se utilizará para mostrar en la ficha la imagen que corresponda al pingüino, también es importante el valor de **_playerToken** que indica el numero de la ficha y se usará para identificar la ficha dentro del array de jugadores **matchReport,** y por ultimo cuantos jugadores ha elegido el usuario que van a participar en la partida, en este ejemplo se muestran el valor **4** en el campo **playerNumberPreference.**

Los métodos **createMainPlayerMenu();** y **createMainPlayerToken();** en el archivo **boardController.js** dibujarán el “cubilete” del jugador principal y la ficha dentro de la vista respectivamente:

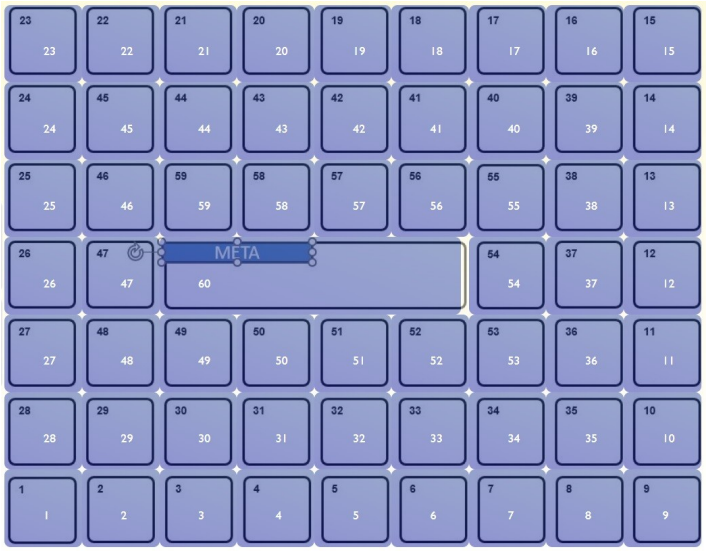
```
function createMainPlayerMenu() {
 let playerPreferences = localStorage.getItem("playerPreferences");
 let playerPreferencesObject = JSON.parse(playerPreferences);
 let playerCharacter = playerPreferencesObject._playerCharacter;
 mainPlayer.innerHTML = `
 <img class="p-1" src="../assets/images/${playerCharacter}.jpg" width="100px"
height="100px" alt="imagen-jugador"/>
 <div class="player-info d-flex flex-column">
 <h4>Personaje:</h4>
   <p>${playerCharacter}</p>
   <h4>Ficha:</h4>
   <p>Azul</p>
 </div>`;
 mainPlayerControl.classList.remove("hide");
}
```

```
//Dibuja la ficha del jugador principal:
function createMainPlayerToken() {
 let playerPreferences = localStorage.getItem("playerPreferences");
 let playerPreferencesObject = JSON.parse(playerPreferences);
 let playerId = playerPreferencesObject._playerId;
 let playerToken = document.getElementById(playerId);
 playerToken.classList.remove("hide");
 let playerCharacter = playerPreferencesObject._playerCharacter;
 playerToken.innerHTML = `
 <img class="token-img align-self" src="../assets/images/${playerCharacter}.jpg"
width="50px" height="50px" alt="imagen-jugador"/>
 `;
}
```

De una forma similar también se crean los “cubiletes” de los jugadores secundarios y sus respectivas fichas.

10) Movimiento de las fichas:

Para hacer que las fichas se muevan sobre el tablero se ha creado un tablero por debajo con el mismo diseño al de la imagen del tablero en forma de espiral:



Lo que podemos ver en la imagen de arriba es la imagen del tablero con una opacidad de 0.5 los cuadros azules que están debajo de la imagen son etiquetas que están posicionadas utilizando el método de maquetación con **grid-template-area:**

```
.board-skeleton {
  position: absolute;
  z-index: 0;
  right: 0;
  top: 50px;
  display: grid;
  grid-template-columns: repeat(9, 1fr);
  grid-template-rows: repeat(7, 1fr);
  grid-template-areas:
    "twenty-three twenty-two twenty-one twenty nineteen eighteen seventeen sixteen fifteen"
    "twenty-four forty-five forty-four forty-three forty-two forty-one forty thirty-nine fourteen"
    "twenty-five forty-six fifty-nine fifty-eight fifty-seven fifty-six fifty-five thirty-eight thirteen"
    "twenty-six forty-seven sixty sixty sixty sixty fifty-four thirty-seven twelve"
    "twenty-seven forty-eight forty-nine fifty fifty-one fifty-two fifty-three thirty-six eleven"
    "twenty-eight twenty-nine thirty thirty-one thirty-two thirty-three thirty-four thirty-five ten"
    "one two three four five six seven eight nine";
}
```

Usando este enfoque podemos obtener las coordenadas de cada casilla sin necesidad de tenerlas escritas directamente en el código(hardcodeadas), en su lugar en el fichero **BoardController.js** se crea una constante que va a contener un array de coordenadas y usando el método **mapBoardSkeleton()** de la clase **Board** se obtienen todas las coordenadas de cada casilla:

```
static mapBoardSkeleton() {
  let boardSkeleton = document.querySelectorAll("div.box");
  let coordinates = [];
  boardSkeleton.forEach(function (box) {
   let boxCoordinates = box.getBoundingClientRect();
   let boxNumber = box.innerHTML;
   let boxInfo = [boxNumber, boxCoordinates.left, boxCoordinates.top];
   coordinates.push(boxInfo);
  });
  return coordinates;
 }
```

En el archivo **BoardController.js:**

```
const coordinates = Board.mapBoardSkeleton();
```

con este array de coordenadas podemos pasar las coordenadas del tablero como parámetros al método **DrawTokenPosition(newTokenPosition, coordinates)**

```
function DrawNewTokenPosition(newTokenPosition, coordinates) {
 let newPositionString = newTokenPosition[1].toString();
 for (let i = 0; i < coordinates.length; i++) {
  if (coordinates[i][0] === newPositionString) {
   let token = newTokenPosition[0];
   if (token === 1) {
    let oneToken = document.querySelector("div.oneToken");
    let adjustLeft = coordinates[i][1];
    adjustLeft += 20;
    let adjustTop = coordinates[i][2];
    adjustTop += 20;
    oneToken.style.left = adjustLeft + "px";
    oneToken.style.top = adjustTop + "px";
   //Resto del código

   }
  }
 }
}
```

En el código de arriba podemos ver solo una parte del método, el funcionamiento merece ser explicado, primero recorremos el array de coordenadas que tiene la siguiente estructura **[[1,x,y], [2,x,y],[3,x,y]….]** en donde el primer valor es el numero de la casilla y los siguientes valores son sus ejes **x(izquierda)** y **y(tope)** por cada casilla en el array, si el numero de casilla coincide con la nueva posición de la ficha, se obtiene del **HTML** la ficha y le cambia sus estilos para que se dibuje en la casilla que le corresponda de acuerdo a su posición.

11) El botón reiniciar:

El método de la clase **Match createMatchReport()** crea el array de jugadores cuando se inicia la partida este método se vuelve ha invocar dentro del método de la misma clase **restartMatch()** y se refresca la página:

```
static restartMatch() {
  this.createMatchReport();
  window.location.reload();
 }
```

al pulsar en la opción de Reiniciar partida se invoca al método **restartMatch();**

```
resetMatch.addEventListener("click", function () {
 Match.restartMatch();
});
```

12) La tabla de ganadores:

En el archivo **BoardController.js** Se crea un array vació llamado **listWinners,** ha este array se le va añadiendo un nuevo registro cada vez que un jugador es declarado ganador, esto quiere decir que ha llegado a la casilla **60** que corresponde con la meta dentro del tablero en espiral. El método **declareWinner();** se activa dentro del método **moveToken()** cuando se cumple la condición anteriormente mencionada:

```
static moveToken(tokenPosition, diceResult) {

  let currentPosition = tokenPosition[1];
  let newPosition = currentPosition + diceResult;
  if (newPosition > 60 && !tokenPosition[2]) {
   let restOfPosition = 60 - currentPosition;
   let stepsToGoBack = diceResult - restOfPosition;
   newPosition = 60 - stepsToGoBack;
   tokenPosition[1] = newPosition;
   return tokenPosition;
  } else if (newPosition === 60) {
   {
    //Si la nueva posición es igual a 60:
    tokenPosition[1] = 60;
    let playerToken = tokenPosition[0];
    this.declareWinner(playerToken);
    return tokenPosition;
   }
  } else if (newPosition > 60 && tokenPosition[2]) {
    //Si la nueva posición es mayor que 60 y el jugador ya se ha declarado ganador
    //Se activa el código que evita que la ficha ganadora se siga moviendo
   tokenPosition[1] = 61;
   return tokenPosition;
  } else {
   tokenPosition[1] = newPosition;
   return tokenPosition;
  }
 }
```

Luego el método **declareWinner(playerToken)** busca en el array **matchReport** la ficha que esta jugando y le cambia el valor booleano que estaba por defecto en false a true para indicar que la ficha ya ha ganado:

```
static declareWinner(playerToken) {
  let matchReport = convertToArray();
  if (playerToken === 1) {
   matchReport[0][2] = true;
  } else if (playerToken === 2) {
   matchReport[1][2] = true;
  } else if (playerToken === 3) {
   matchReport[2][2] = true;
  } else {
   matchReport[3][2] = true;
  }
  localStorage.setItem("matchReport", matchReport);
 }
```

el método **convertToArray();** simplemente es un método auxiliar para sacar del **LocalStorage** el valor **matchReport** guardado cómo un String dentro del **LocalStorage** este método lo convierte en un array y lo devuelve.

Por ultimo el método **listWinners()** obtiene el array vació creado en el archivo **BoardController.js** y lo va llenado.

```
static listWinners(listWinners) {
  let matchReport = convertToArray();
  let isInclude = false;
  for (let i = 0; i < matchReport.length; i++) {
   isInclude = listWinners.includes(matchReport[i][0]);
   if (matchReport[i][2] && !isInclude) {
    listWinners.push(matchReport[i][0]);
    break;
   }
  }
  this.endMatch(listWinners);
  return listWinners;
}
```

al finalizar el bucle for se declara la partida finalizada con el método **endMatch()** que comprueba cuantos registros hay en el array de ganadores si, la partida es de 3 jugadores por ejemplo se declara la partida finalizada cuando dos jugadores estén incluidos en la lista de ganadores, y cuando la partida se establezca como finalizada se muestra la tabla de ganadores.

13) Nivel de dificultad del juego:

Para la tarea se uso el tablero en forma de espiral con 60 casillas y se estableció la regla de que si no se llegaba exactamente a la casilla 60, los movimientos restantes que tocara al jugador fueran de retroceso.

Esto se puede ver perfectamente como una condición dentro del método **moveToken():**

```
static moveToken(tokenPosition, diceResult) {
  //Resto del código
  if (newPosition > 60 && !tokenPosition[2]) {
   let restOfPosition = 60 - currentPosition;
   let stepsToGoBack = diceResult - restOfPosition;
   newPosition = 60 - stepsToGoBack;
   tokenPosition[1] = newPosition;
   return tokenPosition;
  }
//Resto del código
}
```

14) Extras:

Se puede reproducir un sonido de fondo para jugar en la vista **board.html**, solo hay que pulsar el botón Reproducir música y para silenciar se pulsa en Quitar musica:

board.html:

```
<audio id="boardTheme" autoplay loop>
   <source src="../assets/music/match-theme.mp3" type="audio/mpeg" />
  </audio>
```

```
<div id="playMusic" class="option p-1">
    <p>Reproducir musica</p>
   </div>
 <div id="Mute" class="option p-1">
    <p>Quitar musica</p>
</div>
```

boardController.js:

```
//Reproduce musica:
playMusic.addEventListener("click", function () {
 let audio = document.getElementById("boardTheme");
 audio.play();
});

//Silencia musica:
Mute.addEventListener("click", function () {
 let audio = document.getElementById("boardTheme");
 audio.muted = !audio.muted;
});
```