Nombre: Víctor
Apellidos: Valdés Cobos
Desarrollo web en entorno servidor

ACTIVIDAD PRÁCTICA 1 DWEC

Reglas del juego:

Consiste en ordenar las bolas por color, uno en cada frasco, antes de que el contador llegue a cero. No se puede poner una bola encima de otra con distinto color. Para poder jugar al juego, como mínimo uno de los frascos tienen que empezar vacíos y los demás tienen que empezar llenos.

Algunos Ball Sort Puzzle también añaden un sistema de puntos, y con cada movimiento vas perdiendo puntos, y si llegas a cero puntos pierdes la partida; yo no he añadido esta función.

Planteamiento principal del código:

Antes de generar el tablero hay una breve introducción, jugando con setTimeOut() y eventos, con esto consigo que el programa espere a la interacción del usuario y que vaya ejecutándose código cada cierto tiempo.

Después de una breve introducción, genero un TAD tablero que consiste en un array frascos que cada uno contiene un array frasco, no es una matriz, es un array que contiene arrays. Para crear esta estructura hubiera preferido usar pilas en los frascos pero nunca he usado pilas en JS y probablemente hubiera tenido problemas.

Para que se genere correctamente el tablero inicial, se tienen que generar tantos colores como frascos llenos haya, y el mismo número de bolas de cada color, este numero es igual a la capacidad máxima de un frasco, asique para hacer esto, al principio lleno cada frasco de un color y después hago un recorrido para “barajar estas bolas” como si de una baraja se tratase (siempre dejo dos frascos vacíos).

Cada vez que muevo una bola, lo que realmente cambia es la estructura de arrays, y mediante una función, actualizo el entorno gráfico en base a esta estructura de arrays. Cada vez que realizo un movimiento compruebo si el tablero se ha ordenado, en caso de que sí, compruebo si se ha ganado la partida o si se pasa al siguiente nivel, en caso contrario, espero al siguiente movimiento.

Los movimientos de bola se pueden diferenciar por erróneos o correctos, erróneos serían movimientos que no cambian el TAD, como sacar y meter una bola en el mismo frasco, encima de un color distinto a la bola sacada, o meter una bola en un frasco lleno, estos tres serían todos los movimientos erróneos; un movimiento correcto sería meter una boa en un frasco distinto al original y que este no esté lleno y que, si hay bolas, su bola que se encuentra por encima de las demás sea del mismo color que la bola a meter.

En caso de que se pase al siguiente nivel, se borra y se genera el TAD y se actualiza el entorno, y en caso de que se gana la partida se genera una secuencia con motivo de explicar un breve final y, clicando un botón puedes jugar de nuevo.

El juego dispone de sonidos cada vez que realizas un movimiento erróneo o correcto, pasas de nivel, ganas o pierdes, además de un sonido de fondo que se repite en bucle.

El juego solo se puede jugar clickando en los frascos, he elegido este control ya que es muy intuitivo y es muy compatible con cualquier dispositivo.

Funciones:

inicioIntro():

La primera función que se ejecuta es inicioIntro(), cuando se carga la página. Esta función ejecuta la secuencia de introducción, llamo secuencia a algo que sucede cada cierto tiempo y sin necesidad de que el usuario interactúe. Esta secuencia presenta el juego y al final añade un botón para que el usuario comience el juego.

createTextoTitulo ():

Recibe dos variables opcionales y automáticamente crea y añade en pantalla los dos parámetros, que serán alfanuméricos.

cargaPersonaje ():

Carga la información de la imagen del protagonita Theo para mostrarlo en pantalla

iniciaEntorno ():

Resetea la partida, funciona como una especie de breakpoint y devuelve todos los valores a su valor inicial. En esta función se puede modificar la configuración del juego (MAXniveles, MAXfrascos y MAXbolas pueden ser modificados al gusto).

```
//configuración de constantes.
tiempo = 40;
nivel = 0;
MAXniveles = 3;
MAXfrascos = 4;
MAXbolas = 4;
```

En esta función se controla el audio de fondo, el cual es una variable global:

```
if (!audioBg.paused) {
    audioBg.pause();
    audioBg.currentTime = 0;
}
```

Variable global:

```
20    let audioBg = new Audio("./assets/audio/musicaBackground.wav");
21    audioBg.volume = 0.1;
22    audioBg.loop = true;
```

actualizarEntorno ():

Recibe dos variables opcionales. Esta función actualiza el entorno gráfico.

Si recibo dos parámetros actualizo dos frascos en comparación al TAD, dadas sus posiciones por los dos parámetros de la función. editarEntorno()

Si no recibo dos parámetros, significa que estoy generando el mismo, actualiza el entorno añadiendo el evento al contenedor de los frascos. generarEntorno()

tableroRand ():

Contiene dos recorridos de dos dimensiones cada uno.

Esta función presupone que el TAD está vacío para el correcto funcionamiento del juego.

Como menciono en el inicio del documento, esta es la función que se encarga de la aletoriedad de las bolas, primero genero una bola de cada color para cada frasco y posteriormente las 'barajo':

```
//Siempre dejo dos frascos vacíos para poder jugar
for (let index = 0; index < MAXfrascos; index++) {
    bolasFrascos = [];
    arrayFrascos.push(bolasFrascos);
    if (index < MAXfrascos - 2) {
        for (let index2 = 0; index2 < MAXbolas; index2++) {
            bolasFrascos.push(index);
        }
    }
}
```

```
    do {
        //Barajo las bolas tantas veces como bolas por frascos haya
        for (let index = 0; index < MAXfrascos * (MAXfrascos - 2); index++) {
            const randomFrasco1 = Math.floor(Math.random() * (MAXfrascos - 2));
            const randomBola1 = Math.floor(Math.random() * MAXbolas);
            const randomFrasco2 = Math.floor(Math.random() * (MAXfrascos - 2));
            const randomBola2 = Math.floor(Math.random() * MAXbolas);

            let aux = arrayFrascos[randomFrasco1][randomBola1];
            arrayFrascos[randomFrasco1][randomBola1] =
                arrayFrascos[randomFrasco2][randomBola2];
            arrayFrascos[randomFrasco2][randomBola2] = aux;
        }
    } while (recorridoGanado()); // si invierto la comprobación, el tablero comienza ordenado
    // Compruebo que el tablero generado no se genera comienza ordenado
}
```

Barajo las bolas tantas veces como bolas por frascos menos dos haya, esto no quiere decir que todas las bolas sean barajadas, ya que cuando barajas una bola, en realidad cambias de posición 2 bolas por lo que, en realidad cada bola tiene dos oportunidades para ser barajada en una situación ideal. Esta condición me parece relativamente óptima ya que establece un equilibrio entre el tamaño del TAD y la aletoriedad del juego.

Uso mi función recorridoGanado() en el do while para comprobar que la partida no comienza ordenada, cosa que puede ocurrir cuando hay pocas bolas.

Si añado un '!' antes de recorridoGanado() la partida comienza ganando, cosa que puede ayudar para el desarrollo del juego.

accionFrasco ():

Contiene gran parte de la lógica del juego controla lo que pasa cuando queremos realizar un movimiento de bola, en este caso el mayor reto ha sido refactorizar el código y no repetir código, ya que al trabajar con eventos en referencia tengo que controlar los clicks, a una bola, a un frasco y fuera de los anteriores.

Esta función primero edita el TAD y luego actualiza el entorno gráfico mediante actualizar entorno.

Mediante una variable booleana click gestiono si saco o meto bola, cuando saco una bola calculo cuantas bolas quedan para que el frasco se llene, lo multiplico por 40 píxeles y le sumo 75 píxeles aprox, esto se lo resto a la propiedad top de la bola para que la bola quede 75 píxeles aprox encima del frasco (Sin contar posibles paddings o margins).

```
    let numeroBolas = arrayFrascos[posAnterior].length; //Cada bola mide 40px
    //La bola ha de estar 75 px encima del frasco aprox (sin contar posible margin o padding)
    e.target.parentNode.firstChild.style.top =
        "-" + ((MAXbolas - numeroBolas) * 40 + 75).toString() + "px";
```

creaBotonStart ():

Esta función carga un botón en la página que se usará para comenzar el juego, primero crea un section, dentro creará un a y por último un img.

apagarTemporizador ():

Resetea el intervalo de 'intervalo' que es el que controla el tiempo en el juego, y pone el texto del contador a 00:00.

inicioFin ():

Inicia la secuencia final, la cual varía si has perdido o ganado, al final de esta, creo un botón y puedes repetir la partida.

recorridoGanado ():

Función que devuelve si el nivel ha sido superado, llamo a esta función cada vez que realizo un movimiento que no sea erróneo, consiste en un recorrido de dos dimensiones un for y un while. El for recorre todos los frascos, y si el frasco tiene tantas bolas como el máximo de bolas permitidas, guardo el valor de la primera bola y recorro mediante un while hasta que la bola comprobada no corresponda al color de la guardada, si esto ocurre la función devuelve false directamente.

borraEntorno ():

Elimina la capa contenedora de frascos

cieloColor ():

Gestiona las clases que a su vez añaden una animación que cambia el color de la capa div.visor_bg y div.container__visor. En el caso de visor__bg lo que cambia es el porcentaje de mezcla del filtro que aplica blanco y negro.

Cuando se repite la partida, hay que limpiar las clases previamente añadidas y reestablecer todo de por defecto, esto me dio problemas asique gestiono las clases de esta forma

Debido que classList no devuelve un array uso la sintaxis […] que convierte todo en un array y me permite usar funciones callback.

```
      cielo.classList = [...cielo.classList].filter(
        (clase) =>
          clase != "cieloAzul" &&
          clase != "cieloInicial" &&
          clase != "cieloGris" &&
          clase != "cieloNaranja"
      );
      cielo.classList.add("cieloGris");

      bg.classList = [...bg.classList].filter(
        (clase) =>
          clase != "fondoGris3" &&
          clase != "fondoGris2" &&
          clase != "fondoGris1"
      );
      bg.classList.add("fondoGris1");
      break;
```

ganar ():

Gestiona lo que pasa cuando se gana el nivel, pueden pasar dos cosas: Que se gane la partida o que se pase de nivel.

```
if (nivel == MAXniveles) {
  audioGanar();
  borraEntorno();
  tiempo = 0;
  cieloColor(2);
  inicioFin(ganado);
} else {
  audioLevelUp();
  MAXbolas++;
  MAXfrascos++;
  borraEntorno();
  tableroRand();
  actualizarEntorno();
  cieloColor(nivel);
  tiempo = tiempoBase + 10 * nivel; //cada nivel aumenta 10 segundos
}
```

inicioTempo ():

Esta función controla el tiempo del juego.

Trabaja mediante un set Interval que tiene una única vida por partida, es decir, empieza cuando empieza la partida y termina cuando se gana, se pierde o el tiempo llega a cero.

Cuando se invoca por primera vez esta función, se aplica una animación al temporizador para que este baje a la pantalla

Para aplicar el estilo de temporizador uso padStart():

```
let minutos = Math.floor(tiempo / 60).toString();
let segundos = Math.floor(tiempo % 60).toString();

visorHeader__div__temporizador.children[1].textContent =
    minutos.padStart(2, "0") + ":" + segundos.padStart(2, "0");
```

Funciones de audio:

Todas son iguales:

```
const audioMovimientoAtras = () => {
  let audio = new Audio("./assets/audio/movimientoFallido.wav");
  audio.volume = 0.4;
  audio.play();
};
```

Asigno el path, el volumen y le asigno play(). Hay un audio para cada situación del juego.

Estilos:

El estilo del juego en general consiste en un fondo con textura de madera pixel y el visor en el centro dejando un margen.

```css
.container{
    margin: auto;
    display: grid;
    background-image: url("../assets/jpg/simon-sanchez-madera-tilex1.jpg");
    background-repeat: repeat;
    background-size: 370px;

    width: 1300px;

    grid-template-columns: 1fr 8fr 1fr;
    grid-template-rows: 100px 700px 100px;

    grid-template-areas:
    "header header header"
    "asidel visor asider"
    "footer footer footer"
    ;
    border: 10px solid black;


}
```

Aplico un borde que le da un toque arcade/voxel al entorno gráfico.

Añado unas nubes que se van moviendo en un patrón bucle:

```css
.container__nubes--1{
    position: absolute;
    z-index: 0;
    top: 50px;
    left: 0;
    height: 150px;
}

.animacion__infinito{
    animation: nubes 60s linear infinite;
    animation-delay: -15s;
}

@keyframes nubes {
    from{
        transform: translateX(+900px);
    }to{
        transform: translateX(-200px);
    }
}
```

Cada aniamción del cielo consiste en esto:

```css
.container__visor{
    position: relative;
    background-size: cover;
    background-repeat: repeat-y;
    z-index: 0;
    background-color: rgb(175, 226, 255);
    /* podría ser overflow:hidden y alomejor inherit */
    border: 10px solid black;
}
```

```css
.cieloGris{
    animation: cieloGris;
    animation-duration: 2s;
    animation-fill-mode: forwards;
}

.cieloNaranja{
    animation: cieloNaranja;
    animation-duration: 4s;
    animation-fill-mode: forwards;
}

.cieloAzul{
    animation: cieloAzul;
    animation-duration: 4s;
    animation-fill-mode: forwards;
}

.cieloInicial{
    animation: cieloInicial;
    animation-duration: 4s;
    animation-fill-mode: forwards;
}
```

```css
@keyframes cieloInicial {
    from{
        background-color: rgb(175, 226, 255);
    }to{
        background-color: black;
    }
}

@keyframes cieloGris {
    from{
        background-color: black;
    }to{
        background-color: grey;
    }
}

@keyframes cieloNaranja {
    from{
        background-color: grey;
    }to{
        background-color: rgb(115, 131, 141);
    }
}

@keyframes cieloAzul {
    from{
        background-color: rgb(115, 131, 141);
    }to{
        background-color: rgb(175, 226, 255);
    }
}
```

Este es el filtro usado para el blanco y negro:

```css
@keyframes fondoGris1 {
    from{
        filter: grayscale(0%);
    }to{
        filter: grayscale(100%);
    }
}
```

Cabe destacar que esta función parece ser heredativa.

Contexto:

El contexto del juego trata sobre un protagonista llamado Theo que, en su viaje por el bosque pierde todas su canicas de colores, debido a esto deja de ver colores, a medida que pasando de nivel vas ayudando a Theo a recuperar los colores.

Notas:

Pienso que el tamaño del juego visualmente en general es un poco grande: 1300px de ancho y 900px de alto: Para solucionar esto de la mejor forma intenté usar transformaciones gestionadas desde JS que cambiaban el valor pasado por scale() del body o del container, y conseguía efecto pero se descuadraban las posiciones en pixeles del fondo, de las montañas… Para esto necesitaría una función que de alguna forma reescale cada pixel de la pantalla, aunque esto pueda consumir recursos y pueda afectar a la resolución del juego, solo se accionaría cuando el usuario carga la página o cuando intenta zoomear o dezoomear. Con esta función el juego podría jugarse en muchas más plataformas. De momento el usuario que juega en PC probablemente tenga que dezoomear un 10 por ciento aproximadamente si tiene una pantalla convencional estándar y en móvil es posible jugar pero con problemas visuales en el modo horizontal, ya que en navegadores de móviles, normalmente no se puede dezoomear fácilmente.

La función accionFrasco() podría refactorizarse de tal forma que no repita código ya que uno de mis problemas ha sido no poder controlar bien los eventos de dos target que se parecen mucho, ambos son div pero, como los eventos pasan por referencia, me he encontrado con problemas para controlar los dos eventos (el de la bola y el del frasco) por igual y poder refactorizar, hay muchas formas para conseguirlo pero todas ellas requerían de bastante tiempo. La forma fácil hubiera sido hacer muchas funciones distintas pero prefiero hacer la mitad de funciones y que controlen por igual los clicks hacia bolas que hacia frascos.

La función de github pages, para hostear el juego me ha ayudado con los path para las imágenes, ya que cuando usas live server los path funcionan pero cuando los hosteas no.

Seguramente algunos nombres de funciones, intervalos o clases CSS podrían ser mejorables.

Al comienzo del desarrollo no pensaba que usar funciones callback era tan importante, pero me fui dando cuenta de que era mejor usarlas al final del desarrollo, debería haberlas usado más, en cieloColor() uso una función tipo callback y además uso dispersión para convertir un objeto no array a array.

Creo que añadir audio, una pequeña historia, un personaje y un temporizador es básico para la inmersión de una persona en el juego, creo que si el juego fuese un poco más difícil, por ejemplo asignando menos tiempo o más bolas, este sería más adictivo.