# DOCUMENTACIÓN
# TFC
# SET DE JUEGOS RETRO

**Autor:** Jonatan Rosario Matos

**Tutor/a:** Raquel Angulo Castaño

**Ciclo:** Desarrollo de Aplicaciones Web

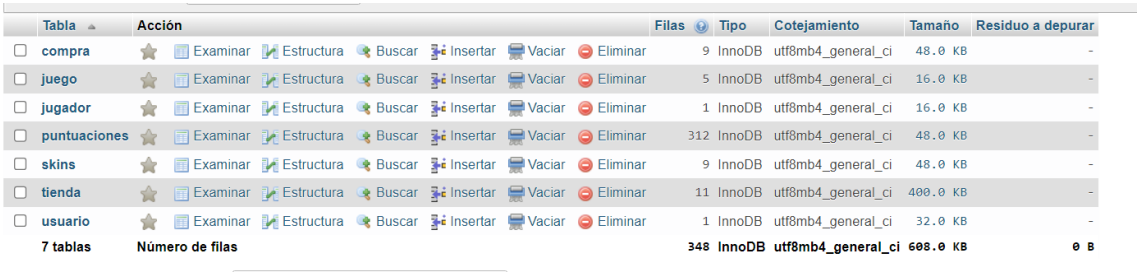
**Centro:** I.E.S Enrique Tierno Galván

Fecha: 22/04/2023

# Tabla de contenido

# BASE DE DATOS

El conjunto de la base de datos está estructurado de la siguiente forma:



| Tabla | Acción | Filas | Tipo | Cotejamiento | Tamaño | Residuo a depurar |
|---|---|---|---|---|---|---|
| compra | Examinar Estructura Buscar Insertar Vaciar Eliminar | 9 | InnoDB | utf8mb4_general_ci | 48.0 KB | - |
| juego | Examinar Estructura Buscar Insertar Vaciar Eliminar | 5 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| jugador | Examinar Estructura Buscar Insertar Vaciar Eliminar | 1 | InnoDB | utf8mb4_general_ci | 16.0 KB | - |
| puntuaciones | Examinar Estructura Buscar Insertar Vaciar Eliminar | 312 | InnoDB | utf8mb4_general_ci | 48.0 KB | - |
| skins | Examinar Estructura Buscar Insertar Vaciar Eliminar | 9 | InnoDB | utf8mb4_general_ci | 48.0 KB | - |
| tienda | Examinar Estructura Buscar Insertar Vaciar Eliminar | 11 | InnoDB | utf8mb4_general_ci | 400.0 KB | - |
| usuario | Examinar Estructura Buscar Insertar Vaciar Eliminar | 1 | InnoDB | utf8mb4_general_ci | 32.0 KB | - |
| 7 tablas | Número de filas | 348 | InnoDB | utf8mb4_general_ci | 608.0 KB | 0 B |

Un conjunto de 7 tablas las cuales será utilizadas para la gestión de los usuarios/jugadores, el manejo de variables internas dentro de la sesión del usuario, como la compra de elementos, búsqueda de otros jugadores o consulta de su perfil o ranking de los juegos, además de poder jugar.

Profundizando en las tablas.

## Tabla Jugador.

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idJugador | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | nombre | varchar(50) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 3 | nivel | int(11) | | | No | 0 | | | Cambiar Eliminar Más |
| 4 | partidasTotales | int(11) | | | No | 0 | | | Cambiar Eliminar Más |
| 5 | mediaPuntos | float | | | No | 0 | | | Cambiar Eliminar Más |
| 6 | maximaPuntuacion | float | | | No | 0 | | | Cambiar Eliminar Más |
| 7 | monedas | int(11) | | | No | 0 | | | Cambiar Eliminar Más |

La tabla jugador tiene de **PK IdJugador** la cual es un número entero que se autoincrementa. Y tendrá diversas relaciones con las demás tablas.

Viene seguido de varios datos que se querrán guardar del propio usuario a la hora de registrarse y ser manejados a la hora de hacer login y de moverse dentro de la sesión.

Estos son:

**nivel**: Un entero que representa el nivel del jugador y tiene un valor predeterminado en 0.

**partidasTotales**: Un entero que contabiliza el total de partidas que tiene un jugador y tiene de valor predefinido en 0.

**mediaPuntos**: Es un float que guardará la media de puntos que tiene el jugador de todas las partidas que ha realizado. Predeterminado en 0.

**maximaPuntuacion**: Entero que guardará la puntuación más alta que haya realizado este jugador de cualquier juego. Predeterminado en 0.

**monedas**: Entero que conserva las monedas que consigue el jugador a partir de jugar. Predeterminado en 0.

## Tabla usuario.

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idUsuario | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | nombre | varchar(50) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 3 | apodo | varchar(50) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 4 | pass | varchar(50) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 5 | apellidos | varchar(100) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 6 | edad | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 7 | idJugador | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |

Esta tabla será meramente usada para guardar información acerca del usuario.

**idUsuario**: Entero que se autoincrementa y es la clave primaria que identificará de forma única a cada usuario de la base de datos.

**nombre:** Cadena de caracteres que almacena el nombre del usuario.

**apodo**: Cadena de caracteres que guarda el apodo del usuario al igual que en la tabla de jugador y este tendrá que ser único a la hora de registrarse.

**pass:** Cadena de caracteres que almacena la contraseña del usuario. Esta esta regida por una serie de restricciones que hará que la contraseña tenga un mínimo de seguridad. Aparte de ser guardad en la base de datos como md5.

**apellidos:** Cadena de caracteres que almacena los apellidos del usuario.

**edad:** Entero que registra la edad del usuario.

**idJugador:** Entero que tendrá una relación con la tabla "jugador" ya que se comportará como una clave foránea.

## Tabla juego.

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idJuego | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | nombre | varchar(50) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |

Dentro de esta tabla solo se guardará un identificador único y nombre de los juegos.

**idJuego**: Entero que autoincrementa y que es clave primaria que identifica de forma única a cada juego.

**nombre**: Cadena de caracteres que almacena el nombre del juego.

## Tabla puntuaciones.

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idPuntuacion | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | idJugador | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 3 | idJuego | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 4 | puntuacion | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |

Esta tabla guardará las puntuaciones de los jugadores después de haber realizado una partida.

**idPuntuacion**: Entero que autoincrementa y que es clave primaria identificacando de forma única a cada puntuación.

**idJugador**: Entero que establece una relación con la tabla "jugador" mediante una clave foránea.

**idJuego**: Entero que establece una relación con la tabla "juego" mediante una clave foránea.

**puntuacion**: Entero que registra la puntuación obtenida por el jugador en un juego específico.

## Tabla Tienda

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idProducto | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | nombre | varchar(50) | utf8mb4_general_ci | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 3 | imagen | longblob | | | No | *Ninguna* | | | Cambiar Eliminar Más |
| 4 | precio | int(11) | | | No | 0 | | | Cambiar Eliminar Más |

Esta tabla se encargará de guardar los elementos, skins de los juegos, que el jugador después de haber jugado unas partidas, podrá comprar para personalizar su experiencia en partida.

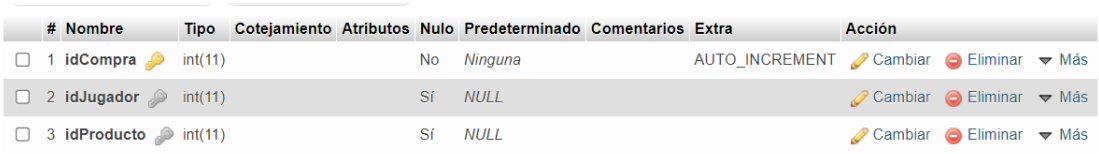
**idProducto**: Entero que autoincrementa y que es clave primaria que identifica de forma única a cada producto en la tienda.

**nombre**: Cadena de caracteres que almacena el nombre del producto.

**imagen**: Datos binarios de una imagen del producto para mostrar en tienda almacenada en formato LONGBLOB.

**precio**: Entero que representa el precio del producto. Tiene un valor predeterminado de 0.

## Tabla compra



| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idCompra | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | idJugador | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 3 | idProducto | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |

Esta se encargará de guardar todos las comprar realizadas por el jugador así como de referencia de que posesiones tiene.

**idCompra**: Entero que autoincremeta y que es clave primaria que identifica de forma única a cada compra realizada por el jugador.

**idJugador**: Entero que establece una relación con la tabla "jugador" mediante una clave foránea.

**idProducto**: Entero que establece una relación con la tabla "tienda" mediante una clave foránea.

## Tabla skin

| # | Nombre | Tipo | Cotejamiento | Atributos | Nulo | Predeterminado | Comentarios | Extra | Acción |
|---|---|---|---|---|---|---|---|---|---|
| 1 | idSkin | int(11) | | | No | *Ninguna* | | AUTO_INCREMENT | Cambiar Eliminar Más |
| 2 | idJugador | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 3 | idProducto | int(11) | | | Sí | *NULL* | | | Cambiar Eliminar Más |
| 4 | skinSeleccionada | tinyint(1) | | | No | 0 | | | Cambiar Eliminar Más |

Esta tabla guardará con un TINYINT como booleano para saber que skin de las que tiene el jugador ha seleccionado.

**idSkins**: Entero que autoincrementa y que es clave primaria que identifica de forma única a cada skin.

**idJugador**: Entero que establece una relación con la tabla "jugador" mediante una clave foránea, representando el jugador al que pertenece la skin.

**idProducto**: Entero que establece una relación con la tabla "juego" mediante una clave foránea, representando el juego al que pertenece la skin.

**skinsSeleccionada**: Valor booleano que indica si la skin ha sido seleccionada por el jugador. Utiliza el tipo de dato TINYINT para representar valores de 0 (no seleccionada) o 1 (seleccionada).

# INICIO, REGISTRO Y LOGIN.

A partir de este punto todo lo que se ha realizado ha sido programado en **PHP**, **JS** y estilizado con **Bootstrap** y **HTML** con **CSS**.

## Inicio

Empezando por la página de inicio.

Es una página con un menú simple, un título principal y dos botones que dan la posibilidad de ir al login o al registro. Apoyada con un script de js que nos llevará a dichas páginas y con una página de estilos css aparte de estar maquetada en Bootstrap.

## Registro

Si nos dirigimos al registros. Habrá una un formulario de para poder registrar un usuario con los campos que se ofrece en la base de datos.

Este formulario tendrá un control tanto de que se completade manera obligatoria todos los campos, como de que estén bien ingresados.

La gestión del envío de los datos será por PHP por método POST, mientras que la gestión de el correcto completado de los campos será por JS.

```php
if (isset($_POST["nombre"]) and isset($_POST["apellidos"]) and isset($_POST["edad"]) and isset($_POST["apodo"]) and isset($_POST["password"])) {
    $nombre = filtrado($_POST["nombre"]);
    $apellidos = filtrado($_POST["apellidos"]);
    $edad = filtrado($_POST["edad"]);
    $apodo = filtrado($_POST["apodo"]);
    $password = md5(filtrado($_POST["password"]));
}
```

Se comprobará que estos existen y además estarán filtrados para no sufrir ninguna inyección de código con ayuda de esta función:

```php
<?php
    function filtrado($texto){
        $texto = htmlspecialchars($texto);
        $texto = stripslashes($texto);
        $texto = trim($texto);
        return $texto;
    }
?>
```

En el JS que tiene importado registro.php hay una función comprobar que revisará cómo se han completado los campos.

Al presionar en el botón de envío, este JS se iniciará con ayuda de un evento haciendo lo siguiente:

La función ‘comprobar()’, se ejecuta cuando se hace clic en el botón de envío. Obtiene los valores de los campos del formulario (nombre, apellidos, edad, apodo y contraseña) y realiza varias validaciones.

Verifica que todos los campos estén completos. Si algún campo está vacío, se muestra un mensaje de error específico de dicho campo.

Verifica la longitud de la contraseña. Si es menor a 7 caracteres, se muestra un mensaje de error.

También comprobará que la contraseña contenga tanto letras mayúsculas como minúsculas. Si no cumple esta regla, se muestra un mensaje de error.

Verifica que la edad sea un número y que sea mayor o igual a 0. Si no cumple estas condiciones, se muestra un mensaje de error.

Si no se encontraron errores, se llama a la función ‘registro()’.

‘registro()’ es una función que simplemente hace un submit del formulario.

Una vez hecho esto se comprobará junto a la base de datos si el apodo ingresado ya existe, en caso de existir se tendrá que completar el formulario de nuevo. En caso de estar bien se insertará los datos en tabla usuario y se insertará también un jugador con todos los datos iniciados en 0 y con el apodo con el que se ha registrado y lo mandará directamente a sesion.php.

### Login

Dentro de la página login.php se mostrará un formulario con dos campos que serán para poner el apodo y la contraseña.

Estos estarán nuevamente controlados por JS para saber si se deja algún campo vacío y luego a través de método POST y una conexión a la base de datos si el apodo escrito y contraseña pertenece al mismo usuario.

En caso de ser así este será llevado a la sesión.php.

## SESIÓN

En esta página se muestra una barra de navegación con enlaces a otras secciones que son el perfil, ranking, tienda, jugar y búsqueda de usuarios.

Se muestra un mensaje de bienvenida al usuario y se realiza consultas a la base de datos para obtener información relacionada con el usuario autenticado por una sesión iniciada.

Funcionalidades que se realizan dentro de la sesión:

### Consulta del perfil.

Dentro de perfil.php podremos ver tanto nuestras estadísticas como jugador como elegir dentro de los elementos comprados en la tienda cuales de ellos queremos usar a la hora de jugar en cada juego.

Estos elementos se despliegan después de pulsar un botón que pone skins que muestra el repertorio de skins que hay pero solo con la posibilidad de habilitar una para cada juego de las que están disponibles según el juego.

Para dejar grabada que skins usar recurriendo a la base de datos, a las tablas jugador para obtener el idJugador a través del apodo que esta guardado en la sesión.

Posteriormente de la tabla compra recoger con ayuda del idJugador los productos que tiene dicho jugador para luego con ayuda de los radio inputs marcados por el usuario (de las skins que ha elegido) dejar marcado como 1 en la tabla skins dicho elemento de un juego en concreto y los demás respecto a cada juego en 0.

Esto nos ayudará posteriormente a saber que tipo de skin ha cogido en cada juego el jugador.

## Navegación de la tienda.

Esta es otra posibilidad que facilita sesión.php, permite acceder a la tienda y poder ver el catálogo y comprar los productos.

En la sección PHP, se inicia una sesión y se establecen las variables de conexión a la base de datos. Luego se obtienen los detalles del jugador actualmente conectado a través de la variable de sesión.

Hay una serie de conexiones SQL para obtener información relevante de la base de datos, como la lista de productos disponibles en la tienda, la cantidad de monedas del jugador y su identificador único.

Al realizar el envío POST se maneja el envío del formulario. Si se ha seleccionado un producto para comprar, se ejecuta la función 'gestionCompra()' que esta en otro archivo que lo contiene llamado 'gestionCompra.php'. Esta función realiza la compra del producto seleccionado y actualiza la base de datos, teniendo en cuenta si el producto ya ha sido comprado mostrándote un mensaje de que ya no lo puedes comprar para que no haya el duplicado de productos por jugador, o si no tienes el crédito suficiente para realizar la compra o si has podido realizarla.

En la parte del formulario, se muestran las tarjetas de los productos disponibles en la tienda. Cada tarjeta muestra una imagen, el nombre del juego, una descripción, el precio y un botón de compra que permite realizar el trámite del producto.

También hay que tener en cuenta que los valores de value en los botones corresponden al identificador único de cada producto que serán cruciales para la gestión del mismo dentro del juego.

## Ranking y Search

En el caso de ranking y search, al pulsar sobre la opción de ranking se hace una llamada a una función que se encuentra en el archivo busquedas.php que se encarga según que se haya solicitado si ver los rankings o buscar un jugador en concreto, mostrar las top 10 mejores puntuaciones de cada juego mostrando el nombre del usuario, la puntuación máxima de ese jugador en ese juego en caso de que esté en las diez mejores y si se ha seleccionado buscar un usuario lo que hará es mostrar el nombre del mismo, el nivel, el numero de partida, la media de puntos y la máxima puntuación que tiene.

### LogOut

Hay un botón submit de logOut que lo que permite es que al ser pulsado la sesión se cierra y te saca de la sesión, mandándote de vuelta a la pagina de inicio, index.php.

### Jugar

Esta página muestra diferentes juegos y permite a los jugadores seleccionar y jugar a cada uno de ellos dirigiendo directamente al juego. La información sobre los juegos seleccionados por el jugador se almacena en una base de datos y se recupera cada vez que se carga la página.

Aquí también se realiza un session_start(), que inicializa una sesión permitiendo recuperar el apodo del jugador que aprovecharemos para usar a la hora de hacer llamadas a la base de datos para ir rascando información de la misma.

Establece una conexión con una base de datos utilizando MySQLI con los datos necesarios: localhost, dbname, password y database.

Se extrae el idJugador de la base de datos basado en el valor almacenado en la variable de sesión.

Se recupera las skins seleccionadas para el jugador desde la base de datos basado con el idJugador obtenido y porque anteriormente hicimos una selección de las mismas desde el perfil del jugador donde ya se determina que skins son las que corresponde a la hora de ser usadas en el juego al que se vaya a dirigir el jugador.

Asigna diferentes variables para cada una de las skins que se han recuperado en la base de datos desde la tabla skins y con esto podremos determinar dentro de una variable un valor que distinguirá la skin concreta que quiere usar el jugador.

Se muestra cinco tarjetas, una por cada juego, representadas con un título y una imagen correspondiente a dicho juego: Tetris, Pacman, Dinosaur, Snake y FlappyBird.

Dentro de estas hay en embebido código PHP que genera unas URLs para cada juego, agregando el apodo (apodo del jugador) y otro parámetro que va asignado a la skin que quiere el jugador que se le muestre a la hora de iniciar el juego.

## JUEGOS

El set de juegos está dividido en una cantidad de 5 juegos que son:

- Tetris.
- Pacman.
- Dinosaur Game.
- Snake.
- FlappyBird.

Estos conforman un conjunto de juegos retro diseñados en 2D con ayuda de las escenas que aporta canvas con ayuda de HTML y JS.

### TETRIS

Dentro del tetris tenemos dos ficheros, uno es tetris.html y el otro tetris.js, en este segundo se desarrollará la lógica que conlleva el juego.

#### Tetris.html

Principalmente se muestra un título 'Proxima figura' para indicar la próxima figura que aparecerá en el juego.

Hay un elemento canvas con el id 'siguiente' que se utiliza para dibujar la representación visual de la próxima figura de tetris.

Hay otro elemento canvas con el id 'tetris' que se utiliza para dibujar y jugar el juego del Tetris propiamente dicho. Tiene un tamaño de 200x400 píxeles. Básicamente es el escenario donde se desarolla el juego.

Hay varias secciones div que muestran información relacionada con el juego:

'Puntuacion:' y un elemento  con el id "puntuacion" que muestra la puntuación actual del jugador.

'Linea:' y un elemento  con el id 'linea' que muestra la cantidad de líneas completadas por el jugador.

'Nivel:' y un elemento  con el id 'nivel' que muestra el nivel actual del jugador.

Este tetris.html se enlaza con un tetris.js por un script. Este archivo contiene el código JavaScript que controla el funcionamiento del juego del Tetris.

En las primeras líneas, se utiliza URLSearchParams para obtener el parámetro apodo de la URL. Esto permite pasar el apodo como un parámetro en la URL y luego usarlo en el juego a la hora de cuando el usuario pierda, porque será en ese momento donde se inserta en la tabla puntuaciones la puntuación del jugador, se sume más uno en el total de partida y ver si ha ganado monedas.

Se guarda en variables las dos escenas creadas en el tetris.html para su gestión dentro del JS.

Se obtienen los contextos de los canvas, contexto_canvas y contexto_canvasSiguientes, respectivamente. Estos contextos se utilizan para dibujar en los canvas.

Se crean dos matrices utilizando la función matriz. tabla es una matriz de tamaño 10x20 que representa el tablero de juego, y tablaPieza es una matriz de tamaño 19x19 que representa el área de juego de la pieza actual.

Se define un array de colores, donde cada elemento representa un color para cada tipo de pieza. El primer elemento es null porque se utiliza para representar las partes vacías de la matriz.

Se crea un objeto llamado jugador que almacenará la información del jugador, como la posición, la matriz de la pieza actual, la puntuación, el nivel, las líneas completadas y la siguiente pieza.

Se realiza un escalado en los contextos de los canvas para ajustar el tamaño de las piezas. Cada pieza tiene un tamaño de 20x20 píxeles.

Se inicializan algunas variables para controlar el tiempo y la velocidad de caída de la pieza.

Se define una variable actualizacion que almacenará el identificador del bucle principal del juego. Es decir, donde el juego se va estar refrescando constantemente.

Se define una variable gameOver inicializada en false para controlar el estado del juego.

Se define una variable ultimoTiempo que almacenará el tiempo del último frame para controlar la actualización del juego.

La función actualizar(tiempo) es el bucle donde el juego se ejecuta en cada frame. Recibe un parámetro tiempo que representa el tiempo actual en milisegundos.

Se calcula la diferencia de tiempo tiempo_aux entre el tiempo actual y el último tiempo registrado. Luego, se actualiza la variable ultimoTiempo con el tiempo actual.

Se incrementa el contador de caída contador_caida con el valor de tiempo_aux.

Si el contador de caída es mayor que el intervalo de caída (intervalo_caida), se llama a la función caidaPieza() que desplaza la pieza hacia abajo.

Luego la función dibujarMapa() actualiza redibujando el tablero de juego y la pieza actual en el canvas.

Finalmente, se realiza una llamada recursiva a la función actualizar() utilizando requestAnimationFrame() para mantener el bucle de actualización del juego, refrescando las modificaciones del mismo.

La función dibujarMatriz(matriz, posicioPieza) se encarga de dibujar una matriz en el canvas de la pieza de tetris que va a caer. Recibe la matriz a dibujar y la posición de la pieza en el tablero. Se utiliza un bucle forEach anidado para recorrer la matriz y, si el valor de una celda es diferente de 0, se dibuja un rectángulo en el canvas con el color correspondiente de la pieza.

La función dibujarMapa() se encarga de dibujar el tablero de juego y la pieza actual en el canvas. Primero, se rellena el fondo del canvas con el

color negro. Luego, se llama a dibujarMatriz() dos veces, una vez para dibujar el tablero y otra vez para dibujar la pieza actual.

La función dibujarSiguienteMatriz(matriz, posicionPieza) es similar a dibujarMatriz(), aunque esta dibuja la vista previa de la siguiente pieza en el canvas siguienteCanvas, de modo que le jugador puede anticipar un poco de tiempo su próxima jugada.

La función matriz(anchura, altura) crea y devuelve una matriz de tamaño anchura x altura con todas las celdas inicializadas en 0.

La función caidaPieza() se encarga de desplazar la pieza actual hacia abajo, es decir sobre el eje Y, en vertical. Verifica si hay colisiones con el tablero utilizando la función colisiones(). Si hay colisiones, se restaura la posición anterior de la pieza, se fusiona la pieza con el tablero utilizando la función merge(), se reinicia el jugador y se realizan algunas operaciones adicionales como la eliminación de líneas completadas y la actualización de la puntuación.

Hay un evento que habilita ciertas teclas permitiendo al jugador mover la pieza hacia abajo con la tecla de flecha hacia abajo, moverla hacia la derecha con la tecla de flecha hacia la derecha, moverla hacia la izquierda con la tecla de flecha hacia la izquierda y rotarla en sentido horario con la tecla "z". Cada acción llama a la función correspondiente para realizar el movimiento o la rotación de la pieza.

La función colisiones(tabla, jugador) verifica si hay colisiones entre la pieza actual y el tablero. Recorre la matriz de la pieza y, si encuentra una celda con un valor distinto de 0, verifica si hay una colisión en la misma posición del tablero. Si encuentra una colisión, devuelve true, indicando que hay colisiones para parar la pieza.

Las funciones colisionLateral(direccion) y merge(tabla, jugador) en conjunto se encargan de manejar las colisiones laterales. La primera desplaza la posición en el eje X del jugador según la dirección especificada, +1 para derecha, -1 para izquierda, y verifica si hay colisiones.

La función playerReset() se encarga de restablecer el estado del jugador y del juego en caso de que surja una colisión. Se define un array piezas que contiene las diferentes formas de las piezas del Tetris. Luego, se genera un número aleatorio entre 0 y 6 para seleccionar una pieza al azar del array que será la que vaya a salir en el canvas. El intervalo de caída se actualiza en función del nivel del jugador, cuanto más nivel, más rápido.

Después, se verifica si el jugador ya tiene una siguiente pieza disponible. Si no la tiene, se crea una nueva pieza para el jugador y se asigna como siguiente pieza. Si ya tiene una siguiente pieza, se asigna la siguiente pieza al jugador.

A continuación, se establece la posición inicial de la pieza en el tablero. La posición X se calcula de manera que la pieza aparezca en el centro y de manera horizontal del tablero. La posición Y se establece en 0, lo que significa que la pieza aparece en la parte superior del tablero.

Si se detecta una colisión entre la pieza y el tablero se reinicia el tablero, la puntuación, el nivel y el número de líneas completadas. Luego, se actualiza la puntuación en la interfaz del juego.

Además, se envía una petición POST a un archivo PHP para guardar la puntuación del jugador en una base de datos.

La función rotacionJugador() se encarga de rotar la pieza actual en el sentido de las agujas del reloj. Primero, se guarda la posición X actual del jugador. Luego, se utiliza una técnica de rotación de matriz para rotar la matriz de la pieza. Después de la rotación, se verifica si hay colisiones entre la pieza rotada y el tablero. Si se detecta una colisión, se intenta mover la pieza hacia la izquierda y hacia la derecha para encontrar una posición sin colisiones. Si no se encuentra una posición sin colisiones, se restaura la posición original de la pieza.

La función rotacion(matriz) se utiliza para rotar una matriz en sentido horario. Se utiliza un par de bucles for para intercambiar los elementos de la matriz en función de su posición. Después de la rotación, se invierte el orden de las filas de la matriz.

La función crearPieza(tipo) se utiliza para crear y devolver una matriz que representa una pieza específica del Tetris. Recibe un parámetro tipo que especifica el tipo de pieza a crear. Dependiendo del tipo de pieza, se devuelve una matriz con los valores adecuados para representar esa forma de pieza en particular.

La función eliminacionLineas() se encarga de eliminar las líneas completadas en el tablero. Recorre el tablero desde la última fila hacia arriba y verifica si cada fila está completamente llena. Si encuentra una fila completa, la elimina del tablero y la inserta en la parte superior. Después de eliminar una línea, se incrementa la puntuación del jugador y se actualiza el nivel si se cumple una condición específica.

La función actualizarPuntuacion() se utiliza para actualizar la puntuación, el nivel y el número de líneas en la interfaz del juego. Obtiene los elementos HTML correspondientes a la puntuación, el nivel y el número de líneas y actualiza su contenido con los valores actuales del jugado

## PACMAN

Dentro del pacman tenemos los ficheros pacman.html, fantasmas.js, comecos.js y juego.js

### pacman.html

Hay un elemento canva> con el id "canvas" que se utilizará para renderizar el juego Pacman. El atributo width y height se establecen en 420 y 500 píxeles respectivamente para definir el tamaño del lienzo.

Dentro del cuerpo se encuentra un elemento  con el atributo style establecido en "display: none". Este elemento contiene imágenes ocultas que se utilizan en el juego, que son la animación de Pacman y las imágenes de los fantasmas.

Y luego hay una biblioteca jQUERY para la comunicación entre js y php para enviar los puntos de cada partida, etc. Exactamente como en el tetris.

El constructor constructor(x, y, anchura, altura, velocidad) se inicializa las propiedades del objeto Comecocos con los valores definidos. Los parámetros x e y representan la posición inicial en el lienzo.

La anchura y altura definen las dimensiones del personaje.

La velocidad determina la velocidad de movimiento del comecocos.

La propiedad direccion guarda la dirección actual del movimiento, proximaDirec guarda la siguiente dirección a tomar, frame y maxframe se utilizan para la animación del personaje.

También se establece un intervalo de tiempo que llama a la función cambioAnimacion() cada 100 milisegundos.

El método procesoMovimiento() controla el movimiento del personaje. Este llama al método cambioDireccion() para verificar si se debe cambiar de dirección.

Después se llama al método moverHaciaDelante() para mover al personaje en la dirección actual. Si hay colisiones con obstáculos en el mapa, llama al método moverHaciaAtras() para detener el movimiento.

El método comer() verifica si el personaje está en una posición donde puede comer un punto del mapa, actualizando el mapa y aumentando la puntuación.

Los métodos moverHaciaDelante() y moverHaciaAtras() se encargan de mover al personaje en la dirección actual u opuesta, dependiendo de la propiedad direccion.

El método colisones() verifica si hay colisiones con obstáculos en el mapa en la posición actual y en las posiciones adyacentes. Si hay colisiones, devuelve true; si no, verifica si el personaje atraviesa un túnel en el mapa y actualiza su posición si es necesario antes de devolver false.

El método colisionesFantasmas() verifica si hay colisiones con algún fantasma en el juego. Recorre la lista de fantasmas y compara las posiciones con la del personaje. Si hay una colisión, devuelve true; de lo contrario, devuelve false.

El método cambioDireccion() verifica si hay una próxima dirección establecida en proximaDirec diferente de la dirección actual. Si es así, se guarda la dirección actual en direccionTemporal, se actualiza la dirección con proximaDirec y se intenta mover al personaje hacia adelante. Si hay colisiones, se mueve hacia atrás y se restablece la dirección original. En caso contrario, también se mueve hacia atrás.

El método cambioAnimacion() se utiliza para cambiar los frames de la animación del personaje. Actualiza el valor de frame y devuelve el valor actual de frame para determinar qué frame se debe mostrar, para ver el movimiento del personaje del jugadore.

El método dibujar() se encarga de dibujar al personaje en el lienzo del juego. Utiliza el contexto del canvas para realizar una serie de transformaciones que rota el personaje según su dirección y luego dibuja el sprite correspondiente en la posición x, y con las dimensiones anchura y altura.

Los métodos getPosicionX(), getPosicionY(), getEstadoX() y getEstadoY() se utilizan para obtener las posiciones actuales y los estados del personaje en relación con el mapa del juego.

### fantasmas.js

La clase Fantasmas tiene un constructor que toma varios parámetros, como las coordenadas (x e y) del fantasma, su tamaño (anchura y altura), su velocidad, las coordenadas y dimensiones de su imagen, y un rango.

Además tiene la dirección que lleva un fantasma, obejtivoAleatorioIndex que coge aleatoriamente un objetivo para el fantasma, y objetivo que puede ser pacman u otro objetivo que tenga definido.

El setInterval cambioAleatorioDireccion() cada 3 segundos, otro que llama a ese mismo método cada 5 segundos para el fantasma naranja, y otro que cambia el objetivo del fantasma rojo cada segundo.

La función cambioAleatorioDireccion() se utiliza para seleccionar aleatoriamente un nuevo objetivo para el fantasma.

La función actitudAzul() selecciona aleatoriamente si el fantasma azul debe perseguir al personaje principal (comecocos) o si debe tener un objetivo aleatorio de objetivoFantasmas.

La función procesoMovimiento() controla el movimiento del fantasma. Determina el objetivo del fantasma en función de la posición de pacman y el rango establecido

Las funciones moverHaciaDelante() y moverHaciaAtras() se encargan de mover al fantasma en la dirección actual.

La función colisones() comprueba si el fantasma ha colisionado con obstáculos en el mapa. Si hay una colisión, se establece la variable colisionado en true.

La función enRangoPacman() verifica si el personaje principal está dentro del rango establecido para el fantasma.

La función cambioDireccion() determina la dirección en la que el fantasma debe moverse para alcanzar su objetivo. Utiliza el algoritmo de búsqueda de amplitud (Dijkstra) para encontrar el camino más corto hacia el objetivo.

La función dibujar() se encarga de dibujar el fantasma en el lienzo de juego, utilizando las coordenadas y dimensiones definidas.

Las funciones getPosicionX(), getPosicionY() getEstadoX() y getEstadoY() se utilizan para obtener las coordenadas actuales del fantasma en relación con el mapa del juego.

Las funciones actualizarFantasmas() y dibujarFantasmas(), que se utilizan para actualizar y dibujar todos los fantasmas presentes en el juego, respectivamente.

juego.js

El código comienza obteniendo los parámetros de la URL, específicamente el apodo y el número de juego para manejar la skin y posteriormente los datos de la puntuación en la base de datos.

Luego, se inicializan algunas variables y se establecen constantes para el juego, como los códigos para las direcciones de movimiento y los colores utilizados.

Se define la estructura del mapa del juego, que está representado por una matriz bidimensional.

Definimos las ubicaciones y objetivos de los fantasmas en el juego.

Se define un evento para detectar la pulsación de la tecla "Espacio", que se utiliza para iniciar el juego.

La función loop() es el bucle principal del juego, que se ejecuta repetidamente para actualizar y dibujar el juego.

Dentro del bucle, se realizan varias tareas, como dibujar el lienzo, actualizar el movimiento de los personajes, gestionar las colisiones y verificar las condiciones de victoria o derrota.

Si el juego ha terminado, se muestra el mensaje de "GAME OVER" o "WINNER" según corresponda y se guarda los datos de la partida.

## DINOSAUR GAME

Dentro del dinosaur game tenemos los ficheros dinosaur.html, terreno.js, puntuacion.js, jugador.js, game.js, cactusAjustes.js y cactus.js.

### Game.js

Hay estilos para eliminar el margen y el relleno predeterminado, centrar verticalmente el contenido en la ventana del navegador y ocultar la selección de texto y los efectos táctiles.

Se incluyen dos archivos JavaScript como jQuery.

El segundo es el archivo game.js. Este archivo contiene el código necesario para ejecutar el juego.

Tambien hay un canvas con el atributo "id" establecido como "mapa". Este elemento proporciona un lienzo en el que se puede dibujar y animar gráficos utilizando JavaScript.

### Terreno.js

El constructor de la clase recibe varios parámetros, incluyendo el contexto del lienzo de dibujo, la anchura y altura del terreno, la velocidad de desplazamiento, y la escala.

La propiedad  contexto hace referencia al contexto de dibujo del lienzo en el que se va a dibujar el terreno.

La propiedad canvas hace referencia al elemento canvas del contexto de dibujo.

Las propiedades anchura y altura representan las dimensiones del terreno.

La propiedad velocidad indica la velocidad de desplazamiento del terreno.

La propiedad escala representa la escala del terreno.

Las propiedades x e y indican la posición inicial del terreno en el lienzo.

La propiedad terrenoImagen es un objeto de la clase Image que se utiliza para cargar y representar la imagen del terreno. La ruta de la imagen se establece en el constructor.

El método dibujar se utiliza para dibujar el terreno en el lienzo. Se utiliza el método drawImage el contexto de dibujo para renderizar la imagen del terreno en las coordenadas (x, y) con la anchura y altura especificadas. Luego, se dibuja una segunda imagen desplazada en la posición (x + anchura, y), lo que crea la ilusión de un terreno infinito. Si la posición x es menor que -anchura, se reinicia la posición x a cero.

El método actualizar se utiliza para actualizar la posición del terreno en función de la velocidad de desplazamiento y el tiempo transcurrido. Recibe los parámetros velocidad y frame, que se utilizan para calcular el desplazamiento en función de la velocidad y la escala del terreno.

El método reinicio reinicia la posición x del terreno a cero.

### cactus.js

El constructor de la clase recibe varios parámetros, incluyendo el contexto del lienzo de dibujo, las coordenadas x e y del cactus, su anchura y altura, y la imagen que se utilizará para representarlo.

La propiedad contexto hace referencia al contexto de dibujo del lienzo en el que se va a dibujar el cactus.

Las propiedades x y y indican la posición del cactus en el lienzo.

Las propiedades anchura y altura representan las dimensiones del cactus.

La propiedad imagen es un objeto de la clase Image que se utiliza para cargar y representar la imagen del cactus. La imagen se pasa como argumento en el constructor.

El método actualizarse utiliza para actualizar la posición del cactus en función de la velocidad de desplazamiento del juego, la velocidad de desplazamiento del cactus, el tiempo transcurrido (frameDelta) y la escala. La posición x del cactus se actualiza restando la velocidad multiplicada por los factores mencionados.

El método dibujar se utiliza para dibujar el cactus en el lienzo. Se utiliza el método "drawImage" del contexto de dibujo para renderizar la imagen del cactus en las coordenadas (x, y) con la anchura y altura especificadas.

El método colision se utiliza para verificar si hay una colisión entre el cactus y otro objeto de juego, que se pasa como argumento (sprite). Se comprueba si las coordenadas y dimensiones del sprite y el cactus se superponen utilizando un cálculo de colisión básico. Si hay una colisión, se devuelve true; de lo contrario, se devuelve false.

### cactusAjustes.js

La clase define dos propiedades constantes, CACTUS_MAX_INT y CACTUS_MIN_INT, que representan los intervalos máximos y mínimos entre la generación de cactus.

La propiedad siguienteIntervalo se utiliza para almacenar el tiempo restante hasta la próxima generación de cactus.

La propiedad cactus es un array que almacena los objetos de la clase Cactus generados.

El constructor de la clase recibe el contexto del lienzo de dibujo, la imagen del cactus, la escala y la velocidad.

El método modificartiempo se utiliza para generar un nuevo intervalo de tiempo aleatorio para la próxima generación de cactus. Este método elige un número aleatorio entre CACTUS_MIN_INT y CACTUS_MAX_INT y lo almacena en la propiedad siguienteIntervalo.

El método numeroAleatorio genera un número aleatorio entre un valor mínimo y máximo, utilizando la fórmula Math.random() y Math.floor().

El método crearCactu se utiliza para crear un nuevo objeto de la clase Cactus con valores aleatorios de posición y selección de imagen. El cactus creado se añade al array cactus.

El método actualizar se utiliza para actualizar la generación de cactus y la posición de los cactus existentes. Si el tiempo restante hasta la próxima generación es menor o igual a cero, se crea un nuevo cactus llamando a crearCactus() y se actualiza el tiempo restante llamando a modificartiempo(). El método también actualiza la posición de los cactus existentes llamando al método actualizar de cada objeto Cactus. Por último, se filtran los cactus que ya no están visibles en el lienzo.

El método dibujar se utiliza para dibujar todos los cactus existentes en el lienzo, llamando al método "dibujar" de cada objeto Cactus.

El método colision se utiliza para verificar si hay una colisión entre el sprite (objeto de juego) y alguno de los cactus existentes. Se utiliza el método some() del array "cactus" para comprobar si al menos uno de los cactus colisiona con el sprite.

El método reinicio se utiliza para reiniciar la lista de cactus, vaciando el array cactus.

### jugador.js

La clase importa las variables numeroDino y letraDino desde el archivo "./game.js", para manejar la skin.

La clase define una propiedad animaciones como un array para almacenar las imágenes de animación del dinosaurio.

El constructor de la clase recibe el contexto del lienzo de dibujo, la anchura y altura del dinosaurio, los límites de salto máximo y mínimo, y la escala.

Se inicializan las propiedades relacionadas con la imagen del dinosaurio estático. La imagen se carga dependiendo del valor de numeroDino y letraDino. También se inicializa la posición inicial del dinosaurio.

Se inicializa la animación de caminar del dinosaurio. Se cargan las imágenes de caminar del dinosaurio dependiendo del valor de numeroDino y letraDino. Estas imágenes se agregan al array animaciones.

Se inicializan las propiedades relacionadas con el salto del dinosaurio. Se registran eventos para el control del salto mediante teclado y toque. Al presionar la tecla de espacio o tocar la pantalla, se establece la propiedad salto_presionado en true. Al soltar la tecla o levantar el dedo de la pantalla, se establece en false.

El método dibujar se utiliza para dibujar la imagen del dinosaurio en el lienzo.

El métodoactualizar" se utiliza para actualizar la posición y animación del dinosaurio. Dependiendo del estado de salto_progreso y caida, se ajusta la posición vertical del dinosaurio. También se actualiza la animación de caminar.

El método saltar se utiliza para controlar el salto del dinosaurio. Si salto_progreso es true y no se está cayendo, el dinosaurio se mueve hacia arriba hasta alcanzar el límite de salto máximo. Luego, el dinosaurio comienza a caer hasta volver a la posición de inicio. Si caida es true, el dinosaurio sigue cayendo hasta que vuelve a la posición de inicio.

El método correr se utiliza para controlar la animación de caminar del dinosaurio. Cada cierto intervalo de tiempo, se cambia la imagen del dinosaurio entre las dos imágenes de animación almacenadas en el array animaciones.

### puntuaciones.js

La clase define dos propiedades: puntuacion y mayorPuntuacion para almacenar la puntuación actual y la puntuación más alta alcanzada.

El constructor de la clase recibe el contexto del lienzo de dibujo y la escala del juego.

El método actualizar se utiliza para actualizar la puntuación en función del tiempo transcurrido desde el último frame. Incrementa la puntuación en función del valor de frameDelta.

El método reinicio se utiliza para reiniciar la puntuación estableciéndola a cero.

El método setMayorPuntuacion se utiliza para actualizar la puntuación más alta si la puntuación actual supera la puntuación más alta almacenada. La puntuación más alta se redondea hacia abajo y se asigna a la propiedad mayorPuntuacion.

El método dibujar se utiliza para dibujar la puntuación en el lienzo. Utiliza el contexto de dibujo para establecer la fuente y el color del texto. Luego, dibuja la puntuación actual y la puntuación más alta en posiciones específicas en el lienzo. La puntuación se muestra en un formato de 6 dígitos con ceros a la izquierda para mantener una longitud constante.

### juego.js

Se importan todas las clases

Se obtienen parámetros de la URL, como el apodo del jugador, el número y la letra del dinosaurio.

Se crea un lienzo y se obtiene el contexto de dibujo.

Se definen varias constantes y variables relacionadas con las dimensiones del juego, el escalado, la velocidad, las imágenes de los cactus, etc.

Se inicializan objetos para el jugador, el terreno, los cactus y la puntuación.

Se define una función para ajustar el escalado del juego en función del tamaño de la ventana.

Se definen funciones para limpiar el lienzo, mostrar el mensaje de "GAME OVER", reiniciar el juego y mostrar el texto de inicio.

Se define una función para actualizar la velocidad del juego en función del tiempo transcurrido.

Se define una función de actualización principal que se llama recursivamente utilizando requestAnimationFrame. Esta función se encarga de actualizar y dibujar los objetos del juego en cada cuadro. También verifica las colisiones y maneja el reinicio del juego.

Se agregan event listeners para detectar la pulsación de la tecla "Espacio" o el toque en la pantalla para reiniciar el juego.

## SNAKE

Dentro de este juego hay dos ficheros prueba3.html y snake.js

La estructura de prueba3.html es similar a los demás, así que pasaremos al js.

### snake.js

Obtenemos el elemento canvas del documento HTML.

Cogemo los parámetros de la URL utilizando el objeto URLSearchParams.

Se cargan las imágenes de la cabeza, cuerpo, cola y manzana de la serpiente.

Se definen variables para el estado del juego, puntuación y puntuación más alta.

Se define un evento de teclado para cambiar la dirección de la serpiente.

Se definen varias funciones para verificar las condiciones de victoria, dibujar la puntuación y mostrar mensajes en el canvas.

La función gameLoop es la función principal del juego. Se ejecuta continuamente y actualiza el estado del juego en cada iteración.

Dentro de gameLoop, se verifica si la serpiente ha chocado consigo misma o alcanzado los límites del canvas, y se muestra un mensaje de "game over" en caso afirmativo.

Si la serpiente come una manzana, se genera una nueva posición para la manzana, se incrementa la puntuación y se incrementa la longitud de la serpiente.

Si la serpiente ha llenado todo el canvas, se muestra un mensaje de "¡Has ganado!" y se finaliza el juego.

Se dibuja el fondo del canvas en cada iteración.

Se dibujan la cabeza, cuerpo y cola de la serpiente utilizando las imágenes correspondientes.

Se dibuja la manzana en su posición actual.

La función resetGame reinicia el juego y vuelve a la pantalla de inicio.

## FLAPPYBIRD

En este juego contemos de ficheros flappyBird.html, pájaro.js, imagnes.js, flappyBird.js

En este caso el fichero html actúa como en el resto de los juegos explicados anteriormente, así que vayamos a la lógica del los archivos JS.

### imagen.js

El constructor de la clase toma dos parámetros: contexto e imagen. El contexto representa el contexto de dibujo en el que se dibujará la imagen, mientras que imagen es el objeto de la imagen que se va a dibujar. El constructor asigna estos valores a las propiedades correspondientes de la clase.

Propiedades: La clase "Imagen" tiene varias propiedades que definen cómo se dibuja la imagen en el lienzo:

sX, sY, w y h definen las coordenadas y dimensiones de una sección de la imagen (suelo).

x y y representan las coordenadas donde se dibujará la sección de la imagen en el lienzo.

cieloSX, cieloSY, cieloW y cieloH definen las coordenadas y dimensiones de otra sección de la imagen (cielo).

cieloX y cieloY representan las coordenadas donde se dibujará la sección del cielo en el lienzo.

modificacionEstado es una variable que representa el estado del juego y se utiliza para realizar modificaciones adicionales en el dibujo de la imagen.

dx es una variable que controla la velocidad de desplazamiento de la imagen en el lienzo.

Método dibujarSuelo: Este método se utiliza para dibujar la sección del suelo de la imagen en el lienzo. Utiliza el contexto de dibujo para llamar al método drawImage y dibujar la sección del suelo en la posición especificada.

Método dibujarCielo: Este método se utiliza para dibujar la sección del cielo de la imagen en el lienzo. Al igual que el método anterior, utiliza el contexto de dibujo y el método drawImage para dibujar la sección del cielo en la posición especificada.

Método actualizarMovimientoSuelo: Este método se encarga de actualizar el movimiento del suelo en el lienzo. Toma un parámetro llamado "estado" que indica el estado actual del juego. Si el estado es 1 o 0, se actualiza la posición del suelo utilizando la propiedad x y el desplazamiento dx. La expresión (this.x - this.dx) % (this.w/2) garantiza que el suelo se desplace de manera continua repitiendo una sección de la imagen. Sin embargo, la línea comentada this.cieloX = (this.cieloX - this.dx)%(this.cieloW/2); no está activa y no se utiliza en la lógica del movimiento del suelo.

### pajaro.js

Importaciones: de las clase Imagen y variable numero desde otros archivos. Estas importaciones permiten utilizar esas clases y variables en el código actual.

Constructor: El constructor de la clase Bird toma dos parámetros: contexto e imagen. El contexto representa el contexto de dibujo en el que se dibujará el pájaro, mientras que imagen es el objeto de la imagen del pájaro. El constructor asigna estos valores a las propiedades correspondientes de la clase.

Propiedades: La clase Bird tiene varias propiedades que definen el aspecto y el comportamiento del pájaro en el juego:

animaciones es un arreglo que contiene diferentes animaciones del pájaro en forma de objetos con las propiedades sX y sY, que representan las coordenadas de la imagen donde se encuentra cada animación.

x y y representan las coordenadas del pájaro en el lienzo.

w y h definen el ancho y la altura del pájaro.

frame es un contador que indica el índice actual de la animación en el arreglo animaciones.

periodo define la velocidad de cambio de animación.

gravedad es la fuerza de gravedad que afecta al pájaro.

salto es la fuerza con la que el pájaro salta cuando se hace clic en el lienzo.

velocidad representa la velocidad actual del pájaro.

rotacion define la rotación actual del pájaro.

radio es el radio del pájaro utilizado para detectar colisiones.

imagenSuelo es una imagen adicional utilizada para el suelo en el juego.

suelo es una instancia de la clase Imagen que representa el suelo en el juego.

modificacionEstado es una variable que representa el estado del juego y se utiliza para realizar modificaciones adicionales en la lógica del pájaro.

sonido es un objeto de audio utilizado para reproducir un sonido cuando el pájaro muere.

Método dibujar: Este método se encarga de dibujar el pájaro en el lienzo. Primero, guarda el estado actual del contexto de dibujo. Luego, utiliza el método drawImage del contexto para dibujar la animación actual del pájaro en la posición especificada. Finalmente, restaura el estado anterior del contexto.

Método aleteo: Este método se llama cuando se hace clic en el lienzo y provoca que el pájaro vuele al cambiar su velocidad.

Método actualizar: Este método se encarga de actualizar la posición y la animación del pájaro en función del estado actual del juego. Toma tres parámetros: estado (representa el estado del juego), frames (representa el número actual de fotogramas del juego) y grados (representa la cantidad de grados a rotar el pájaro).

El pájaro cambia de animación según el valor de estado y frames. El contador frame se incrementa en cada periodo definido por estado, y luego se ajusta al rango válido utilizando el operador módulo (%).

Si el estado es 0, lo que indica que el juego está listo para comenzar, el pájaro vuelve a su posición inicial y su velocidad se restablece a 0. Además, la rotación se establece en 0 grados.

Si el estado no es 0, lo que significa que el juego está en progreso, el pájaro comienza a caer debido a la gravedad. La velocidad se incrementa según la gravedad y la posición del pájaro se actualiza en consecuencia. Si el pájaro toca el suelo, se detiene en la posición del suelo y se reproduce un sonido. La rotación del pájaro se ajusta dependiendo de la velocidad actual, simulando su movimiento de caída.

El último bloque comentado del código se usa para dibujar el radio del pájaro en el lienzo, pero está actualmente desactivado.

### flappyBird.js

Importaciones: Se importan las clases Imagen, Bird y Tuberias desde archivos externos.

Obtención del parámetro de la URL: El código utiliza el objeto URLSearchParams para obtener el parámetro llamado num de la URL actual. Este parámetro se guarda en la constante num.

Exportación de la constante numero: La constante num se exporta para que esté disponible pájaro.js.

Obtención del elemento de puntuación más alta: El código busca el elemento HTML con el ID highScore y lo asigna a la variable highScoreDiv.

Creación de la imagen del juego: Se crea una instancia de la clase Imagen llamada mapa utilizando el contexto de dibujo y la imagen cargada.

Creación del pájaro: Se crea una instancia de la clase Bird llamada pajaro utilizando el contexto de dibujo y la imagen cargada.

Creación de las tuberías: Se crea una instancia de la clase Tuberias llamada tuberias utilizando el contexto de dibujo y la imagen cargada.

Definición de los estados del juego: Se define un objeto llamado estado que contiene diferentes estados del juego, como corriendo, listo, juego y perdio.

Inicialización de variables de puntuación y sonido: Se inicializan las variables score y highScore en 0. También se crean instancias de los objetos de audio sonido1 y sonido2 para reproducir sonidos en el juego.

Función dibujar: Esta función se encarga de dibujar los elementos del juego en el lienzo. Se dibujan el cielo, el suelo, las tuberías, el pájaro y los textos de inicio y puntuación. Además, se llama a la función gameOver para dibujar el mensaje de fin del juego si corresponde.

Evento de clic: El código agrega un evento de clic al documento. Dependiendo del estado actual del juego (estado.corriendo), se ejecutan diferentes acciones. Por ejemplo, si el estado es listo, se cambia el estado a juego y se reproduce un sonido. Si el estado es juego, se llama al método aleteo del pájaro y se reproduce otro sonido. Si el estado es perdio, se maneja el reinicio del juego y se envía la puntuación al servidor mediante una petición POST.

Función gameOver: Esta función se encarga de dibujar el mensaje de fin del juego en el lienzo cuando el estado es perdio. Se dibuja una imagen y un botón de reinicio.

Función dibujarTextoInicio: Esta función dibuja el texto de inicio en el lienzo cuando el estado es listo.

Función dibujarPuntuacion: Esta función dibuja la puntuación en el lienzo. La puntuación se obtiene de las tuberías y se muestra en diferentes posiciones dependiendo del estado del juego.

Función update: Esta función actualiza el estado del juego y llama a los métodos actualizar de las instancias de Bird, Imagen y Tuberias. Además, comprueba si el estado del pájaro o las tuberías ha cambiado a perdio y cambia el estado del juego en consecuencia.

Función loop: Esta función se encarga de llamar a las funciones update, dibujar y requestAnimationFrame en un bucle continuo. Se incrementa la variable frames en cada iteración.

Llamada inicial a loop: Al final del código, se realiza la primera llamada a la función loop para iniciar el bucle del juego.

# BIBLIOGRAFÍA

https://github.com/jonatandaw2/PROYECTOTFCSETMINIJUEGOS/blob/master/proyecto/documentos/README.md