

Fundamentos Web: *El Tetris*

Fechas de entrega: 2 de Noviembre de 2021

Contexto

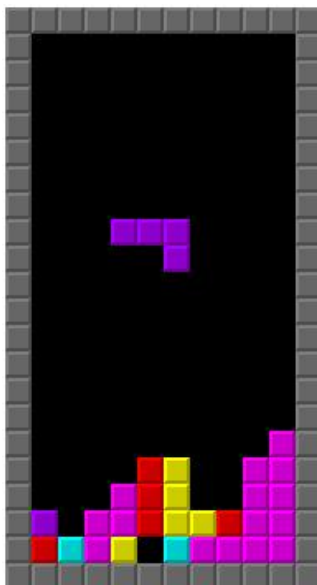
La industria de videojuegos ha crecido exponencialmente en las últimas décadas, hasta tal punto que tenemos en nuestra educación grados dedicados únicamente al desarrollo de videojuegos. En el año 2019, esta industria facturó *133.670 millones de euros*, pero no siempre fue así.

Aunque hay mucho debate sobre cuál fue el primer videojuego, el primer juego que se ha podido jugar en un ordenador es el **Spacewars**.

Este juego vio la luz en 1962 dentro del MIT, ocupaba 9kb de memoria y se necesitaba un hardware que costaba 120,000 dólares.

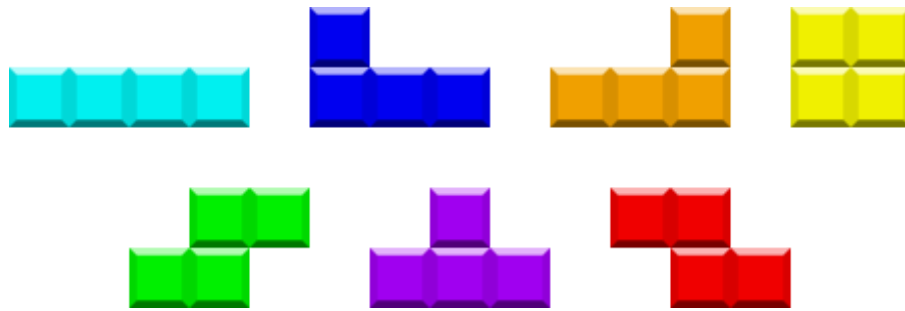


Unos años más tarde, el 6 de junio de 1984, se lanza uno de los videojuegos más importantes de la historia, **El Tetris**. El diseñador y creador del juego [Alekséi Pázhitnov](#) en la antigua URSS



En el Tetris se juega con los [tetrominós](#), el caso especial de cuatro elementos de [poliominós](#). Estas son las piezas que forman el juego.

En el juego tenemos 7 Tetrominos (cuyos nombres como aparecen en la imagen de derecha a izquierda son: I, J, L, O, S, T y Z)



La mecánica del juego es la siguiente:

Tetrominos van cayendo por un tablero y el jugador no puede evitar esta caída, pero si puede rotar la pieza (0, 90, 180 y 270 grados) y poder encajarlas dentro del tablero.

Cuando una fila está completa, esta se elimina del tablero y todas las piezas que hay encima bajarán una fila.

El juego termina cuando la pieza que cae al parar queda por fuera del tablero.

Objetivos

Desarrollar un videojuego clásico utilizando HTML, CSS y Vanilla JS.

Requisitos

Para realizar esta práctica se han de tener conocimientos de:

- HTML y Elementos semánticos de HTML
- CSS
- JS
 - Fundamentos del lenguaje
 - BOM y gestión del DOM

Trabajo en equipo

La práctica se va a realizar en equipos de 3 personas.

Para poder trabajar en equipo deberéis usar Git junto con su plataforma Github.

El modelo será como el que hacemos en clase pero entre vosotr@s.

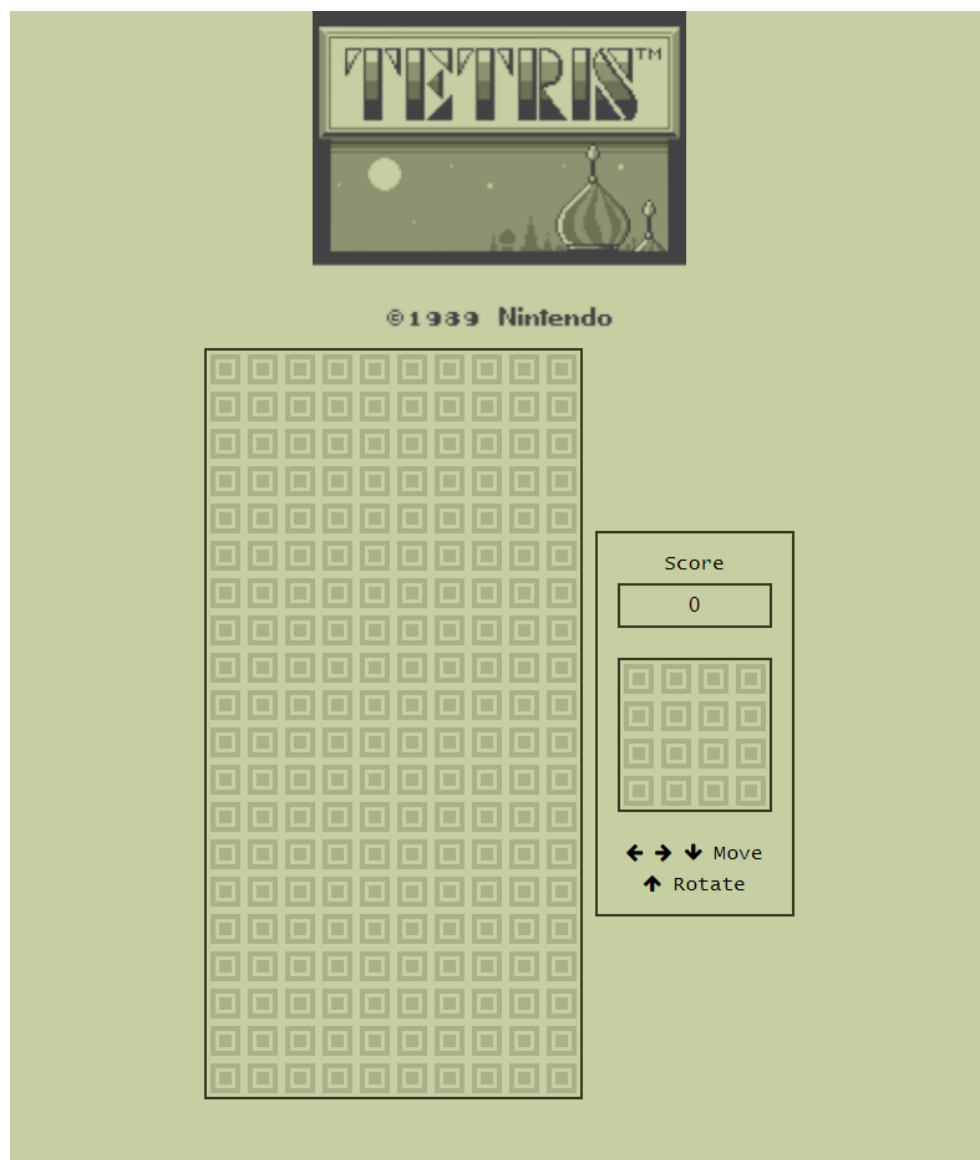
Alguien del equipo debe crear el repositorio y el resto de personas deberán hacerse su fork y modificar el repositorio principal a través de Pull Request.

Nunca se ha de escribir código en la rama main (ni el dueño del repositorio).

Diseños

Pantalla Principal

La pantalla principal del juego tendrá la visualización siguiente:



En primer lugar tendremos como cabecera una imagen centrada con un **width de 300px**. La imagen la podéis descargar del siguiente enlace

Luego tendremos dos paneles. El primer panel es un tablero, donde vamos a poner las piezas.

Este tablero tiene un tamaño de *300px x 600px* y dentro tiene una grid de *10x20* con bloques que miden *30px x 30px* cada uno.

a continuación tenemos el panel de estadísticas, donde vemos el score, una mini-grid de *4x4* con bloques del mismo tamaño que el board y la ayuda para los comandos del juego

Las celdas del board tienen una opacidad cuando están libres y no tendrán opacidad cuando están ocupadas, como se muestra en la siguiente imagen:



Fuente

La página tendrá la siguiente fuente:

```
font-family: "Lucida Console", Courier, monospace;
```

Colores

- Color de fondo del juego: #C6CFA2
- Color del texto: #303a21
- Opacidad de las celdas no ocupadas: 0.2
- Opacidad de las celdas ocupadas: 1

Música

- Tema principal: [tetris-Gameboy-made-in-JavaScript/theme.mp3 at master · Reyzartz/tetris-Gameboy-made-in-JavaScript · GitHub](#)
- Otros sonidos: [js-tetris/samples at master · az23/js-tetris · GitHub](#)

Funcionalidades

Durante la programación de este videojuego será importante ir consiguiendo pequeños hitos que nos ayuden a avanzar, mejor que intentar atacar todo el problema de una vez (Divide y Vencerás)

Los videojuegos llevan intrínsecos muchísima matemática y física. Para evitar meternos en conceptos como la transpuesta de una matriz, se utilizará un algoritmo a base de sumas, restas, multiplicaciones y divisiones.

Creación del tablero

Como se puede ver en los diseños, el tablero visualmente será una grid de 10x20. Para ello en vez de escribir los 200 divs, podemos utilizar JS y construirlos sobre su contenedor.

En primer lugar será buena idea la generación de dos constantes:

- *BOARD WIDTH*: Representa el ancho del tablero en número de bloques
- *BOARD_HEIGHT*: Representa el alto del tablero en número de bloques





























Para poder implementar el tablero y la interfaz base necesitaremos implementar las siguientes funciones:

- ***generateBoardBlock()***: función sin parámetros de entrada que genera y devuelve la estructura HTML que representa un bloque de un tablero
- ***drawBoard(containerClass, width, height)***: función que genera dentro del contenedor definido en *containerClass* (string) todos los bloques de hijos necesarios para cumplir con el *width* (number) y el *height* (number). Por ejemplo si tenemos 10 de width y 20 de height tendrá que generar 200 bloques de tablero dentro del contenedor. Esta función la podremos utilizar para pintar tanto el board general como el mini-board de previsualización de la siguiente pieza

Generación y visualización de los tetrominós

Una vez que hemos generado el board, lo siguiente que tenemos que hacer es saber generar las piezas y poder visualizarlas en el board.

En este caso el enfoque para la generación de las piezas serán unas constantes que representan a cada pieza y a sus rotaciones.

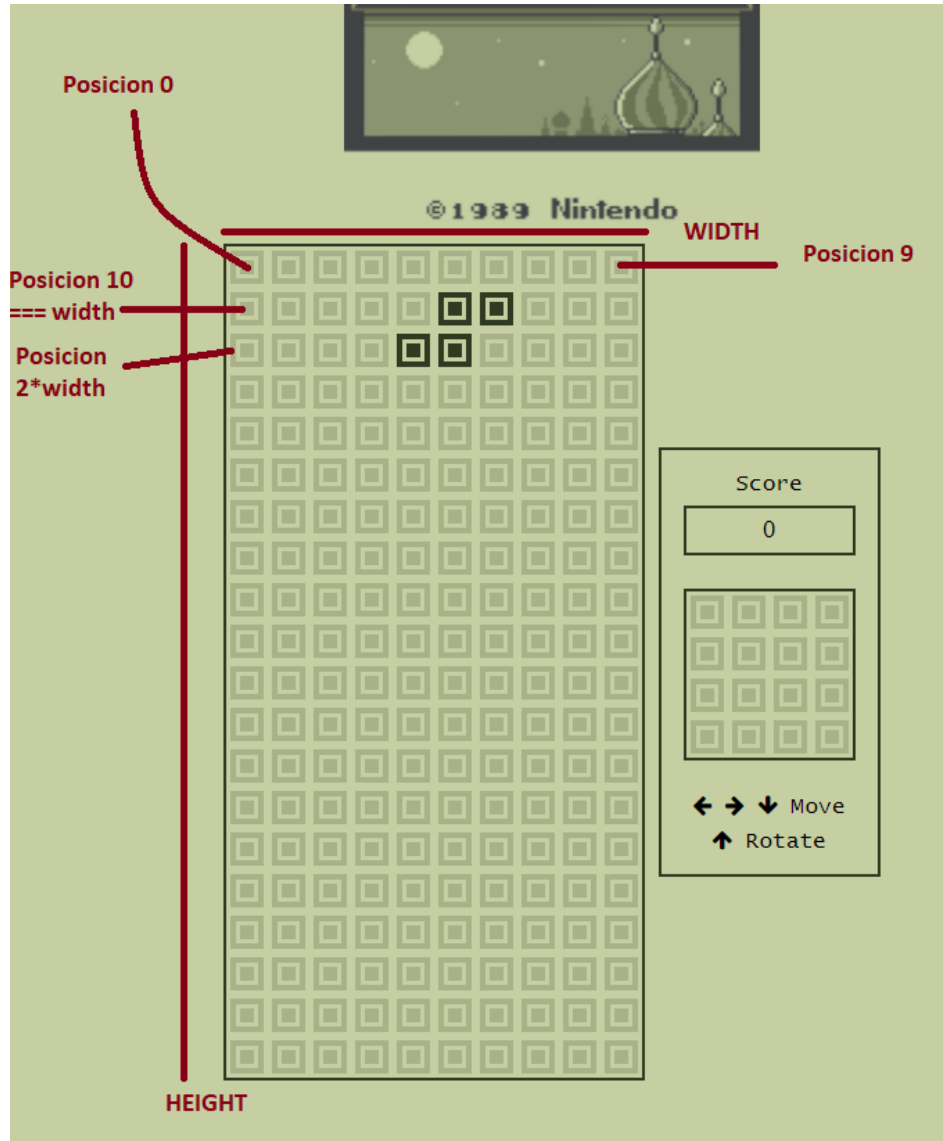
Tetrominoes	I	L	S	Z	J	O	T
r = 1							
r = 2							
r = 3							
r = 4							

Cada pieza será una constante con una lista de rotaciones, que representan las posibles rotaciones de cada pieza, como máximo de longitud 4 (dependiendo de la pieza).

Cada rotación a su vez será una lista de longitud 4 con la posición que debe tener la pieza dentro del board empezando desde la posición 0.

Nuestro board se visualizará como una matriz de filas y columnas (un array bidimensional) pero lo que tendremos al hacer el *querySelectorAll()*, será una array unidimensional. Necesitamos entonces hacer una extrapolación.

Si la posición (0,0) del tablero es el índice 0 del array, la siguiente fila (1,0) será el índice BOARD_WIDTH y la siguiente fila (2,0) será el índice 2*BOARD_WIDTH y así sucesivamente, como se muestra en la imagen siguiente.



Por ejemplo para la pieza I, tendremos dos rotaciones (ya que $r=3$ y $r=4$ es la misma).

La primera rotación tendrá las siguientes posiciones $[0, 1, 2, 3]$

La segunda rotación tendrá las siguientes posiciones $[1, BOARD_WIDTH+1, BOARD_WIDTH*2+1, BOARD_WIDTH*3+1]$

Una vez tengamos definidas las piezas como constantes globales, tendremos que definir una nueva constante que es **una lista de tetrominós**

A continuación definimos una variable global **currentTetrominoe**, donde se guardará la información de la pieza actual donde va cayendo

Por último necesitamos construir 5 funciones:

- **generateRandomTetrominoe()**: devuelve un Tetrominó de la lista de manera aleatoria. El método devuelve un objeto con las siguientes propiedades:
 - *positionAtTetrominoeList (number)*: la posición que ocupa en la lista original de tetrominos
 - *piece (Array<number>)*: La lista de posiciones de la rotación actual (en este caso la inicial)

- *position (number)*: La posición actual de la pieza en el tablero (al inicio estará en la fila 0, columna mitad del tablero)
- *rotation (number)*: el índice de la rotación actual de la pieza
- ***drawTetrominoelnMainBoard()***: pinta el tetromino actual en la posición en la que se encuentra. Para pintar habrá que quitarle la opacidad a los nodos del DOM que corresponden con las posiciones del tetrominó.
- ***undrawTetrominoelnMainBoard()***: limpia el tablero del tetromino actual, volviéndole a poner la opacidad
- ***drawTetrominoelnMiniBoard(tetrominoe)***: pinta el tetromino de entrada dentro del miniboard, suponiendo que es la posición 0.
- ***cleanMiniBoard()***: limpia el miniboard

Moviendo y rotando el tetrominó

Una vez que podemos pintar una pieza al inicio del tablero, llega el momento de poder moverla y rotarla.

Para ello necesitaremos definir 3 funciones que puedan gestionar el movimiento (abajo, derecha e izquierda) y otra que pueda gestionar la rotación. estas funciones además deben gestionar que la pieza no se salga del tablero o que no choque con ninguna otra pieza.:

Por último estas funciones se ejecutarán con la interacción del usuario, por lo que tendremos que escuchar cuando el usuario pulse la tecla de las flechas (arriba, abajo, derecha e izquierda) y ejecutar la acción que corresponda en cada caso.

Por tanto las funciones que habrá que construir son:

- ***moveRight()***: mueve la posición de la pieza actual una posición a la derecha. Comprobando que al realizar el movimiento no hay ninguna otra pieza ni hemos llegado al límite del tablero.
- ***moveLeft()***: mueve la posición de la pieza actual una posición a la izquierda. Comprobando que al realizar el movimiento no hay ninguna otra pieza ni hemos llegado al límite del tablero.
- ***moveDown()***: mueve la posición de la pieza actual ***una fila hacia abajo***. Comprobando que al realizar el movimiento no hay ninguna otra pieza ni hemos llegado al límite del tablero.
- ***rotate()***: cambia la rotación de la pieza actual por la siguiente en el array en orden creciente. Si no hay más rotaciones deberá empezar por la rotación 0 de nuevo. Debe comprobar antes de hacer la rotación que no hay ningún conflicto.

Todas las funciones deben devolver un ***booleano***, que indica si se ha podido o no realizar la operación.

El algoritmo general de todas las funciones es:

- *Comprobar que el cambio se puede realizar y si se puede:*
 - *Despintar la pieza actual*
 - *Cambiar la posición de la pieza*
 - *Pintar la pieza con la nueva posición*

Una vez que vayamos teniendo las funciones las podemos ir probando escuchando los eventos de teclado del usuario.

Como recomendación, intentad ir en el orden que os he puesto y no pasad a la siguiente hasta que no hayáis probado la anterior.

Si termináis esta parte ya podréis mover una pieza por todo el tablero (menos hacia arriba) y podréis rotarla

Game Loop

Todos los juegos tienen un algoritmo principal de ejecución que se va repitiendo hasta que el juego haya terminado.

Cuando programamos en JS este bucle no puede ser un for o un while, ya que es código síncrono que bloquea el hilo de ejecución de JS y no visualizaríamos nada en el HTML

Podemos decir que JavaScript se divide en frames. Cada frame ejecuta las funciones que estén preparadas para ser ejecutadas en dicho frame (Manejadores de eventos, timers, respuestas del servidor, etc).

Una vez que se termine la ejecución del código de un frame JS prepara el siguiente y lo ejecuta. Esto se le conoce como **Event Loop**

Si conseguimos que los frames no tarden más de 16,67ms, entonces nuestra web irá a 60fps.

Para poder hacer el bucle del juego necesitamos aprovecharnos del Event Loop de JS y repetir una misma función cada Xms.

Esto lo podemos hacer principalmente de dos formas

- Utilizando **setInterval** ponemos un intervalo cada segundo o dependiendo de la velocidad que queramos ponerle a la pieza.
- Utilizando **requestAnimationFrame**: que es una función que solicita a JS un nuevo frame. Esta función es la mejor opción, aunque un poco más avanzada así que con que hagáis la versión con el intervalo sería suficiente.

Para poder completar el game Loop necesitaremos varias funciónes que haga el bucle del juego

- **gameLoop()**: función que se encarga de llevar la lógica del juego que será la que ejecute el interval. Esta función tendrá que:
 - Mover la ficha actual una fila hacia abajo
 - Si hemos perdido, ejecutar gameOver
 - Si no hemos perdido
 - Si no se ha podido mover la pieza
 - eliminar las filas que estén completadas sumando el score
 - asignar a la pieza actual lo que haya en la pieza siguiente
 - limpiar el mini-board

- generar una nueva pieza siguiente que pintemos en el mini-board
- pintamos la pieza actual
- **isGameOver()**: Es una función devuelve true si la pieza actual se encuentra encima de otra pieza (si hemos puesto una pieza y está encima de otra cuando la ponemos, se produce un gameOver)
- **gameOver()**: Función que para el game loop y muestra un mensaje al usuario por encima del tablero indicando GAME OVER. Además si el usuario pulsa en el intro la partida entera se reinicia

Iniciando el juego

Una vez que tenemos el tablero, las piezas y el bucle del juego, llega el momento de iniciar toda la maquinaria.

Para ello tenemos que crear una función **init**

- limpie el tablero entero y el mini-board en caso que exista algo
- genere un nuevo tablero y un nuevo mini-board
- genere dos piezas y las guarde en las variables globales **currentTetrominoe** y **nextTetrominoe**
- Pinte la pieza current en el board y la pieza next en el mini-board
- Inicie el game loop

Eliminando una fila y scoring

En el juego cuando una fila se completa con tetrominos, eliminaremos los bloques de esa fila y los de las filas superiores bajarán una fila.

Cada fila que eliminemos debe sumar 50 puntos al marcador de la puntuación

para poder implementar esto necesitaremos una función:

- **updateTetrisBoard()**: Esta función tiene que mirar si hay una fila completa que esté ocupada. Si lo está la elimina desplazando todas las filas hay por encima en 1 y añadiendo 50 al scoring de la interfaz del usuario.

Si conseguimos tetris en un solo movimiento (eliminar 4 filas seguidas) se sumarán 1000 puntos al scoring extras.

hint: Un algoritmo fácil para desplazar las filas podría ser eliminar los elementos de la fila del array y concatenarlos al principio, esto ya tendrá la fila eliminada al principio

Música

La música es uno de los aspectos más importantes de un videojuego, la que nos hace sentir muchas emociones dentro del mismo.

Debemos modificar las funciones del juego para ir añadiendo la música que necesitamos.

Dentro del apartado de música tendremos además del archiconocido tema principal, diferentes archivos que habrá que ir ejecutando dependiendo de la acción del juego (conseguir una línea, gameOver, rotar, etc).

Para poder poner música dentro de una web, os dejo un post que explica como hacer play a los elementos de audio o que suenen de manera infinita.

[API de Audio Javascript - Javascript en español \(lenguajejs.com\)](https://lenguajejs.com/api-de-audio-javascript/)

Entrega

La entrega se realizará el Martes 2 de Noviembre y se expondrá a los compañeros en clase.

Se debe desplegar la URL pública de la web generada con Github Pages.

Os dejo un tutorial para poder desplegar un repositorio en github. [Creating and deploying a static website using Github Pages | by Erick Camacho | Medium](https://medium.com/@erickcamacho/creating-and-deploying-a-static-website-using-github-pages-by-erick-camacho-1234567890)