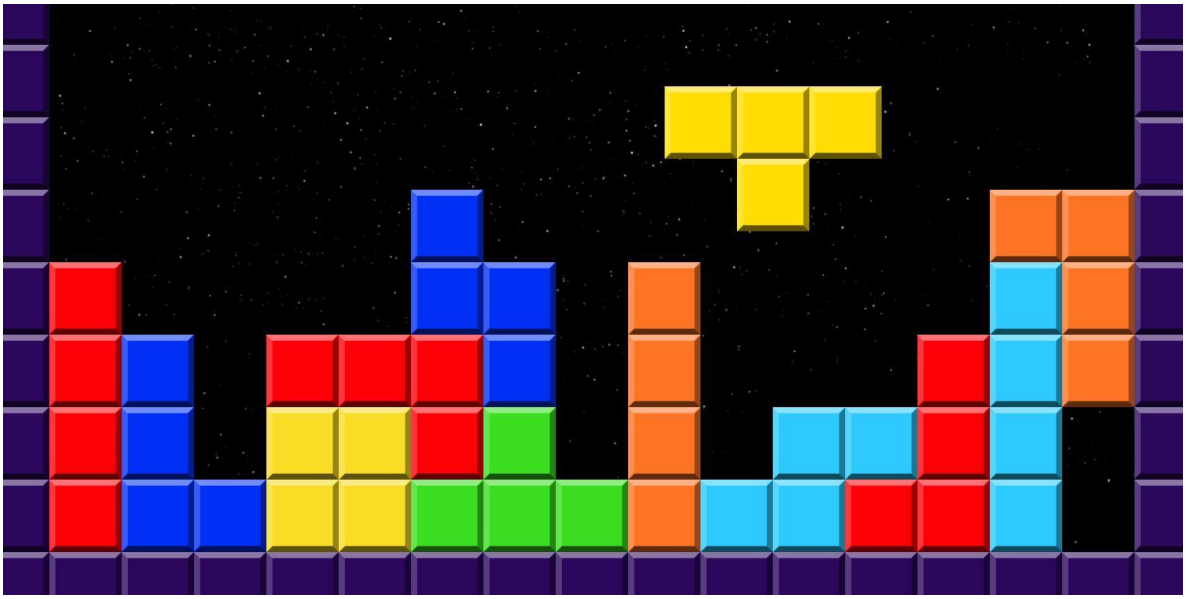

Juegos digitales programados en HTML, JAVA SCRIPT Y CSS

Generalidades del juego del tetris universalmente conocido

¿Qué es un juego digital tipo tetris, realizado en css, html y java script? y qué características presenta explicar detlladamente.

Un juego digital tipo Tetris realizado en HTML, CSS y JavaScript es una versión en la web del clásico juego de bloques, donde el jugador debe colocar figuras geométricas que caen desde la parte superior de la pantalla, de modo que completen líneas horizontales sin dejar espacios vacíos. Al completar una línea, esta desaparece, y se otorgan puntos al jugador. Si las piezas se acumulan hasta llegar a la parte superior de la pantalla, el juego termina.

Características clave de un Tetris digital en HTML, CSS y JavaScript:

1**. Interfaz gráfica (HTML y CSS):**

HTML define la estructura básica del juego, como el contenedor donde caen las piezas (el tablero) y los elementos interactivos, como botones o el marcador de puntos.

CSS se encarga del estilo visual, permitiendo la personalización de colores, formas y posiciones de los elementos. El fondo, los colores de las piezas y el diseño del tablero pueden ajustarse para darle una estética moderna o clásica.

2. Estructura del Tablero:

El tablero es una cuadrícula de celdas, típicamente de 10 columnas y 20 filas.

Cada celda representa un píxel donde las piezas pueden caer y posicionarse.

El tablero se implementa usando elementos HTML, como un div para representar cada celda, o a veces con un canvas para más control sobre los gráficos.

3. **Piezas geométricas (Tetrominós):**

Las piezas del juego, llamadas tetrominós, están formadas por bloques cuadrados. Las formas comunes incluyen:

I: Línea recta de cuatro bloques.

T: Forma de "T".

O: Cuadrado de 2x2 bloques.

L: Forma de "L".

J: Forma de "L" invertida.

S y Z: Formas en zig-zag.

Las piezas deben ser rotables, lo cual se maneja en JavaScript usando matrices que representan las diferentes orientaciones de cada tetrominó.

4. Lógica del Juego (JavaScript):

Movimiento de las piezas: Las piezas caen automáticamente hacia abajo con el tiempo, pero el jugador puede moverlas a izquierda o derecha, o rotarlas.

Detección de colisiones: Se debe comprobar si una pieza ha llegado al fondo del tablero o si ha chocado con otra pieza. Si ocurre una colisión, la pieza se bloquea en su posición actual.

Eliminación de líneas: Cada vez que una línea del tablero se llena completamente con bloques, esa línea se borra y el resto de las piezas sobre ella se mueve hacia abajo.

Puntaje y niveles: Al eliminar líneas, se suman puntos. A medida que el jugador avanza, el juego puede aumentar de nivel, haciendo que las piezas caigan más rápido.

Finalización del juego: Si las piezas se acumulan hasta el tope del tablero, el juego finaliza.

5. Controles del jugador:

El jugador interactúa con las piezas usando teclas del teclado:

Flecha izquierda (←) para mover la pieza hacia la izquierda.

Flecha derecha (→) para moverla hacia la derecha.

Flecha arriba (↑) para rotarla.

Flecha abajo (↓) para acelerar la caída.

Estos controles se manejan en JavaScript capturando eventos del teclado con addEventListener.

6. Velocidad y Niveles:

El juego comienza con una velocidad lenta para las piezas, pero a medida que el jugador avanza, las piezas caen más rápido. Esto se controla ajustando el intervalo de tiempo que regula el movimiento de las piezas hacia abajo.

7. Animación:

Las animaciones, como el movimiento de las piezas y el borrado de las líneas, se manejan mediante JavaScript. Se pueden usar setInterval o requestAnimationFrame para actualizar el estado del juego de manera periódica.

8. Responsive design:

El juego puede adaptarse a diferentes tamaños de pantalla, lo cual es importante si se desea que sea jugable tanto en computadoras de escritorio como en dispositivos móviles. Esto se logra utilizando unidades relativas en CSS y ajustando el tamaño del tablero dinámicamente con JavaScript.

9. Sonido y Efectos:

Opcionalmente, se pueden agregar efectos de sonido usando elementos <audio> o la API Web Audio para mejorar la experiencia del jugador, por ejemplo, al eliminar una línea o al perder el juego.

Ejemplo simplificado y general de la estructura de un Tetris:

código:HTML:

```
<div id="game-board"></div>
<div id="score">Score: 0</div>
```

Código CSS:

```
#game-board {
    display: grid;
    grid-template-columns: repeat(10, 30px);
    grid-template-rows: repeat(20, 30px);
    gap: 2px;
    background-color: #000;
}
.cell {
    width: 30px;
    height: 30px;
    background-color: #ccc;
}
```

Código JavaScript:

```
const board = document.getElementById('game-board');
const cells = [] y;

for (let i = 0; i < 200; i++) {
    const cell = document.createElement('div');
    cell.classList.add('cell');
    board.appendChild(cell);
    cells.push(cell);
}

let score = 0;
document.addEventListener('keydown', (event) => {
    if (event.key === 'ArrowRight') {
```

```
        // Mover la pieza a la derecha
    }
    // en este trayecto del código se pueden incluir más controles lo que haría que el juego sea más atractivo más complejo para los jugadores.
});
```

Este tipo de implementación como lo es el ejemplo anterior, fue realizado con las etiquetas en HTML, la programación en CSS y JavaScript crea un juego de Tetris funcional y personalizable, que puede ser mejorado con características adicionales como ***música***, ***temas visuales*** y ***modos de juego avanzados.***

EJEMPLO *CÓDIGO GENERAL PARA crear, diseñar y desarrollar UN JUEGO TIPO TETRIS PROGRAMADO EN HTML, CSS Y JAVA SCRIPT:*

Explicar en su cuaderno de trabajo de manera detallada y amplia el siguiente código que permite crear un juego tipo Tetris. Realice dibujar representativos que aomplien y mejoren sus explicaciones.

Crear una carpeta nueva en Visual Studio Code llamada JuegoTetris y dentro de ella un archivo llamado tetris.html digite el siguiente código y demuestre su funcionamiento para afianzar sus conocimientos en el uso y aprendizaje de la programación en los programas html, css y java script.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tetris Universal</title>
    <style>
        body {
            background-color: #FF4500; /* Anaranjado fosforescente */
            font-family: Arial, sans-serif;
            font-size: 18px;
            color: black;
            text-align: center;
        }
```

```
.container {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 50px;
}

canvas {
    background-color: #000;
    border: 4px solid #FFFFFF;
}

.dynamicButton {
    padding: 10px 20px;
    margin: 20px;
    background: linear-gradient(90deg, #ff00ff, #00ffff, #ffff00);
    border: none;
    color: white;
    font-size: 18px;
    cursor: pointer;
    border-radius: 10px;
    transition: transform 0.3s ease;
}

.dynamicButton:hover {
    transform: scale(1.1);
}
```

```
        .hidden {
            display: none;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>¡Bienvenido al Tetris Universal!</h1>
        <button id="startGame" class="dynamicButton">Vamos a Jugar</button>
        <canvas id="tetrisBoard" width="300" height="600"></canvas>
        <div id="winMessage" class="hidden">
            <p>¡Felicitaciones, has ganado el juego!</p>
            <button id="closeGame" class="dynamicButton">Cerrar el Programa</button>
        </div>
    </div>

    <script>
        const canvas = document.getElementById('tetrisBoard');
        const context = canvas.getContext('2d');
        const startGameBtn = document.getElementById('startGame');
        const winMessage = document.getElementById('winMessage');
        const closeGameBtn = document.getElementById('closeGame');
        const grid = 30; // Tamaño de cada celda
        const boardWidth = canvas.width / grid;
        const boardHeight = canvas.height / grid;
        let gameInterval;
        let board = Array.from({ length: boardHeight }, () => Array(boardWidth).fill(0));
```

```
let currentPiece;
let currentX = 4;
let currentY = 0;
let score = 0;
let isGameOver = false;

const pieces = [
    { shape: [[1, 1], [1, 1]], color: '#ff0000' }, // Cuadrado
    { shape: [[1, 1, 1], [0, 1, 0]], color: '#00ff00' }, // T
    { shape: [[1, 1, 0], [0, 1, 1]], color: '#0000ff' }, // Z
    { shape: [[0, 1, 1], [1, 1, 0]], color: '#ffff00' }, // S
    { shape: [[1, 1, 1, 1]], color: '#ff00ff' }, // Línea
];

function startGame() {
    startGameBtn.style.display = 'none';
    currentPiece = getRandomPiece();
    currentX = Math.floor(boardWidth / 2) - Math.floor(currentPiece.shape[0].length / 2);
    currentY = 0;
    gameInterval = setInterval(dropPiece, 500);
    document.addEventListener('keydown', controlPiece);
}

function getRandomPiece() {
    return pieces[Math.floor(Math.random() * pieces.length)];
}

function dropPiece() {
```

```
        if (isGameOver) return;

        if (canMove(0, 1)) {
            currentY++;
        } else {
            mergePiece();
            clearLines();
            currentPiece = getRandomPiece();
            currentX = Math.floor(boardWidth / 2) - Math.floor(currentPiece.shape[0].length / 2);
            currentY = 0;

            if (!canMove(0, 0)) {
                endGame();
            }
        }
        drawBoard();
    }

    function canMove(offsetX, offsetY) {
        return currentPiece.shape.every((row, y) => {
            return row.every((cell, x) => {
                const newX = currentX + x + offsetX;
                const newY = currentY + y + offsetY;
                return (
                    cell === 0 ||
                    (newY < boardHeight && newX >= 0 && newX < boardWidth && board[newY][newX] === 0)
                );
```

```
        });
    });
}

function mergePiece() {
    currentPiece.shape.forEach((row, y) => {
        row.forEach((cell, x) => {
            if (cell) {
                board[currentY + y][currentX + x] = currentPiece.color;
            }
        });
    });
}

function clearLines() {
    let linesCleared = 0;
    board = board.filter(row => {
        if (row.every(cell => cell !== 0)) {
            linesCleared++;
            return false;
        }
        return true;
    });
    while (board.length < boardHeight) {
        board.unshift(Array(boardWidth).fill(0));
    }

    if (linesCleared > 0) {
```

```
            score += linesCleared * 100;
            if (score >= 1000) {
                winMessage.classList.remove('hidden');
                clearInterval(gameInterval);
            }
        }
    }

    function drawBoard() {
        context.clearRect(0, 0, canvas.width, canvas.height);
        board.forEach((row, y) => {
            row.forEach((cell, x) => {
                if (cell) {
                    context.fillStyle = cell;
                    context.fillRect(x * grid, y * grid, grid, grid);
                    context.strokeRect(x * grid, y * grid, grid, grid);
                }
            });
        });
        context.fillStyle = currentPiece.color;
        currentPiece.shape.forEach((row, y) => {
            row.forEach((cell, x) => {
                if (cell) {
                    context.fillRect((currentX + x) * grid, (currentY + y) * grid, grid, grid);
                    context.strokeRect((currentX + x) * grid, (currentY + y) * grid, grid,
grid);
                }
            });
        });
```

```
    }

    function controlPiece(event) {
        if (isGameOver) return;

        if (event.key === 'ArrowLeft' && canMove(-1, 0)) {
            currentX--;
        } else if (event.key === 'ArrowRight' && canMove(1, 0)) {
            currentX++;
        } else if (event.key === 'ArrowDown') {
            dropPiece();
        } else if (event.key === 'ArrowUp') {
            rotatePiece();
        }
        drawBoard();
    }

    function rotatePiece() {
        const rotatedShape = currentPiece.shape[0].map((_, index) => currentPiece.shape.map(row => row[index])).reverse();
        const oldShape = currentPiece.shape;
        currentPiece.shape = rotatedShape;
        if (!canMove(0, 0)) {
            currentPiece.shape = oldShape;
        }
    }

    function endGame() {
        clearInterval(gameInterval);
```

```
            isGameOver = true;
            alert("Game Over");
        }

        closeGameBtn.addEventListener('click', () => {
            window.close(); // Cierra la ventana del navegador
        });

        startGameBtn.addEventListener('click', startGame);
    </script>
</body>
</html>
```