index.html

<!DOCTYPE html>

<html>

<head>

    <title>Tetris</title>

    <meta charset="UTF-8">

    <link rel="stylesheet" href="./assets/css/stylesheet.css">

</head>

<body>

    <canvas width="320" height="640" id="game"></canvas>

    <script src="./assets/js/index.js"></script>

</body>

</html>

stylesheet.css

html,

body {

    height: 100%;

    margin: 0;

}

body {

    background: black;

    display: flex;

    align-items: center;

    justify-content: center;

}

canvas {

    border: 1px solid white;

}

index.js

*// \* see https://tetris.fandom.com/wiki/Tetris\_Guideline*

*// Get a reference to the HTML canvas element.*

const canvas = document.getElementById('game');

*// this is a 2D game, so we need to define that here*

const context = canvas.getContext('2d');

*// Define the size of each grid cell.*

*// keep track of what is in every cell of the game using a 2d array*

*// tetris playfield is 10x20, with a few rows offscreen*

const grid = 32;

*// Create an empty array representing the playfield.*

const playfield = [];

*// Define the shapes of different tetrominos and their corresponding colors.*

*// how to draw each tetromino*

*// \* see https://tetris.fandom.com/wiki/SRS*

*// Each tetromino shape is represented by a matrix of 0s and 1s.*

const tetrominos = {

    'I': [

        [0, 0, 0, 0],

        [1, 1, 1, 1],

        [0, 0, 0, 0],

        [0, 0, 0, 0]

    ],

    'J': [

        [1, 0, 0],

        [1, 1, 1],

        [0, 0, 0],

    ],

    'L': [

        [0, 0, 1],

        [1, 1, 1],

        [0, 0, 0],

    ],

    'O': [

        [1, 1],

        [1, 1],

    ],

    'S': [

        [0, 1, 1],

        [1, 1, 0],

        [0, 0, 0],

    ],

    'Z': [

        [1, 1, 0],

        [0, 1, 1],

        [0, 0, 0],

    ],

    'T': [

        [0, 1, 0],

        [1, 1, 1],

        [0, 0, 0],

    ]

};

*// color of each tetromino*

const colors = {

    'I': 'cyan',

    'O': 'yellow',

    'T': 'purple',

    'S': 'green',

    'Z': 'red',

    'J': 'blue',

    'L': 'orange'

};

*// get a random integer between the range of [min,max]*

*// \* see https://stackoverflow.com/a/1527820/2124254*

*// This function generates a random integer between a minimum and maximum value.*

function getRandomInt(min, max) {

*// The following lines make sure the minimum and maximum values are integers.*

*// Math.ceil rounds a number up to the nearest whole number.*

*// Math.floor rounds a number down to the nearest whole number.*

    min = Math.ceil(min);

    max = Math.floor(max);

*// Math.random generates a random decimal between 0 and 1.*

*// By multiplying it by (max - min + 1) and adding min,*

*// we get a random integer within the desired range.*

    return Math.floor(Math.random() \* (max - min + 1)) + min;

}

*// generate a new tetromino sequence*

const tetrominoSequence = [];

*// \* see https://tetris.fandom.com/wiki/Random\_Generator*

*// This function generates a new sequence of tetromino shapes used in the game.*

function generateSequence() {

*// This array contains the names of different tetromino shapes.*

    const sequence = ['I', 'J', 'L', 'O', 'S', 'T', 'Z'];

*// While there are still elements in the sequence array,*

*// randomly select one element and remove it from the array.*

*// Add the selected element to another array called tetrominoSequence.*

    while (sequence.length) {

        const rand = getRandomInt(0, sequence.length - 1);

        const name = sequence.splice(rand, 1)[0];

        tetrominoSequence.push(name);

    }

}

*// get the next tetromino in the sequence*

*// This function gets the next tetromino shape from the sequence.*

function getNextTetromino() {

*// If the tetromino sequence is empty, generate a new sequence.*

    if (tetrominoSequence.length === 0) {

        generateSequence();

    }

*// Get the last element from the tetromino sequence and remove it.*

*// Use the name to get the corresponding matrix of the tetromino shape.*

*// Calculate the starting position of the tetromino.*

    const name = tetrominoSequence.pop();

    const matrix = tetrominos[name];

*// I and O start centered, all others start in left-middle*

    const col = playfield[0].length / 2 - Math.ceil(matrix[0].length / 2);

*// I starts on row 21 (-1), all others start on row 22 (-2)*

    const row = name === 'I' ? -1 : -2;

*// Return an object that represents the next tetromino shape.*

    return {

        name: name, *// name of the piece (L, O, etc.)*

        matrix: matrix, *// the current rotation matrix*

        row: row, *// current row (starts offscreen)*

        col: col *// current col*

    };

}

*// rotate an NxN matrix 90deg*

*// \* see https://codereview.stackexchange.com/a/186834*

*// This function rotates a matrix 90 degrees.*

function rotate(matrix) {

*// The following lines create a new matrix by rearranging the rows and columns of the original matrix.*

    const N = matrix.length - 1;

    const result = matrix.map((row, i) =>

        row.map((val, j) => matrix[N - j][i])

    );

*// Return the rotated matrix.*

    return result;

}

*// check to see if the new matrix/row/col is valid*

*// This function checks if a move is valid by checking if a tetromino can be placed in a certain position.*

function isValidMove(matrix, cellRow, cellCol) {

*// Iterate over each cell of the tetromino matrix.*

    for (let row = 0; row < matrix.length; row++) {

        for (let col = 0; col < matrix[row].length; col++) {

*// Check if the cell is filled and if it would collide with the playfield boundaries or other filled cells.*

            if (matrix[row][col] && (

                    cellCol + col < 0 ||

                    cellCol + col >= playfield[0].length ||

                    cellRow + row >= playfield.length ||

                    playfield[cellRow + row][cellCol + col])) {

                return false;

            }

        }

    }

*// If no collisions are found, the move is valid.*

    return true;

}

*// place the tetromino on the playfield*

function placeTetromino() {

*// Iterate over each cell of the tetromino matrix.*

    for (let row = 0; row < tetromino.matrix.length; row++) {

        for (let col = 0; col < tetromino.matrix[row].length; col++) {

            if (tetromino.matrix[row][col]) {

*// Check if any part of the tetromino is offscreen, which results in a game over.*

*// game over if piece has any part offscreen*

                if (tetromino.row + row < 0) {

                    return showGameOver();

                }

*// Place the tetromino in the corresponding cell of the playfield.*

                playfield[tetromino.row + row][tetromino.col + col] = tetromino.name;

            }

        }

    }

*// check for line clears starting from the bottom and working our way up*

    for (let row = playfield.length - 1; row >= 0;) {

        if (playfield[row].every(cell => !!cell)) {

*// drop every row above this one*

            for (let r = row; r >= 0; r--) {

                for (let c = 0; c < playfield[r].length; c++) {

                    playfield[r][c] = playfield[r - 1][c];

                }

            }

        } else {

            row--;

        }

    }

*// Get the next tetromino shape.*

    tetromino = getNextTetromino();

}

*// show the game over screen*

function showGameOver() {

*// Initialize variables for the game loop, current tetromino, animation frame, and game over state.*

    cancelAnimationFrame(rAF);

    gameOver = true;

*// Draw a black semi-transparent rectangle in the middle of the canvas.*

    context.fillStyle = 'black';

    context.globalAlpha = 0.75;

    context.fillRect(0, canvas.height / 2 - 30, canvas.width, 60);

*// Draw the text "GAME OVER!" in white.*

    context.globalAlpha = 1;

    context.fillStyle = 'white';

    context.font = '36px monospace';

    context.textAlign = 'center';

    context.textBaseline = 'middle';

    context.fillText('GAME OVER!', canvas.width / 2, canvas.height / 2);

}

*// populate the empty state*

for (let row = -2; row < 20; row++) {

    playfield[row] = [];

    for (let col = 0; col < 10; col++) {

        playfield[row][col] = 0;

    }

}

let count = 0;

let tetromino = getNextTetromino();

let rAF = null; *// keep track of the animation frame so we can cancel it*

let gameOver = false;

*// game loop*

*// This function is the game loop that runs continuously.*

*// It clears the canvas, draws the playfield and active tetromino,*

*// and handles the movement and placement of tetrominos.*

function loop() {

    rAF = requestAnimationFrame(loop);

    context.clearRect(0, 0, canvas.width, canvas.height);

*// draw the playfield*

    for (let row = 0; row < 20; row++) {

        for (let col = 0; col < 10; col++) {

            if (playfield[row][col]) {

                const name = playfield[row][col];

                context.fillStyle = colors[name];

*// drawing 1 px smaller than the grid creates a grid effect*

                context.fillRect(col \* grid, row \* grid, grid - 1, grid - 1);

            }

        }

    }

*// draw the active tetromino*

    if (tetromino) {

*// tetromino falls every 35 frames*

        if (++count > 35) {

            tetromino.row++;

            count = 0;

*// place piece if it runs into anything*

            if (!isValidMove(tetromino.matrix, tetromino.row, tetromino.col)) {

                tetromino.row--;

                placeTetromino();

            }

        }

        context.fillStyle = colors[tetromino.name];

        for (let row = 0; row < tetromino.matrix.length; row++) {

            for (let col = 0; col < tetromino.matrix[row].length; col++) {

                if (tetromino.matrix[row][col]) {

*// drawing 1 px smaller than the grid creates a grid effect*

                    context.fillRect((tetromino.col + col) \* grid, (tetromino.row + row) \* grid, grid - 1, grid - 1);

                }

            }

        }

    }

}

*// listen to keyboard events to*

*// move and rotate the tetromino.*

document.addEventListener('keydown', function (e) {

    if (gameOver) return;

*// left and right arrow keys (move)*

    if (e.which === 37 || e.which === 39) {

        const col = e.which === 37 ?

            tetromino.col - 1 :

            tetromino.col + 1;

        if (isValidMove(tetromino.matrix, tetromino.row, col)) {

            tetromino.col = col;

        }

    }

*// up arrow key (rotate)*

    if (e.which === 38) {

        const matrix = rotate(tetromino.matrix);

        if (isValidMove(matrix, tetromino.row, tetromino.col)) {

            tetromino.matrix = matrix;

        }

    }

*// down arrow key (drop)*

    if (e.which === 40) {

        const row = tetromino.row + 1;

        if (!isValidMove(tetromino.matrix, row, tetromino.col)) {

            tetromino.row = row - 1;

            placeTetromino();

            return;

        }

        tetromino.row = row;

    }

});

*// Start the game loop.*

rAF = requestAnimationFrame(loop);