<!DOCTYPE html>

<html>

<head>

  <title>Basic Tetris HTML Game</title>

  <meta charset="UTF-8">

  <style>

*/\*\**

*\* 기본 HTML 구조 및 스타일 설정*

*\**

*\* 이 섹션은 테트리스 게임의 기본 HTML 구조와 초기 스타일을 정의합니다.*

*\* - DOCTYPE 선언으로 HTML5 문서임을 명시*

*\* - 'Basic Tetris HTML Game'이라는 제목 설정*

*\* - UTF-8 문자 인코딩 사용*

*\* - html과 body 요소의 높이를 100%로 설정하고 마진 제거*

*\*/*

*/\*\**

*\* html과 body 요소의 기본 스타일 설정*

*\**

*\* 이 CSS 규칙은 html과 body 요소에 대한 기본 스타일을 정의합니다.*

*\* - 높이를 100%로 설정하여 전체 뷰포트를 차지하도록 합니다.*

*\* - 마진을 0으로 설정하여 브라우저 기본 여백을 제거합니다.*

*\**

*\* 이렇게 설정함으로써 게임 컨테이너가 전체 화면을 채우도록 합니다.*

*\*/*

  html, body {

    height: 100%;

    margin: 0;

  }

  /\*\*

   \* body 요소의 스타일 설정

   \*

   \* 이 CSS 규칙은 body 요소에 대한 스타일을 정의합니다.

   \* - 배경색을 검정색으로 설정하여 게임의 어두운 분위기를 조성합니다.

   \* - flexbox를 사용하여 내용을 중앙에 배치합니다.

   \* - align-items와 justify-content를 center로 설정하여 수직, 수평 중앙 정렬을 합니다.

   \*

   \* 이렇게 설정함으로써 게임 캔버스가 화면 중앙에 위치하게 됩니다.

   \*/

  body {

    background: black;

    display: flex;

    align-items: center;

    justify-content: center;

  }

  /\*\*

   \* 게임 컨테이너 스타일

   \*

   \* 게임 영역과 사이드 패널을 수평으로 배치합니다.

   \*/

  #game-container {

    display: flex;

    align-items: flex-end; /\* 하단 정렬 \*/

  }

  /\*\*

   \* 게임 캔버스 스타일

   \*

   \* 게임 보드의 테두리를 흰색으로 설정합니다.

   \*/

  canvas#game {

    border: 1px solid white;

  }

  /\*\*

   \* 사이드 패널 스타일

   \*

   \* 게임 정보, 컨트롤, 다음 조각 미리보기를 수직으로 배치합니다.

   \*/

  #side-panel {

    display: flex;

    flex-direction: column;

    justify-content: space-between;

    align-items: center;

    height: 640px;

    margin-left: 20px;

  }

  /\*\*

   \* 게임 정보 스타일

   \*

   \* 점수와 레벨 정보를 표시합니다.

   \*/

  #game-info {

    color: white;

    font-size: 20px;

    text-align: left;

  }

  /\*\*

   \* 다음 조각 미리보기 캔버스 스타일

   \*/

  #next-piece {

    border: 1px solid white;

  }

  /\*\*

   \* 게임 컨트롤 영역 스타일

   \*/

  #game-controls {

    display: flex;

    flex-direction: column;

    align-items: center;

    margin: 20px 0;

  }

  /\*\*

   \* 컨트롤 그룹 스타일

   \*

   \* 각 컨트롤 버튼과 라벨을 그룹화합니다.

   \*/

  .control-group {

    display: flex;

    flex-direction: column;

    align-items: center;

    margin-bottom: 10px;

  }

  /\*\*

   \* 컨트롤 라벨 스타일

   \*/

  .control-label {

    color: white;

    font-size: 18px;

    margin-bottom: 5px;

  }

  /\*\*

   \* 일시정지 및 다시 시작 버튼 공통 스타일

   \*/

  #pause-button, #restart-button {

    background-color: rgba(255, 255, 255, 0.7);

    border: none;

    border-radius: 5px;

    padding: 10px 15px;

    font-size: 18px;

    cursor: pointer;

    width: 100px;

  }

  /\*\*

   \* 버튼 호버 효과

   \*/

  #pause-button:hover, #restart-button:hover {

    background-color: rgba(255, 255, 255, 0.9);

  }

  /\*\*

   \* 일시정지 오버레이 스타일

   \*

   \* 게임이 일시정지될 때 표시되는 반투명 오버레이

   \*/

  #pause-overlay {

    position: absolute;

    top: 0;

    left: 0;

    width: 100%;

    height: 100%;

    background-color: rgba(0, 0, 0, 0.5);

    display: none;

    justify-content: center;

    align-items: center;

    z-index: 20;

  }

  /\*\*

   \* 일시정지 텍스트 스타일

   \*/

  #pause-text {

    color: white;

    font-size: 24px;

    font-weight: bold;

  }

  /\*\*

   \* 일시정지 메뉴 스타일

   \*/

  #pause-menu {

    display: flex;

    flex-direction: column;

    align-items: center;

    margin: 20px 0;

    color: white;

    font-size: 18px;

  }

  /\*\*

   \* 일시정지 버튼 스타일

   \*/

  #pause-button {

    background-color: rgba(255, 255, 255, 0.7);

    border: none;

    border-radius: 5px;

    padding: 10px 15px;

    font-size: 18px;

    cursor: pointer;

    margin-top: 10px;

  }

  /\*\*

   \* 일시정지 버튼 호버 효과

   \*/

  #pause-button:hover {

    background-color: rgba(255, 255, 255, 0.9);

  }

  /\*\*

   \* 다시 시작 버튼 스타일

   \*/

  #restart-button {

    background-color: rgba(255, 255, 255, 0.7);

    border: none;

    border-radius: 5px;

    padding: 10px 15px;

    font-size: 18px;

    cursor: pointer;

    margin-top: 10px;

  }

  /\*\*

   \* 다시 시작 버튼 호버 효과

   \*/

  #restart-button:hover {

    background-color: rgba(255, 255, 255, 0.9);

  }

  </style>

</head>

<body>

<!-- 게임 컨테이너 -->

<div id="game-container">

  <!-- 게임 영역 -->

  <div id="game-area">

    <!-- 테트리스 게임 캔버스 -->

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

    <!-- 일시정지 오버레이 -->

    <div id="pause-overlay">

      <div id="pause-text">일시정지</div>

    </div>

  </div>

  <!-- 사이드 패널 -->

  <div id="side-panel">

    <!-- 게임 정보 (점수, 레벨) -->

    <div id="game-info">

      <div id="score">점수: <span id="score-value">0</span></div>

      <div id="level">레벨: <span id="level-value">1</span></div>

    </div>

    <!-- 게임 컨트롤 (일시정지, 다시 시작) -->

    <div id="game-controls">

      <div class="control-group">

        <div class="control-label">일시정지</div>

        <button id="pause-button">II</button>

      </div>

      <div class="control-group">

        <div class="control-label">다시 시작</div>

        <button id="restart-button">↺</button>

      </div>

    </div>

    <!-- 다음 조각 미리보기 -->

    <canvas width="96" height="96" id="next-piece"></canvas>

  </div>

</div>

<!-- 효과음 -->

<audio id="clearSound" src="clear\_sound.mp3"></audio>

<script>

// 게임 상수 및 변수

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

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

const grid = 32;

const tetrominoSequence = [];

// 게임 상태

let playfield = [];

let score = 0;

let level = 1;

let speed = 35;

let isPaused = false;

let rAF = null;

let count = 0;

let tetromino = null;

let nextTetromino = null;

let gameOver = false;

const clearSound = document.getElementById('clearSound');

// 초기 게임 필드 생성

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

  playfield[row] = [];

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

    playfield[row][col] = 0;

  }

}

// 테트로미노 모양 정의

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]]

};

// 테트로미노 색상

const colors = {

  'I': 'cyan',

  'O': 'yellow',

  'T': 'purple',

  'S': 'green',

  'Z': 'red',

  'J': 'blue',

  'L': 'orange'

};

/\*\*

 \* 랜덤 정수 생성 함수

 \*

 \* @param {number} min - 최소값

 \* @param {number} max - 최대값

 \* @returns {number} min과 max 사이의 랜덤 정수

 \*/

function getRandomInt(min, max) {

  min = Math.ceil(min);

  max = Math.floor(max);

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

}

/\*\*

 \* 새로운 테트로미노 시퀀스 생성

 \*

 \* 모든 테트로미노 타입을 포함하는 랜덤 시퀀스를 생성합니다.

 \*/

function generateSequence() {

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

  while (sequence.length) {

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

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

    tetrominoSequence.push(name);

  }

}

/\*\*

 \* 다음 테트로미노 가져오기

 \*

 \* @returns {Object} 다음 테트로미노의 정보 (이름, 매트릭스, 시작 위치)

 \*/

function getNextTetromino() {

  if (tetrominoSequence.length === 0) {

    generateSequence();

  }

  const name = tetrominoSequence.pop();

  const matrix = tetrominos[name];

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

  const row = name === 'I' ? -1 : -2;

  return {

    name: name,

    matrix: matrix,

    row: row,

    col: col

  };

}

/\*\*

 \* 테트로미노 회전

 \*

 \* @param {Array} matrix - 회전할 테트로미노 매트릭스

 \* @returns {Array} 90도 시계방향으로 회전된 새 매트릭스

 \*/

function rotate(matrix) {

  const N = matrix.length - 1;

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

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

  );

  return result;

}

/\*\*

 \* 충돌 검사

 \*

 \* @param {Array} matrix - 테트로미노 매트릭스

 \* @param {number} cellRow - 테트로미노의 현재 행 위치

 \* @param {number} cellCol - 테트로미노의 현재 열 위치

 \* @returns {boolean} 유효한 이동인지 여부

 \*/

function isValidMove(matrix, cellRow, cellCol) {

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

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

      if (matrix[row][col] && (

          cellCol + col < 0 ||

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

          cellRow + row >= playfield.length ||

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

        ) {

        return false;

      }

    }

  }

  return true;

}

/\*\*

 \* 테트로미노 배치

 \*

 \* 현재 테트로미노를 게임 필드에 배치하고, 줄 제거 및 점수 계산을 수행합니다.

 \*/

function placeTetromino() {

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

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

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

        if (tetromino.row + row < 0) {

          return showGameOver();

        }

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

      }

    }

  }

  // 완성된 줄 제거

  let linesCleared = 0;

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

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

      // 줄 제거

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

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

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

        }

      }

      linesCleared++;

    }

    else {

      row--;

    }

  }

  // 점수 계산 및 효과음 재생

  if (linesCleared > 0) {

    score += linesCleared \* 100;

    document.getElementById('score-value').textContent = score;

    // 효과음 재생

    try {

      clearSound.currentTime = 0;

      clearSound.play().catch(e => console.error("효과음 재생 실패:", e));

    } catch (error) {

      console.error("효과음 재생 중 오류 발생:", error);

    }

    // 레벨 업 로직 (선택사항)

    if (score > level \* 1000) {

      level++;

      document.getElementById('level-value').textContent = level;

      speed = Math.max(speed - 5, 10); // 속도 증가, 최소 10으로 제한

    }

  }

  tetromino = nextTetromino;

  nextTetromino = getNextTetromino();

  drawNextPiece();

}

/\*\*

 \* 다음 조각 그리기

 \*

 \* 다음에 등장할 테트로미노를 미리보기 캔버스에 그립니다.

 \*/

function drawNextPiece() {

  const canvasNext = document.getElementById('next-piece');

  const contextNext = canvasNext.getContext('2d');

  contextNext.clearRect(0, 0, canvasNext.width, canvasNext.height);

  const matrix = nextTetromino.matrix;

  const blockSize = 24;

  const offsetX = (canvasNext.width - matrix[0].length \* blockSize) / 2;

  const offsetY = (canvasNext.height - matrix.length \* blockSize) / 2;

  contextNext.fillStyle = colors[nextTetromino.name];

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

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

      if (matrix[row][col]) {

        contextNext.fillRect(offsetX + col \* blockSize, offsetY + row \* blockSize, blockSize - 1, blockSize - 1);

      }

    }

  }

}

/\*\*

 \* 게임 오버 표시

 \*

 \* 게임 오버 메시지를 화면에 표시합니다.

 \*/

function showGameOver() {

  cancelAnimationFrame(rAF);

  gameOver = true;

  context.fillStyle = 'black';

  context.globalAlpha = 0.75;

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

  context.globalAlpha = 1;

  context.fillStyle = 'white';

  context.font = '36px monospace';

  context.textAlign = 'center';

  context.textBaseline = 'middle';

  context.fillText('게임 오버!', canvas.width / 2, canvas.height / 2);

}

/\*\*

 \* 게임 재시작

 \*

 \* 게임 상태를 초기화하고 새 게임을 시작합니다.

 \*/

function restartGame() {

  gameOver = false;

  score = 0;

  level = 1;

  speed = 35;

  playfield = [];

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

    playfield[row] = [];

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

      playfield[row][col] = 0;

    }

  }

  document.getElementById('score-value').textContent = score;

  document.getElementById('level-value').textContent = level;

  initGame();

  rAF = requestAnimationFrame(loop);

}

/\*\*

 \* 게임 루프

 \*

 \* 게임의 메인 루프 함수입니다. 게임 상태를 업데이트하고 화면을 그립니다.

 \*/

function loop() {

  rAF = requestAnimationFrame(loop);

  if (!isPaused) {

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

    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];

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

        }

      }

    }

    if (tetromino) {

      if (++count > speed) {

        tetromino.row++;

        count = 0;

        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]) {

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

          }

        }

      }

    }

  }

}

/\*\*

 \* 게임 초기화

 \*

 \* 게임 시작 시 초기 상태를 설정합니다.

 \*/

function initGame() {

  tetromino = getNextTetromino();

  nextTetromino = getNextTetromino();

  drawNextPiece();

}

/\*\*

 \* 일시정지 토글

 \*

 \* 게임의 일시정지 상태를 전환합니다.

 \*/

function togglePause() {

  isPaused = !isPaused;

  if (isPaused) {

    pauseOverlay.style.display = 'flex';

    cancelAnimationFrame(rAF);

  } else {

    pauseOverlay.style.display = 'none';

    rAF = requestAnimationFrame(loop);

  }

}

// 키보드 이벤트 리스너

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

  if (e.key === 'p' || e.key === 'P') {

    togglePause();

  }

  if (isPaused) return;

  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;

    }

  }

  if (e.which === 38) {

    const matrix = rotate(tetromino.matrix);

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

      tetromino.matrix = matrix;

    }

  }

  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;

  }

});

// 게임 시작

initGame();

rAF = requestAnimationFrame(loop);

// 일시정지 버튼 이벤트 리스너

document.getElementById('pause-button').addEventListener('click', togglePause);

// 다시 시작 버튼 이벤트 리스너

document.getElementById('restart-button').addEventListener('click', restartGame);

</script>

</body>

</html>