**title: מדריך מלא לבניית משחק טטריס description: מדריך צעד אחר צעד לבניית משחק טטריס מלא עם JavaScript ו-Canvas, כולל לוגיקת משחק, ניקוד ורמות קושי tags: [tetris, game-development, canvas, javascript, tutorial, step-by-step, game-logic, scoring] priority: גבוהה language: he**

**מדריך מלא לבניית משחק טטריס**

**מבוא**

טטריס הוא אחד המשחקים הקלאסיים והפופולריים ביותר בהיסטוריה של משחקי המחשב. בניית טטריס היא פרויקט מצוין ללמידת עקרונות פיתוח משחקים, כולל game loop, ניהול מצב, גרפיקה וטיפול באירועי משתמש. במדריך זה נבנה משחק טטריס מלא מאפס באמצעות JavaScript ו-Canvas API.

**מבנה הפרויקט**

**1. הגדרת הבסיס והקבועים**

// קבועים למשחק

const COLS = 10; // מספר עמודות

const ROWS = 20; // מספר שורות

const BLOCK\_SIZE = 30; // גודל בלוק בפיקסלים

const COLORS = [

null,

'#FF0D72', // I

'#0DC2FF', // J

'#0DFF72', // L

'#F538FF', // O

'#FF8E0D', // S

'#FFE138', // T

'#3877FF' // Z

];

// הגדרת הצורות (Tetrominos)

const SHAPES = [

[], // ריק

[[1,1,1,1]], // I

[[2,0,0],[2,2,2]], // J

[[0,0,3],[3,3,3]], // L

[[4,4],[4,4]], // O

[[0,5,5],[5,5,0]], // S

[[0,6,0],[6,6,6]], // T

[[7,7,0],[0,7,7]] // Z

];

**2. מחלקת הלוח (Board)**

class Board {

constructor(ctx) {

this.ctx = ctx;

this.grid = this.createEmptyGrid();

this.piece = null;

}

createEmptyGrid() {

return Array.from({length: ROWS}, () => Array(COLS).fill(0));

}

draw() {

// ניקוי הלוח

this.ctx.clearRect(0, 0, this.ctx.canvas.width, this.ctx.canvas.height);

// ציור הרשת

this.grid.forEach((row, y) => {

row.forEach((value, x) => {

if (value > 0) {

this.ctx.fillStyle = COLORS[value];

this.ctx.fillRect(x \* BLOCK\_SIZE, y \* BLOCK\_SIZE,

BLOCK\_SIZE, BLOCK\_SIZE);

this.ctx.strokeRect(x \* BLOCK\_SIZE, y \* BLOCK\_SIZE,

BLOCK\_SIZE, BLOCK\_SIZE);

}

});

});

// ציור החלק הנופל

if (this.piece) {

this.piece.draw();

}

}

valid(piece) {

return piece.shape.every((row, dy) => {

return row.every((value, dx) => {

const x = piece.x + dx;

const y = piece.y + dy;

return value === 0 ||

(this.insideWalls(x, y) && this.notOccupied(x, y));

});

});

}

insideWalls(x, y) {

return x >= 0 && x < COLS && y < ROWS;

}

notOccupied(x, y) {

return this.grid[y] && this.grid[y][x] === 0;

}

freeze() {

this.piece.shape.forEach((row, y) => {

row.forEach((value, x) => {

if (value > 0) {

this.grid[y + this.piece.y][x + this.piece.x] = value;

}

});

});

}

clearLines() {

let lines = 0;

this.grid.forEach((row, y) => {

if (row.every(value => value > 0)) {

this.grid.splice(y, 1);

this.grid.unshift(Array(COLS).fill(0));

lines++;

}

});

return lines;

}

}

**3. מחלקת החלק (Piece)**

class Piece {

constructor(ctx, typeId) {

this.ctx = ctx;

this.typeId = typeId;

this.shape = SHAPES[typeId];

this.color = COLORS[typeId];

this.x = Math.floor((COLS - this.shape[0].length) / 2);

this.y = 0;

}

draw() {

this.ctx.fillStyle = this.color;

this.shape.forEach((row, y) => {

row.forEach((value, x) => {

if (value > 0) {

this.ctx.fillRect((this.x + x) \* BLOCK\_SIZE,

(this.y + y) \* BLOCK\_SIZE,

BLOCK\_SIZE, BLOCK\_SIZE);

this.ctx.strokeRect((this.x + x) \* BLOCK\_SIZE,

(this.y + y) \* BLOCK\_SIZE,

BLOCK\_SIZE, BLOCK\_SIZE);

}

});

});

}

move(piece) {

this.x = piece.x;

this.y = piece.y;

this.shape = piece.shape;

}

rotate() {

const clone = JSON.parse(JSON.stringify(this));

clone.shape = this.shape[0].map((val, index) =>

this.shape.map(row => row[index]).reverse()

);

return clone;

}

}

**4. לוגיקת המשחק הראשית**

class Game {

constructor(canvas) {

this.canvas = canvas;

this.ctx = canvas.getContext('2d');

this.board = new Board(this.ctx);

this.score = 0;

this.lines = 0;

this.level = 1;

this.dropCounter = 0;

this.dropInterval = 1000; // מילישניות

this.lastTime = 0;

this.gameOver = false;

this.initEventHandlers();

this.reset();

}

initEventHandlers() {

document.addEventListener('keydown', event => {

if (this.gameOver) return;

switch(event.key) {

case 'ArrowLeft':

this.movePiece(-1);

break;

case 'ArrowRight':

this.movePiece(1);

break;

case 'ArrowDown':

this.drop();

break;

case 'ArrowUp':

this.rotatePiece();

break;

case ' ':

this.hardDrop();

break;

}

});

}

reset() {

this.board.grid = this.board.createEmptyGrid();

this.board.piece = new Piece(this.ctx, this.randomPiece());

this.score = 0;

this.lines = 0;

this.level = 1;

this.gameOver = false;

this.updateScore();

}

randomPiece() {

return Math.floor(Math.random() \* (SHAPES.length - 1)) + 1;

}

movePiece(dir) {

const clone = JSON.parse(JSON.stringify(this.board.piece));

clone.x += dir;

if (this.board.valid(clone)) {

this.board.piece.move(clone);

}

}

rotatePiece() {

const rotated = this.board.piece.rotate();

if (this.board.valid(rotated)) {

this.board.piece.move(rotated);

}

}

drop() {

const clone = JSON.parse(JSON.stringify(this.board.piece));

clone.y++;

if (this.board.valid(clone)) {

this.board.piece.move(clone);

} else {

this.board.freeze();

const clearedLines = this.board.clearLines();

this.updateScore(clearedLines);

this.board.piece = new Piece(this.ctx, this.randomPiece());

if (!this.board.valid(this.board.piece)) {

this.gameOver = true;

this.showGameOver();

}

}

}

hardDrop() {

while (true) {

const clone = JSON.parse(JSON.stringify(this.board.piece));

clone.y++;

if (!this.board.valid(clone)) break;

this.board.piece.move(clone);

this.score += 2;

}

this.drop();

}

updateScore(clearedLines = 0) {

const linePoints = [0, 40, 100, 300, 1200];

this.lines += clearedLines;

this.score += linePoints[clearedLines] \* this.level;

this.level = Math.floor(this.lines / 10) + 1;

this.dropInterval = Math.max(100, 1000 - (this.level - 1) \* 100);

document.getElementById('score').textContent = this.score;

document.getElementById('lines').textContent = this.lines;

document.getElementById('level').textContent = this.level;

}

showGameOver() {

this.ctx.fillStyle = 'rgba(0, 0, 0, 0.75)';

this.ctx.fillRect(0, 3 \* BLOCK\_SIZE, COLS \* BLOCK\_SIZE, 3 \* BLOCK\_SIZE);

this.ctx.fillStyle = 'white';

this.ctx.font = '2em Arial';

this.ctx.textAlign = 'center';

this.ctx.fillText('GAME OVER', COLS \* BLOCK\_SIZE / 2, 4.5 \* BLOCK\_SIZE);

this.ctx.font = '1em Arial';

this.ctx.fillText('Press R to restart', COLS \* BLOCK\_SIZE / 2, 5.5 \* BLOCK\_SIZE);

}

animate(time = 0) {

const deltaTime = time - this.lastTime;

this.lastTime = time;

if (!this.gameOver) {

this.dropCounter += deltaTime;

if (this.dropCounter > this.dropInterval) {

this.drop();

this.dropCounter = 0;

}

}

this.board.draw();

requestAnimationFrame(this.animate.bind(this));

}

start() {

this.animate();

}

}

**5. HTML לאתחול המשחק**

<!DOCTYPE html>

<html lang="he" dir="rtl">

<head>

<meta charset="UTF-8">

<title>משחק טטריס</title>

<style>

body {

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

margin: 0;

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

font-family: Arial, sans-serif;

}

.game-container {

display: flex;

gap: 20px;

padding: 20px;

background: rgba(255, 255, 255, 0.1);

border-radius: 10px;

box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);

}

canvas {

border: 2px solid #333;

background: #111;

}

.info-panel {

color: white;

padding: 20px;

background: rgba(0, 0, 0, 0.3);

border-radius: 5px;

}

.info-panel h2 {

margin: 0 0 20px 0;

}

.info-panel div {

margin: 10px 0;

font-size: 1.2em;

}

.controls {

margin-top: 30px;

padding-top: 20px;

border-top: 1px solid rgba(255, 255, 255, 0.3);

}

.controls h3 {

margin: 0 0 10px 0;

}

.controls p {

margin: 5px 0;

font-size: 0.9em;

}

</style>

</head>

<body>

<div class="game-container">

<canvas id="gameCanvas" width="300" height="600"></canvas>

<div class="info-panel">

<h2>טטריס</h2>

<div>ניקוד: <span id="score">0</span></div>

<div>שורות: <span id="lines">0</span></div>

<div>רמה: <span id="level">1</span></div>

<div class="controls">

<h3>בקרות:</h3>

<p>← → - הזזה</p>

<p>↑ - סיבוב</p>

<p>↓ - האצה</p>

<p>רווח - נפילה מהירה</p>

<p>R - התחלה מחדש</p>

</div>

</div>

</div>

<script src="tetris.js"></script>

<script>

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

const game = new Game(canvas);

document.addEventListener('keydown', event => {

if (event.key === 'r' || event.key === 'R') {

game.reset();

}

});

game.start();

</script>

</body>

</html>

**טיפים ושיפורים מתקדמים**

**1. מערכת Preview לחלק הבא**

class NextPiecePreview {

constructor(ctx) {

this.ctx = ctx;

this.nextPiece = null;

}

setNext(typeId) {

this.nextPiece = SHAPES[typeId];

this.color = COLORS[typeId];

}

draw() {

this.ctx.clearRect(0, 0, 120, 120);

if (!this.nextPiece) return;

const offsetX = (4 - this.nextPiece[0].length) / 2;

const offsetY = (4 - this.nextPiece.length) / 2;

this.ctx.fillStyle = this.color;

this.nextPiece.forEach((row, y) => {

row.forEach((value, x) => {

if (value > 0) {

this.ctx.fillRect((x + offsetX) \* 30,

(y + offsetY) \* 30,

30, 30);

this.ctx.strokeRect((x + offsetX) \* 30,

(y + offsetY) \* 30,

30, 30);

}

});

});

}

}

**2. מערכת Hold (שמירת חלק)**

class HoldSystem {

constructor() {

this.heldPiece = null;

this.canHold = true;

}

hold(currentPiece) {

if (!this.canHold) return null;

const temp = this.heldPiece;

this.heldPiece = currentPiece.typeId;

this.canHold = false;

return temp;

}

reset() {

this.canHold = true;

}

}

**3. אפקטים ויזואליים**

class Effects {

constructor(ctx) {

this.ctx = ctx;

this.particles = [];

}

createLineEffect(y) {

for (let x = 0; x < COLS; x++) {

this.particles.push({

x: x \* BLOCK\_SIZE + BLOCK\_SIZE / 2,

y: y \* BLOCK\_SIZE + BLOCK\_SIZE / 2,

vx: (Math.random() - 0.5) \* 5,

vy: (Math.random() - 0.5) \* 5,

life: 1.0,

color: `hsl(${Math.random() \* 360}, 100%, 50%)`

});

}

}

update(deltaTime) {

this.particles = this.particles.filter(p => {

p.x += p.vx;

p.y += p.vy;

p.life -= deltaTime \* 0.002;

return p.life > 0;

});

}

draw() {

this.particles.forEach(p => {

this.ctx.save();

this.ctx.globalAlpha = p.life;

this.ctx.fillStyle = p.color;

this.ctx.fillRect(p.x - 2, p.y - 2, 4, 4);

this.ctx.restore();

});

}

}

**4. שמירת High Scores**

class HighScores {

constructor() {

this.scores = this.load();

}

load() {

const saved = localStorage.getItem('tetrisHighScores');

return saved ? JSON.parse(saved) : [];

}

save() {

localStorage.setItem('tetrisHighScores', JSON.stringify(this.scores));

}

add(score, name = 'Anonymous') {

this.scores.push({ score, name, date: new Date().toISOString() });

this.scores.sort((a, b) => b.score - a.score);

this.scores = this.scores.slice(0, 10); // שמור רק את 10 הטובים

this.save();

}

isHighScore(score) {

return this.scores.length < 10 || score > this.scores[9].score;

}

}

**Best Practices**

**1. ביצועים**

* השתמש ב-requestAnimationFrame לאנימציות חלקות
* צייר רק את מה שהשתנה כשאפשר
* השתמש ב-offscreen canvas לרינדור מורכב

**2. חווית משתמש**

* הוסף צלילים ומוזיקה למשחק
* הוסף אנימציות מעבר חלקות
* שמור על תגובתיות גבוהה לקלט המשתמש

**3. קוד נקי**

* הפרד לוגיקה מתצוגה
* השתמש במחלקות לארגון הקוד
* תעד את הפונקציות המורכבות

**4. נגישות**

* הוסף תמיכה במקלדת מלאה
* אפשר התאמה אישית של הבקרות
* הוסף אפשרות להשהות את המשחק

**סיכום**

בניית משחק טטריס היא דרך מצוינת ללמוד עקרונות פיתוח משחקים. המשחק משלב אלמנטים רבים כמו game loop, ניהול מצב, גרפיקה, פיזיקה בסיסית וטיפול באירועים. הקוד המוצג כאן מספק בסיס מוצק שניתן להרחיב עם תכונות נוספות כמו מולטיפלייר, רמות מיוחדות ואפקטים מתקדמים.