<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Asteroid Blaster</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

background: linear-gradient(45deg, #0a0a0a, #1a1a2e, #16213e);

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

font-family: 'Courier New', monospace;

overflow: hidden;

}

.game-container {

position: relative;

width: 800px;

height: 600px;

background: radial-gradient(circle, #0f0f23 0%, #000000 100%);

border: 2px solid #00ffff;

border-radius: 10px;

overflow: hidden;

box-shadow: 0 0 50px rgba(0, 255, 255, 0.3);

}

canvas {

background: transparent;

display: block;

cursor: crosshair;

}

.ui {

position: absolute;

top: 20px;

left: 20px;

color: #00ffff;

font-size: 18px;

text-shadow: 0 0 10px rgba(0, 255, 255, 0.8);

z-index: 10;

}

.game-over {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

color: #ff0040;

font-size: 32px;

text-align: center;

display: none;

z-index: 20;

text-shadow: 0 0 20px rgba(255, 0, 64, 0.8);

}

.instructions {

position: absolute;

bottom: 20px;

left: 20px;

color: #00ffff;

font-size: 14px;

opacity: 0.8;

z-index: 10;

}

.stars {

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

z-index: 1;

}

.star {

position: absolute;

background: white;

border-radius: 50%;

animation: twinkle 2s infinite alternate;

}

@keyframes twinkle {

0% { opacity: 0.3; }

100% { opacity: 1; }

}

.restart-btn {

background: #00ffff;

color: #000;

border: none;

padding: 10px 20px;

font-size: 16px;

margin-top: 20px;

cursor: pointer;

border-radius: 5px;

transition: all 0.3s;

}

.restart-btn:hover {

background: #ff0040;

color: white;

transform: scale(1.1);

}

</style>

</head>

<body>

<div class="game-container">

<div class="stars" id="stars"></div>

<div class="ui">

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

<div>Lives: <span id="lives">3</span></div>

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

</div>

<div class="game-over" id="gameOver">

<div>GAME OVER</div>

<div style="font-size: 18px; margin-top: 10px;">Final Score: <span id="finalScore">0</span></div>

<button class="restart-btn" onclick="restartGame()">RESTART</button>

</div>

<div class="instructions">

WASD/Arrow Keys: Move | Space: Shoot | Mouse: Aim & Shoot

</div>

<canvas id="gameCanvas" width="800" height="600"></canvas>

</div>

<script>

// Game variables

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

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

let gameRunning = true;

let score = 0;

let lives = 3;

let level = 1;

let keys = {};

let mousePos = { x: 400, y: 300 };

// Game objects

let player;

let bullets = [];

let asteroids = [];

let particles = [];

// Player class

class Player {

constructor(x, y) {

this.x = x;

this.y = y;

this.width = 30;

this.height = 30;

this.speed = 5;

this.angle = 0;

this.targetAngle = 0;

this.rotationSpeed = 0.1;

this.invulnerable = 0;

this.isMoving = false;

}

update() {

let moveX = 0;

let moveY = 0;

this.isMoving = false;

// Smooth 8-directional movement

if (keys['w'] || keys['ArrowUp']) {

moveY = -this.speed;

this.isMoving = true;

}

if (keys['s'] || keys['ArrowDown']) {

moveY = this.speed;

this.isMoving = true;

}

if (keys['a'] || keys['ArrowLeft']) {

moveX = -this.speed;

this.isMoving = true;

}

if (keys['d'] || keys['ArrowRight']) {

moveX = this.speed;

this.isMoving = true;

}

// Normalize diagonal movement so it's not faster

if (moveX !== 0 && moveY !== 0) {

moveX \*= 0.707; // 1/√2

moveY \*= 0.707;

}

// Apply movement with screen bounds

this.x = Math.max(0, Math.min(canvas.width - this.width, this.x + moveX));

this.y = Math.max(0, Math.min(canvas.height - this.height, this.y + moveY));

// Always rotate to face movement direction when moving

if (this.isMoving && (moveX !== 0 || moveY !== 0)) {

this.targetAngle = Math.atan2(moveY, moveX);

// Smooth rotation towards movement direction

let angleDiff = this.targetAngle - this.angle;

// Normalize angle difference to [-π, π]

while (angleDiff > Math.PI) angleDiff -= Math.PI \* 2;

while (angleDiff < -Math.PI) angleDiff += Math.PI \* 2;

// Apply smooth rotation (faster when moving)

if (Math.abs(angleDiff) > 0.01) {

this.angle += angleDiff \* (this.rotationSpeed \* 2);

} else {

this.angle = this.targetAngle;

}

}

// When not moving, face mouse cursor

else {

const dx = mousePos.x - (this.x + this.width / 2);

const dy = mousePos.y - (this.y + this.height / 2);

this.targetAngle = Math.atan2(dy, dx);

let angleDiff = this.targetAngle - this.angle;

while (angleDiff > Math.PI) angleDiff -= Math.PI \* 2;

while (angleDiff < -Math.PI) angleDiff += Math.PI \* 2;

if (Math.abs(angleDiff) > 0.01) {

this.angle += angleDiff \* this.rotationSpeed;

} else {

this.angle = this.targetAngle;

}

}

// Reduce invulnerability

if (this.invulnerable > 0) this.invulnerable--;

}

draw() {

ctx.save();

ctx.translate(this.x + this.width / 2, this.y + this.height / 2);

ctx.rotate(this.angle + Math.PI / 2);

// Flash when invulnerable

if (this.invulnerable % 10 < 5) {

ctx.fillStyle = '#00ffff';

ctx.strokeStyle = '#ffffff';

ctx.lineWidth = 2;

// Draw ship

ctx.beginPath();

ctx.moveTo(0, -15);

ctx.lineTo(-10, 15);

ctx.lineTo(0, 10);

ctx.lineTo(10, 15);

ctx.closePath();

ctx.fill();

ctx.stroke();

}

ctx.restore();

}

shoot() {

bullets.push(new Bullet(

this.x + this.width / 2,

this.y + this.height / 2,

this.angle

));

}

takeDamage() {

if (this.invulnerable <= 0) {

lives--;

this.invulnerable = 120; // 2 seconds at 60fps

updateUI();

if (lives <= 0) {

gameOver();

}

}

}

}

// Bullet class

class Bullet {

constructor(x, y, angle) {

this.x = x;

this.y = y;

this.speed = 8;

this.angle = angle;

this.vx = Math.cos(angle) \* this.speed;

this.vy = Math.sin(angle) \* this.speed;

this.life = 100;

}

update() {

this.x += this.vx;

this.y += this.vy;

this.life--;

}

draw() {

ctx.fillStyle = '#ffff00';

ctx.shadowColor = '#ffff00';

ctx.shadowBlur = 10;

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

ctx.shadowBlur = 0;

}

isOffScreen() {

return this.x < 0 || this.x > canvas.width ||

this.y < 0 || this.y > canvas.height ||

this.life <= 0;

}

}

// Asteroid class

class Asteroid {

constructor(x, y, size) {

this.x = x;

this.y = y;

this.size = size;

this.speed = Math.random() \* 2 + 1;

this.angle = Math.random() \* Math.PI \* 2;

this.vx = Math.cos(this.angle) \* this.speed;

this.vy = Math.sin(this.angle) \* this.speed;

this.rotation = 0;

this.rotationSpeed = (Math.random() - 0.5) \* 0.1;

}

update() {

this.x += this.vx;

this.y += this.vy;

this.rotation += this.rotationSpeed;

// Wrap around screen

if (this.x < -this.size) this.x = canvas.width + this.size;

if (this.x > canvas.width + this.size) this.x = -this.size;

if (this.y < -this.size) this.y = canvas.height + this.size;

if (this.y > canvas.height + this.size) this.y = -this.size;

}

draw() {

ctx.save();

ctx.translate(this.x, this.y);

ctx.rotate(this.rotation);

ctx.strokeStyle = '#ff6600';

ctx.fillStyle = '#331100';

ctx.lineWidth = 2;

ctx.beginPath();

for (let i = 0; i < 8; i++) {

const angle = (i / 8) \* Math.PI \* 2;

const radius = this.size + Math.sin(angle \* 3) \* (this.size \* 0.3);

const x = Math.cos(angle) \* radius;

const y = Math.sin(angle) \* radius;

if (i === 0) ctx.moveTo(x, y);

else ctx.lineTo(x, y);

}

ctx.closePath();

ctx.fill();

ctx.stroke();

ctx.restore();

}

}

// Particle class for explosions

class Particle {

constructor(x, y, color) {

this.x = x;

this.y = y;

this.vx = (Math.random() - 0.5) \* 10;

this.vy = (Math.random() - 0.5) \* 10;

this.life = 30;

this.color = color;

this.size = Math.random() \* 3 + 1;

}

update() {

this.x += this.vx;

this.y += this.vy;

this.vx \*= 0.98;

this.vy \*= 0.98;

this.life--;

this.size \*= 0.98;

}

draw() {

ctx.globalAlpha = this.life / 30;

ctx.fillStyle = this.color;

ctx.fillRect(this.x, this.y, this.size, this.size);

ctx.globalAlpha = 1;

}

}

// Initialize game

function init() {

player = new Player(canvas.width / 2 - 15, canvas.height - 50);

bullets = [];

asteroids = [];

particles = [];

createAsteroids();

createStars();

updateUI();

}

function createAsteroids() {

asteroids = [];

const count = 4 + level;

for (let i = 0; i < count; i++) {

let x, y;

do {

x = Math.random() \* canvas.width;

y = Math.random() \* canvas.height;

} while (distance(x, y, player.x + player.width/2, player.y + player.height/2) < 150);

asteroids.push(new Asteroid(x, y, Math.random() \* 20 + 20));

}

}

function createStars() {

const starContainer = document.getElementById('stars');

starContainer.innerHTML = '';

for (let i = 0; i < 100; i++) {

const star = document.createElement('div');

star.className = 'star';

star.style.left = Math.random() \* 100 + '%';

star.style.top = Math.random() \* 100 + '%';

star.style.width = Math.random() \* 2 + 1 + 'px';

star.style.height = star.style.width;

star.style.animationDelay = Math.random() \* 2 + 's';

starContainer.appendChild(star);

}

}

function distance(x1, y1, x2, y2) {

return Math.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2);

}

function checkCollisions() {

// Bullet-asteroid collisions

for (let bulletIndex = bullets.length - 1; bulletIndex >= 0; bulletIndex--) {

const bullet = bullets[bulletIndex];

for (let asteroidIndex = asteroids.length - 1; asteroidIndex >= 0; asteroidIndex--) {

const asteroid = asteroids[asteroidIndex];

if (distance(bullet.x, bullet.y, asteroid.x, asteroid.y) < asteroid.size) {

// Create explosion particles

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

particles.push(new Particle(asteroid.x, asteroid.y, '#ff6600'));

}

// Split asteroid if large enough

if (asteroid.size > 15) {

for (let i = 0; i < 2; i++) {

asteroids.push(new Asteroid(

asteroid.x + (Math.random() - 0.5) \* 20,

asteroid.y + (Math.random() - 0.5) \* 20,

asteroid.size \* 0.6

));

}

}

bullets.splice(bulletIndex, 1);

asteroids.splice(asteroidIndex, 1);

score += 100;

updateUI();

break;

}

}

}

// Player-asteroid collisions

asteroids.forEach(asteroid => {

if (distance(player.x + player.width/2, player.y + player.height/2,

asteroid.x, asteroid.y) < asteroid.size + 15) {

player.takeDamage();

// Create damage particles

for (let i = 0; i < 15; i++) {

particles.push(new Particle(player.x + player.width/2, player.y + player.height/2, '#ff0040'));

}

}

});

}

function update() {

if (!gameRunning) return;

player.update();

// Update bullets

for (let i = bullets.length - 1; i >= 0; i--) {

bullets[i].update();

if (bullets[i].isOffScreen()) {

bullets.splice(i, 1);

}

}

// Update asteroids

asteroids.forEach(asteroid => asteroid.update());

// Update particles

for (let i = particles.length - 1; i >= 0; i--) {

particles[i].update();

if (particles[i].life <= 0) {

particles.splice(i, 1);

}

}

checkCollisions();

// Check if level complete

if (asteroids.length === 0) {

level++;

updateUI();

createAsteroids();

}

}

function draw() {

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

player.draw();

bullets.forEach(bullet => bullet.draw());

asteroids.forEach(asteroid => asteroid.draw());

particles.forEach(particle => particle.draw());

}

function gameLoop() {

update();

draw();

requestAnimationFrame(gameLoop);

}

function updateUI() {

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

document.getElementById('lives').textContent = lives;

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

}

function gameOver() {

gameRunning = false;

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

document.getElementById('gameOver').style.display = 'block';

}

function restartGame() {

gameRunning = true;

score = 0;

lives = 3;

level = 1;

document.getElementById('gameOver').style.display = 'none';

init();

}

// Event listeners

document.addEventListener('keydown', (e) => {

keys[e.key.toLowerCase()] = true;

keys[e.code] = true;

if (e.code === 'Space') {

e.preventDefault();

if (gameRunning) player.shoot();

}

});

document.addEventListener('keyup', (e) => {

keys[e.key.toLowerCase()] = false;

keys[e.code] = false;

});

canvas.addEventListener('mousemove', (e) => {

const rect = canvas.getBoundingClientRect();

mousePos.x = e.clientX - rect.left;

mousePos.y = e.clientY - rect.top;

});

canvas.addEventListener('click', (e) => {

if (gameRunning) player.shoot();

});

// Initialize and start game

init();

gameLoop();

</script>

</body>

</html>