```
// A cross-browser requestAnimationFrame
// See https://hacks.mozilla.org/2011/08/animating-with-javascript-from-setinterval-to-
requestanimationframe/
var requestAnimFrame = (function(){
    return window.requestAnimationFrame       ||
        window.webkitRequestAnimationFrame ||
        window.mozRequestAnimationFrame    ||
        window.oRequestAnimationFrame      ||
        window.msRequestAnimationFrame     ||
        function(callback){
            window.setTimeout(callback, 1000 / 60);
        };
})();

// Create the canvas
var canvas = document.createElement("canvas");
var ctx = canvas.getContext("2d");
canvas.width = 512;
canvas.height = 480;
document.body.appendChild(canvas);

// The main game loop
var lastTime;
function main() {
    var now = Date.now();
    var dt = (now - lastTime) / 1000.0;

    update(dt);
    render();

    lastTime = now;
    requestAnimFrame(main);
};

function init() {
    terrainPattern = ctx.createPattern(resources.get('img/terrain.png'), 'repeat');

    document.getElementById('play-again').addEventListener('click', function() {
        reset();
    });

    reset();
    lastTime = Date.now();
    main();
}

resources.load([
    'img/sprites.png',
    'img/terrain.png'
]);
resources.onReady(init);

// Game state
var player = {
    pos: [0, 0],
    sprite: new Sprite('img/sprites.png', [0, 0], [39, 39], 16, [0, 1])
};

var bullets = [];
var enemies = [];
var explosions = [];

var lastFire = Date.now();
var gameTime = 0;
var isGameOver;
var terrainPattern;

var score = 0;
var scoreEl = document.getElementById('score');

// Speed in pixels per second
var playerSpeed = 200;
var bulletSpeed = 500;
var enemySpeed = 100;

// Update game objects
function update(dt) {
    gameTime += dt;

    handleInput(dt);
    updateEntities(dt);

    // It gets harder over time by adding enemies using this
    // equation: 1-.993^gameTime
    if(Math.random() < 1 - Math.pow(.993, gameTime)) {
        enemies.push({
            pos: [canvas.width,
                  Math.random() * (canvas.height - 39)],
            sprite: new Sprite('img/sprites.png', [0, 78], [80, 39],
                               6, [0, 1, 2, 3, 2, 1])
        });
    }

    checkCollisions();

    scoreEl.innerHTML = score;
};

function handleInput(dt) {
    if(input.isDown('DOWN') || input.isDown('s')) {
        player.pos[1] += playerSpeed * dt;
    }

    if(input.isDown('UP') || input.isDown('w')) {
        player.pos[1] -= playerSpeed * dt;
    }

    if(input.isDown('LEFT') || input.isDown('a')) {
        player.pos[0] -= playerSpeed * dt;
    }

    if(input.isDown('RIGHT') || input.isDown('d')) {
        player.pos[0] += playerSpeed * dt;
    }

    if(input.isDown('SPACE') &&
       !isGameOver &&
       Date.now() - lastFire > 100) {
        var x = player.pos[0] + player.sprite.size[0] / 2;
        var y = player.pos[1] + player.sprite.size[1] / 2;

        bullets.push({ pos: [x, y],
                       dir: 'forward',
                       sprite: new Sprite('img/sprites.png', [0, 39], [18, 8]) });
        bullets.push({ pos: [x, y],
                       dir: 'up',
                       sprite: new Sprite('img/sprites.png', [0, 50], [9, 5]) });
        bullets.push({ pos: [x, y],
                       dir: 'down',
                       sprite: new Sprite('img/sprites.png', [0, 60], [9, 5]) });

        lastFire = Date.now();
    }
}

function updateEntities(dt) {
    // Update the player sprite animation
    player.sprite.update(dt);

    // Update all the bullets
    for(var i=0; i<bullets.length; i++) {
        var bullet = bullets[i];

        switch(bullet.dir) {
        case 'up': bullet.pos[1] -= bulletSpeed * dt; break;
        case 'down': bullet.pos[1] += bulletSpeed * dt; break;
        default:
            bullet.pos[0] += bulletSpeed * dt;
        }

        // Remove the bullet if it goes offscreen
        if(bullet.pos[1] < 0 || bullet.pos[1] > canvas.height ||
           bullet.pos[0] > canvas.width) {
            bullets.splice(i, 1);
            i--;
        }
    }

    // Update all the enemies
    for(var i=0; i<enemies.length; i++) {
        enemies[i].pos[0] -= enemySpeed * dt;
        enemies[i].sprite.update(dt);

        // Remove if offscreen
        if(enemies[i].pos[0] + enemies[i].sprite.size[0] < 0) {
            enemies.splice(i, 1);
            i--;
        }
    }

    // Update all the explosions
    for(var i=0; i<explosions.length; i++) {
        explosions[i].sprite.update(dt);

        // Remove if animation is done
        if(explosions[i].sprite.done) {
            explosions.splice(i, 1);
            i--;
        }
    }
}

// Collisions

function collides(x, y, r, b, x2, y2, r2, b2) {
    return !(r <= x2 || x > r2 ||
             b <= y2 || y > b2);
}

function boxCollides(pos, size, pos2, size2) {
    return collides(pos[0], pos[1],
                    pos[0] + size[0], pos[1] + size[1],
                    pos2[0], pos2[1],
                     pos2[0] + size2[0], pos2[1] + size2[1]);
}

function checkCollisions() {
    checkPlayerBounds();

    // Run collision detection for all enemies and bullets
    for(var i=0; i<enemies.length; i++) {
        var pos = enemies[i].pos;
        var size = enemies[i].sprite.size;

        for(var j=0; j<bullets.length; j++) {
            var pos2 = bullets[j].pos;
            var size2 = bullets[j].sprite.size;

            if(boxCollides(pos, size, pos2, size2)) {
                // Remove the enemy
                enemies.splice(i, 1);
                i--;

                // Add score
                score += 100;

                // Add an explosion
                explosions.push({
                    pos: pos,
                    sprite: new Sprite('img/sprites.png',
                                       [0, 117],
                                       [39, 39],
                                       16,
                                       [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
                                       null,
                                       true)
                });

                // Remove the bullet and stop this iteration
                bullets.splice(j, 1);
                break;
            }
        }

        if(boxCollides(pos, size, player.pos, player.sprite.size)) {
            gameOver();
        }
    }
}

function checkPlayerBounds() {
    // Check bounds
    if(player.pos[0] < 0) {
        player.pos[0] = 0;
    }
    else if(player.pos[0] > canvas.width - player.sprite.size[0]) {
        player.pos[0] = canvas.width - player.sprite.size[0];
    }

    if(player.pos[1] < 0) {
        player.pos[1] = 0;
    }
    else if(player.pos[1] > canvas.height - player.sprite.size[1]) {
        player.pos[1] = canvas.height - player.sprite.size[1];
    }
}

// Draw everything
```

```
function render() {
    ctx.fillStyle = terrainPattern;
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    // Render the player if the game isn't over
    if(!isGameOver) {
        renderEntity(player);
    }

    renderEntities(bullets);
    renderEntities(enemies);
    renderEntities(explosions);
};

function renderEntities(list) {
    for(var i=0; i<list.length; i++) {
        renderEntity(list[i]);
    }
}

function renderEntity(entity) {
    ctx.save();
    ctx.translate(entity.pos[0], entity.pos[1]);
    entity.sprite.render(ctx);
    ctx.restore();
}

// Game over
function gameOver() {
    document.getElementById('game-over').style.display = 'block';
    document.getElementById('game-over-overlay').style.display = 'block';
    isGameOver = true;
}

// Reset game to original state
function reset() {
    document.getElementById('game-over').style.display = 'none';
    document.getElementById('game-over-overlay').style.display = 'none';
    isGameOver = false;
    gameTime = 0;
    score = 0;

    enemies = [];
    bullets = [];

    player.pos = [50, canvas.height / 2];
};
```