## **index.html**

import \* as THREE from 'three';

const GAME\_STATE = {

    LOADING: 0,

    MENU: 1,

    PLAY: 2,

    PAUSE: 3,

    GAMEOVER: 4

};

let state = GAME\_STATE.LOADING;

let renderer, scene, camera, clock;

let player, arena, skybox;

let enemies = [];

let projectiles = [];

let pickups = [];

let particles = [];

let ui = {};

let input = {};

let spawnTimer = 0;

let wave = 1;

let score = 0;

let xp = 0;

let level = 1;

let playerHP = 100;

let playerMaxHP = 100;

let playerKi = 100;

let playerMaxKi = 100;

let playerStamina = 100;

let playerMaxStamina = 100;

let transformationActive = false;

let transformationTime = 0;

let transformationCooldown = 0;

let soulAbilityReady = false;

let soulAbilityCharge = 0;

let soulAbilityCooldown = 0;

let bossSoulActive = false;

let cameraTarget = new THREE.Vector3();

let cameraYaw = 0;

let cameraPitch = 0;

let mouseDown = false;

let pointerLocked = false;

let gameOverReason = '';

let lastAttackTime = 0;

let meleeCombo = 0;

let meleeComboTimer = 0;

let menuSelection = 0;

let gameOverSelection = 0;

let pauseSelection = 0;

let loadingProgress = 0;

let loadingDone = false;

let lastTimestamp = 0;

const ARENA\_RADIUS = 24;

const ENEMY\_SPAWN\_INTERVAL = 3;

const BOSS\_WAVE\_INTERVAL = 5;

const ENEMY\_BASE\_HP = 24;

const ENEMY\_BASE\_DAMAGE = 8;

const ENEMY\_BASE\_SPEED = 5;

const BOSS\_BASE\_HP = 80;

const BOSS\_BASE\_DAMAGE = 20;

const BOSS\_BASE\_SPEED = 7;

const PLAYER\_SPEED = 8;

const PLAYER\_DASH\_SPEED = 20;

const PLAYER\_FLY\_SPEED = 12;

const PLAYER\_ATTACK\_RANGE = 2.4;

const PLAYER\_ATTACK\_COOLDOWN = 0.36;

const PLAYER\_ATTACK\_DAMAGE = 16;

const PLAYER\_KI\_BLAST\_COST = 18;

const PLAYER\_KI\_BLAST\_DAMAGE = 24;

const PLAYER\_KI\_BLAST\_SPEED = 20;

const PLAYER\_ULTIMATE\_COST = 80;

const PLAYER\_DASH\_COST = 16;

const TRANSFORMATION\_DURATION = 8;

const TRANSFORMATION\_COOLDOWN = 6;

const TRANSFORMATION\_KI\_COST = 60;

const SOUL\_ABILITY\_CHARGE\_REQUIRED = 100;

const SOUL\_ABILITY\_COOLDOWN = 8;

const SOUL\_PICKUP\_RADIUS = 2;

const ENEMY\_ATTACK\_RANGE = 2.2;

const ENEMY\_ATTACK\_COOLDOWN = 0.7;

const ENEMY\_KI\_BLAST\_COST = 18;

const ENEMY\_KI\_BLAST\_DAMAGE = 12;

const ENEMY\_KI\_BLAST\_SPEED = 14;

const PARTICLE\_LIFETIME = 1;

const PARTICLE\_BURST\_COUNT = 18;

const XP\_PER\_ENEMY = 10;

const XP\_PER\_BOSS = 40;

const LEVEL\_UP\_XP = 80;

function randRange(a, b) {

    return a + Math.random() \* (b - a);

}

function clamp(v, a, b) {

    return Math.max(a, Math.min(b, v));

}

function getDirXZ(yaw) {

    return new THREE.Vector3(Math.sin(yaw), 0, Math.cos(yaw));

}

function get2DAngle(x, z) {

    return Math.atan2(x, z);

}

function setArenaSkybox() {

    const loader = new THREE.TextureLoader();

    const tex = loader.load('https:

        skybox.material.map = tex;

        loadingProgress += 0.3;

    });

}

function createArena() {

    const geo = new THREE.CylinderGeometry(ARENA\_RADIUS, ARENA\_RADIUS, 2, 64);

    const mat = new THREE.MeshToonMaterial({ color: 0x5e2d1e, gradientMap: null });

    const mesh = new THREE.Mesh(geo, mat);

    mesh.position.y = -1;

    mesh.receiveShadow = true;

    scene.add(mesh);

    return mesh;

}

function createSkybox() {

    const geo = new THREE.SphereGeometry(200, 24, 16);

    const mat = new THREE.MeshBasicMaterial({ color: 0x222244, side: THREE.BackSide });

    const mesh = new THREE.Mesh(geo, mat);

    scene.add(mesh);

    return mesh;

}

function createPlayer() {

    const group = new THREE.Group();

    const bodyMat = new THREE.MeshToonMaterial({ color: 0xcccccc });

    const hairMat = new THREE.MeshToonMaterial({ color: 0x444400, emissive: 0x222200 });

    const auraMat = new THREE.MeshToonMaterial({ color: 0xffff44, transparent: true, opacity: 0.24, emissive: 0xffff00 });

    const body = new THREE.Mesh(new THREE.CapsuleGeometry(0.7, 2.2, 6, 8), bodyMat);

    body.castShadow = true;

    group.add(body);

    const head = new THREE.Mesh(new THREE.SphereGeometry(0.7, 12, 8), bodyMat);

    head.position.y = 1.8;

    head.castShadow = true;

    group.add(head);

    const hair = new THREE.Mesh(new THREE.ConeGeometry(0.5, 1.2, 5), hairMat);

    hair.position.y = 2.5;

    group.add(hair);

    const aura = new THREE.Mesh(new THREE.SphereGeometry(1.8, 18, 10), auraMat);

    aura.position.y = 1.1;

    aura.visible = false;

    group.add(aura);

    group.position.set(0, 1, 0);

    group.userData = {

        speed: PLAYER\_SPEED,

        dash: false,

        dashTime: 0,

        flying: false,

        attackState: 0,

        attackTimer: 0,

        combo: 0,

        comboTimer: 0,

        kiRegen: 10,

        staminaRegen: 12,

        transformation: 0,

        transformationColor: 0xffff44,

        hair: hair,

        aura: aura,

        auraMat: auraMat,

        hairMat: hairMat,

        baseHairColor: 0x444400,

        baseAuraColor: 0xffff44

    };

    scene.add(group);

    return group;

}

function createEnemy(isBoss = false) {

    const group = new THREE.Group();

    const color = isBoss ? 0x8e34db : 0x44bfff;

    const bodyMat = new THREE.MeshToonMaterial({ color: color });

    const body = new THREE.Mesh(new THREE.CapsuleGeometry(isBoss ? 0.9 : 0.7, isBoss ? 2.8 : 1.7, 6, 8), bodyMat);

    group.add(body);

    const head = new THREE.Mesh(new THREE.SphereGeometry(isBoss ? 0.8 : 0.6, 10, 8), bodyMat);

    head.position.y = isBoss ? 2.2 : 1.5;

    group.add(head);

    group.position.set(randRange(-ARENA\_RADIUS \* 0.8, ARENA\_RADIUS \* 0.8), 1, randRange(-ARENA\_RADIUS \* 0.8, ARENA\_RADIUS \* 0.8));

    group.userData = {

        hp: isBoss ? BOSS\_BASE\_HP + wave \* 10 : ENEMY\_BASE\_HP + wave \* 4,

        maxHp: isBoss ? BOSS\_BASE\_HP + wave \* 10 : ENEMY\_BASE\_HP + wave \* 4,

        damage: isBoss ? BOSS\_BASE\_DAMAGE + wave \* 2 : ENEMY\_BASE\_DAMAGE + wave,

        speed: isBoss ? BOSS\_BASE\_SPEED + wave \* 0.6 : ENEMY\_BASE\_SPEED + wave \* 0.3,

        attackCooldown: 0,

        isBoss: isBoss,

        ki: isBoss ? 100 : 0,

        kiBlastTimer: isBoss ? randRange(2, 4) : 0,

        attackRange: isBoss ? 2.6 : 2.0,

        target: player

    };

    scene.add(group);

    return group;

}

function createKiBlast(pos, dir, color, speed, damage, owner) {

    const mat = new THREE.MeshToonMaterial({ color: color, emissive: color, transparent: true, opacity: 0.8 });

    const mesh = new THREE.Mesh(new THREE.SphereGeometry(0.38, 10, 6), mat);

    mesh.position.copy(pos);

    mesh.userData = {

        dir: dir.clone().normalize(),

        speed: speed,

        damage: damage,

        owner: owner,

        lifetime: 2

    };

    scene.add(mesh);

    return mesh;

}

function createSoulPickup(pos) {

    const mat = new THREE.MeshToonMaterial({ color: 0xffff99, emissive: 0xffee44, transparent: true, opacity: 0.7 });

    const mesh = new THREE.Mesh(new THREE.IcosahedronGeometry(0.68, 1), mat);

    mesh.position.copy(pos);

    mesh.userData = { rotate: true, time: 0 };

    scene.add(mesh);

    return mesh;

}

function createParticleBurst(pos, color, count = PARTICLE\_BURST\_COUNT, size = 0.22, speed = 6) {

    for (let i = 0; i < count; ++i) {

        const mat = new THREE.MeshToonMaterial({ color: color, transparent: true, opacity: 0.8 });

        const mesh = new THREE.Mesh(new THREE.SphereGeometry(size, 6, 4), mat);

        mesh.position.copy(pos);

        const dir = new THREE.Vector3(randRange(-1, 1), randRange(-1, 1), randRange(-1, 1)).normalize();

        mesh.userData = {

            dir: dir,

            speed: randRange(speed \* 0.4, speed),

            lifetime: PARTICLE\_LIFETIME

        };

        scene.add(mesh);

        particles.push(mesh);

    }

}

function createUI() {

    ui.health = document.createElement('div');

    ui.health.style.position = 'absolute';

    ui.health.style.left = '20px';

    ui.health.style.top = '20px';

    ui.health.style.color = '#fff';

    ui.health.style.font = 'bold 22px monospace';

    ui.health.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.health);

    ui.ki = document.createElement('div');

    ui.ki.style.position = 'absolute';

    ui.ki.style.left = '20px';

    ui.ki.style.top = '56px';

    ui.ki.style.color = '#ffe';

    ui.ki.style.font = 'bold 18px monospace';

    ui.ki.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.ki);

    ui.stamina = document.createElement('div');

    ui.stamina.style.position = 'absolute';

    ui.stamina.style.left = '20px';

    ui.stamina.style.top = '84px';

    ui.stamina.style.color = '#9ff';

    ui.stamina.style.font = 'bold 18px monospace';

    ui.stamina.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.stamina);

    ui.xp = document.createElement('div');

    ui.xp.style.position = 'absolute';

    ui.xp.style.left = '20px';

    ui.xp.style.top = '112px';

    ui.xp.style.color = '#fff';

    ui.xp.style.font = 'bold 18px monospace';

    ui.xp.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.xp);

    ui.score = document.createElement('div');

    ui.score.style.position = 'absolute';

    ui.score.style.right = '24px';

    ui.score.style.top = '20px';

    ui.score.style.color = '#fff';

    ui.score.style.font = 'bold 22px monospace';

    ui.score.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.score);

    ui.wave = document.createElement('div');

    ui.wave.style.position = 'absolute';

    ui.wave.style.right = '24px';

    ui.wave.style.top = '54px';

    ui.wave.style.color = '#fff';

    ui.wave.style.font = 'bold 18px monospace';

    ui.wave.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.wave);

    ui.transformation = document.createElement('div');

    ui.transformation.style.position = 'absolute';

    ui.transformation.style.left = '50%';

    ui.transformation.style.bottom = '60px';

    ui.transformation.style.transform = 'translateX(-50%)';

    ui.transformation.style.color = '#ffe600';

    ui.transformation.style.font = 'bold 20px monospace';

    ui.transformation.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.transformation);

    ui.soul = document.createElement('div');

    ui.soul.style.position = 'absolute';

    ui.soul.style.left = '50%';

    ui.soul.style.bottom = '30px';

    ui.soul.style.transform = 'translateX(-50%)';

    ui.soul.style.color = '#fff';

    ui.soul.style.font = 'bold 18px monospace';

    ui.soul.style.textShadow = '2px 2px 8px #222';

    document.body.appendChild(ui.soul);

    ui.center = document.createElement('div');

    ui.center.style.position = 'absolute';

    ui.center.style.left = '50%';

    ui.center.style.top = '50%';

    ui.center.style.transform = 'translate(-50%, -50%)';

    ui.center.style.color = '#fff';

    ui.center.style.font = 'bold 34px monospace';

    ui.center.style.textShadow = '2px 2px 16px #222';

    ui.center.style.pointerEvents = 'none';

    document.body.appendChild(ui.center);

    ui.menu = document.createElement('div');

    ui.menu.style.position = 'absolute';

    ui.menu.style.left = '50%';

    ui.menu.style.top = '54%';

    ui.menu.style.transform = 'translate(-50%, -50%)';

    ui.menu.style.color = '#fff';

    ui.menu.style.font = 'bold 28px monospace';

    ui.menu.style.textShadow = '2px 2px 16px #222';

    ui.menu.style.textAlign = 'center';

    document.body.appendChild(ui.menu);

}

function updateUI() {

    if (state === GAME\_STATE.PLAY) {

        ui.health.textContent = `HP: ${Math.round(playerHP)}/${playerMaxHP}`;

        ui.ki.textContent = `KI: ${Math.round(playerKi)}/${playerMaxKi}`;

        ui.stamina.textContent = `STAMINA: ${Math.round(playerStamina)}/${playerMaxStamina}`;

        ui.xp.textContent = `XP: ${xp}/${LEVEL\_UP\_XP}  LVL: ${level}`;

        ui.score.textContent = `Score: ${score}`;

        ui.wave.textContent = `Wave: ${wave}`;

        if (transformationActive) {

            ui.transformation.textContent = `TRANSFORMED! ${transformationTime.toFixed(1)}s`;

        } else if (transformationCooldown > 0) {

            ui.transformation.textContent = `Transformation cooldown: ${transformationCooldown.toFixed(1)}s`;

        } else {

            ui.transformation.textContent = '';

        }

        if (bossSoulActive) {

            ui.soul.textContent = `Soul Ability: ${soulAbilityReady ? 'READY!' : `${Math.round(soulAbilityCharge)}%`}`;

        } else {

            ui.soul.textContent = '';

        }

        ui.center.textContent = '';

        ui.menu.textContent = '';

    } else if (state === GAME\_STATE.MENU) {

        ui.health.textContent = '';

        ui.ki.textContent = '';

        ui.stamina.textContent = '';

        ui.xp.textContent = '';

        ui.score.textContent = '';

        ui.wave.textContent = '';

        ui.transformation.textContent = '';

        ui.soul.textContent = '';

        ui.center.textContent = 'Saiyan Soul Unleashed\nArena Battle';

        let menuStr = '';

        menuStr += menuSelection === 0 ? '> START GAME <\n' : '  START GAME\n';

        menuStr += menuSelection === 1 ? '> HOW TO PLAY <\n' : '  HOW TO PLAY\n';

        menuStr += menuSelection === 2 ? '> EXIT <' : '  EXIT';

        ui.menu.textContent = menuStr;

    } else if (state === GAME\_STATE.PAUSE) {

        ui.center.textContent = 'PAUSED';

        let menuStr = '';

        menuStr += pauseSelection === 0 ? '> RESUME <\n' : '  RESUME\n';

        menuStr += pauseSelection === 1 ? '> MAIN MENU <\n' : '  MAIN MENU\n';

        menuStr += pauseSelection === 2 ? '> EXIT <' : '  EXIT';

        ui.menu.textContent = menuStr;

    } else if (state === GAME\_STATE.GAMEOVER) {

        ui.center.textContent = 'GAME OVER!';

        let menuStr = '';

        menuStr += `Score: ${score}\nLevel: ${level}\n\n`;

        menuStr += gameOverSelection === 0 ? '> RETRY <\n' : '  RETRY\n';

        menuStr += gameOverSelection === 1 ? '> MAIN MENU <\n' : '  MAIN MENU\n';

        ui.menu.textContent = menuStr;

    } else if (state === GAME\_STATE.LOADING) {

        ui.center.textContent = 'Loading...\n' + Math.floor(loadingProgress \* 100) + '%';

        ui.menu.textContent = '';

    }

}

function clearUI() {

    for (let k in ui) {

        if (ui[k] && ui[k].parentElement) ui[k].parentElement.removeChild(ui[k]);

    }

}

function resetGame() {

    playerHP = playerMaxHP = 100;

    playerKi = playerMaxKi = 100;

    playerStamina = playerMaxStamina = 100;

    transformationActive = false;

    transformationTime = 0;

    transformationCooldown = 0;

    soulAbilityReady = false;

    soulAbilityCharge = 0;

    soulAbilityCooldown = 0;

    bossSoulActive = false;

    wave = 1;

    score = 0;

    xp = 0;

    level = 1;

    enemies.forEach(e => scene.remove(e));

    enemies = [];

    projectiles.forEach(p => scene.remove(p));

    projectiles = [];

    pickups.forEach(p => scene.remove(p));

    pickups = [];

    particles.forEach(p => scene.remove(p));

    particles = [];

    player.position.set(0, 1, 0);

    player.userData.hairMat.color.set(player.userData.baseHairColor);

    player.userData.aura.visible = false;

    player.userData.auraMat.color.set(player.userData.baseAuraColor);

    player.userData.speed = PLAYER\_SPEED;

    player.userData.kiRegen = 10;

    player.userData.staminaRegen = 12;

    player.userData.hair.scale.set(1, 1, 1);

    cameraYaw = 0;

    cameraPitch = 0;

    meleeCombo = 0;

    meleeComboTimer = 0;

    lastAttackTime = 0;

    spawnTimer = 0;

}

function spawnEnemies() {

    let count = 3 + Math.floor(wave \* 0.6);

    for (let i = 0; i < count; ++i) {

        enemies.push(createEnemy(false));

    }

    if (wave % BOSS\_WAVE\_INTERVAL === 0) {

        enemies.push(createEnemy(true));

    }

}

function handleInput(dt) {

    if (state !== GAME\_STATE.PLAY) return;

    let move = new THREE.Vector3();

    if (input['KeyW']) move.z -= 1;

    if (input['KeyS']) move.z += 1;

    if (input['KeyA']) move.x -= 1;

    if (input['KeyD']) move.x += 1;

    move.normalize();

    let speed = player.userData.speed;

    if (input['ShiftLeft'] && playerStamina > 0 && (move.x !== 0 || move.z !== 0)) {

        speed = PLAYER\_DASH\_SPEED;

        player.userData.dash = true;

        player.userData.dashTime = 0.17;

        playerStamina -= PLAYER\_DASH\_COST \* dt;

        playerStamina = clamp(playerStamina, 0, playerMaxStamina);

    } else {

        player.userData.dash = false;

    }

    if (input['Space']) {

        player.userData.flying = true;

        move.y += 1;

    } else {

        player.userData.flying = false;

    }

    if (move.length() > 0) {

        move.normalize();

        move.applyAxisAngle(new THREE.Vector3(0, 1, 0), cameraYaw);

        player.position.add(move.multiplyScalar(speed \* dt));

    }

    player.position.y = clamp(player.position.y, 1, 12);

    let dist = Math.sqrt(player.position.x \* player.position.x + player.position.z \* player.position.z);

    if (dist > ARENA\_RADIUS - 1.2) {

        let angle = Math.atan2(player.position.x, player.position.z);

        player.position.x = Math.sin(angle) \* (ARENA\_RADIUS - 1.2);

        player.position.z = Math.cos(angle) \* (ARENA\_RADIUS - 1.2);

        createParticleBurst(player.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xffffcc, 8, 0.12, 2);

    }

    if (playerStamina < playerMaxStamina && !player.userData.dash) {

        playerStamina += player.userData.staminaRegen \* dt;

        playerStamina = clamp(playerStamina, 0, playerMaxStamina);

    }

    if (playerKi < playerMaxKi && !input['Mouse1']) {

        playerKi += player.userData.kiRegen \* dt \* (transformationActive ? 1.7 : 1);

        playerKi = clamp(playerKi, 0, playerMaxKi);

    }

}

function handleAttacks(dt) {

    if (state !== GAME\_STATE.PLAY) return;

    if (input['Mouse1']) {

        if (clock.getElapsedTime() - lastAttackTime > PLAYER\_ATTACK\_COOLDOWN) {

            let enemy = getNearestEnemyInFront();

            if (enemy && player.position.distanceTo(enemy.position) < PLAYER\_ATTACK\_RANGE + 0.6) {

                meleeCombo = (meleeCombo + 1) % 3;

                meleeComboTimer = 0.28;

                lastAttackTime = clock.getElapsedTime();

                let dmg = PLAYER\_ATTACK\_DAMAGE + (transformationActive ? 18 : 0);

                enemy.userData.hp -= dmg;

                createParticleBurst(enemy.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xffe400, 8, 0.12, 2.8);

                soulAbilityCharge += 8;

                if (soulAbilityCharge >= SOUL\_ABILITY\_CHARGE\_REQUIRED) {

                    soulAbilityReady = true;

                    soulAbilityCharge = SOUL\_ABILITY\_CHARGE\_REQUIRED;

                }

                if (enemy.userData.hp <= 0) {

                    handleEnemyDeath(enemy);

                }

            }

        }

    }

    if (input['Mouse2']) {

        if (playerKi > PLAYER\_KI\_BLAST\_COST) {

            let dir = getDirXZ(cameraYaw);

            let pos = player.position.clone().add(new THREE.Vector3(0, 1.2, 0)).add(dir.clone().multiplyScalar(1.2));

            projectiles.push(createKiBlast(pos, dir, 0x44ffff, PLAYER\_KI\_BLAST\_SPEED + (transformationActive ? 8 : 0), PLAYER\_KI\_BLAST\_DAMAGE + (transformationActive ? 14 : 0), player));

            playerKi -= PLAYER\_KI\_BLAST\_COST;

        }

        input['Mouse2'] = false;

    }

    if (input['KeyE']) {

        if (bossSoulActive && soulAbilityReady && soulAbilityCooldown <= 0) {

            activateSoulAbility();

            soulAbilityReady = false;

            soulAbilityCharge = 0;

            soulAbilityCooldown = SOUL\_ABILITY\_COOLDOWN;

        }

        input['KeyE'] = false;

    }

    if (input['KeyQ']) {

        if (!transformationActive && transformationCooldown <= 0 && playerKi >= TRANSFORMATION\_KI\_COST) {

            transformationActive = true;

            transformationTime = TRANSFORMATION\_DURATION;

            transformationCooldown = TRANSFORMATION\_COOLDOWN + TRANSFORMATION\_DURATION;

            playerKi -= TRANSFORMATION\_KI\_COST;

            player.userData.aura.visible = true;

            player.userData.auraMat.color.set(0x66bbff);

            player.userData.hairMat.color.set(0x33aaff);

            player.userData.hair.scale.set(1.3, 2.1, 1.3);

            player.userData.speed = PLAYER\_SPEED \* 1.7;

            player.userData.kiRegen = 24;

            player.userData.staminaRegen = 20;

            createParticleBurst(player.position.clone().add(new THREE.Vector3(0, 1, 0)), 0x66bbff, 32, 0.18, 8);

        }

        input['KeyQ'] = false;

    }

    if (input['KeyF']) {

        for (let i = 0; i < pickups.length; ++i) {

            if (player.position.distanceTo(pickups[i].position) < SOUL\_PICKUP\_RADIUS) {

                absorbSoul(pickups[i]);

                break;

            }

        }

        input['KeyF'] = false;

    }

}

function updatePlayer(dt) {

    if (transformationActive) {

        transformationTime -= dt;

        if (transformationTime <= 0) {

            transformationActive = false;

            player.userData.aura.visible = false;

            player.userData.auraMat.color.set(player.userData.baseAuraColor);

            player.userData.hairMat.color.set(player.userData.baseHairColor);

            player.userData.hair.scale.set(1, 1, 1);

            player.userData.speed = PLAYER\_SPEED;

            player.userData.kiRegen = 10;

            player.userData.staminaRegen = 12;

        }

    }

    if (transformationCooldown > 0) {

        transformationCooldown -= dt;

        if (transformationCooldown < 0) transformationCooldown = 0;

    }

    if (soulAbilityCooldown > 0) {

        soulAbilityCooldown -= dt;

        if (soulAbilityCooldown < 0) soulAbilityCooldown = 0;

    }

}

function updateEnemies(dt) {

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

        let e = enemies[i];

        let dir = player.position.clone().sub(e.position);

        dir.y = 0;

        let dist = dir.length();

        if (dist > 0.1) dir.normalize();

        if (dist > e.userData.attackRange) {

            e.position.add(dir.multiplyScalar(e.userData.speed \* dt));

        }

        if (e.userData.isBoss) {

            e.userData.kiBlastTimer -= dt;

            if (e.userData.kiBlastTimer < 0 && e.userData.ki >= ENEMY\_KI\_BLAST\_COST) {

                let kdir = player.position.clone().sub(e.position).normalize();

                let pos = e.position.clone().add(new THREE.Vector3(0, 1.3, 0)).add(kdir.clone().multiplyScalar(1.1));

                projectiles.push(createKiBlast(pos, kdir, 0xff44dd, ENEMY\_KI\_BLAST\_SPEED, ENEMY\_KI\_BLAST\_DAMAGE, e));

                e.userData.ki -= ENEMY\_KI\_BLAST\_COST;

                e.userData.kiBlastTimer = randRange(2, 4);

            } else if (e.userData.ki < 100) {

                e.userData.ki += 20 \* dt;

            }

        }

        e.userData.attackCooldown -= dt;

        if (e.userData.attackCooldown < 0 && dist < e.userData.attackRange + 0.5) {

            e.userData.attackCooldown = ENEMY\_ATTACK\_COOLDOWN;

            playerHP -= e.userData.damage;

            createParticleBurst(player.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xff4444, 7, 0.12, 2.2);

            if (playerHP <= 0) {

                gameOverReason = 'defeat';

                setGameOver();

            }

        }

        let edist = Math.sqrt(e.position.x \* e.position.x + e.position.z \* e.position.z);

        if (edist > ARENA\_RADIUS - 1.2) {

            let angle = Math.atan2(e.position.x, e.position.z);

            e.position.x = Math.sin(angle) \* (ARENA\_RADIUS - 1.2);

            e.position.z = Math.cos(angle) \* (ARENA\_RADIUS - 1.2);

            createParticleBurst(e.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xffffcc, 6, 0.09, 1.5);

        }

    }

}

function getNearestEnemyInFront() {

    let minDist = 1000, closest = null;

    let forward = getDirXZ(cameraYaw);

    for (let i = 0; i < enemies.length; ++i) {

        let toEnemy = enemies[i].position.clone().sub(player.position);

        toEnemy.y = 0;

        let dist = toEnemy.length();

        if (dist < 3.8) {

            toEnemy.normalize();

            if (forward.dot(toEnemy) > 0.4) {

                if (dist < minDist) {

                    minDist = dist;

                    closest = enemies[i];

                }

            }

        }

    }

    return closest;

}

function handleEnemyDeath(enemy) {

    if (enemy.userData.isBoss) {

        createParticleBurst(enemy.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xffe400, 40, 0.23, 10);

        let soul = createSoulPickup(enemy.position.clone().add(new THREE.Vector3(0, 1, 0)));

        pickups.push(soul);

        score += 100 \* wave;

        xp += XP\_PER\_BOSS;

    } else {

        createParticleBurst(enemy.position.clone().add(new THREE.Vector3(0, 1, 0)), 0x44bfff, 18, 0.13, 4.8);

        score += 20 \* wave;

        xp += XP\_PER\_ENEMY;

    }

    scene.remove(enemy);

    enemies.splice(enemies.indexOf(enemy), 1);

    if (xp >= LEVEL\_UP\_XP) {

        level += 1;

        xp -= LEVEL\_UP\_XP;

        playerMaxHP += 20;

        playerMaxKi += 20;

        playerMaxStamina += 10;

        playerHP = playerMaxHP;

        playerKi = playerMaxKi;

        playerStamina = playerMaxStamina;

    }

}

function absorbSoul(soul) {

    bossSoulActive = true;

    soulAbilityReady = false;

    soulAbilityCharge = 0;

    soulAbilityCooldown = 0;

    scene.remove(soul);

    pickups.splice(pickups.indexOf(soul), 1);

    createParticleBurst(player.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xffe400, 30, 0.22, 8);

}

function activateSoulAbility() {

    let dir = getDirXZ(cameraYaw);

    let pos = player.position.clone().add(new THREE.Vector3(0, 1.2, 0)).add(dir.clone().multiplyScalar(1.2));

    let beam = new THREE.Mesh(new THREE.CylinderGeometry(0.5, 0.8, 12, 8), new THREE.MeshToonMaterial({ color: 0x88eeff, emissive: 0x22ccff, transparent: true, opacity: 0.7 }));

    beam.position.copy(pos.clone().add(dir.clone().multiplyScalar(6)));

    beam.rotation.x = Math.PI / 2;

    beam.rotation.z = -cameraYaw;

    beam.userData = { dir: dir, speed: 0, damage: 70 + 24 \* level, owner: player, lifetime: 0.6, beam: true };

    scene.add(beam);

    projectiles.push(beam);

    createParticleBurst(pos.clone().add(new THREE.Vector3(0, 0.5, 0)), 0x88eeff, 32, 0.19, 8);

}

function updateProjectiles(dt) {

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

        let p = projectiles[i];

        if (p.userData.beam) {

            p.userData.lifetime -= dt;

            if (p.userData.lifetime <= 0) {

                scene.remove(p);

                projectiles.splice(i, 1);

                continue;

            }

            let hit = false;

            for (let j = 0; j < enemies.length; ++j) {

                if (player.position.distanceTo(enemies[j].position) < 7) {

                    enemies[j].userData.hp -= p.userData.damage \* dt;

                    createParticleBurst(enemies[j].position.clone().add(new THREE.Vector3(0, 1, 0)), 0x88eeff, 6, 0.09, 2.5);

                    if (enemies[j].userData.hp <= 0) {

                        handleEnemyDeath(enemies[j]);

                    }

                    hit = true;

                }

            }

            if (hit) soulAbilityCharge = 0;

        } else {

            p.position.add(p.userData.dir.clone().multiplyScalar(p.userData.speed \* dt));

            p.userData.lifetime -= dt;

            if (p.userData.lifetime <= 0) {

                scene.remove(p);

                projectiles.splice(i, 1);

                continue;

            }

            let targets = p.userData.owner === player ? enemies : [player];

            for (let j = 0; j < targets.length; ++j) {

                if (p.position.distanceTo(targets[j].position) < 1.2) {

                    if (targets[j] === player) {

                        playerHP -= p.userData.damage;

                        createParticleBurst(player.position.clone().add(new THREE.Vector3(0, 1, 0)), 0xff4444, 8, 0.13, 2.2);

                        if (playerHP <= 0) {

                            gameOverReason = 'defeat';

                            setGameOver();

                        }

                    } else {

                        targets[j].userData.hp -= p.userData.damage;

                        createParticleBurst(targets[j].position.clone().add(new THREE.Vector3(0, 1, 0)), 0x44ffff, 8, 0.13, 2.2);

                        soulAbilityCharge += 5;

                        if (soulAbilityCharge >= SOUL\_ABILITY\_CHARGE\_REQUIRED) {

                            soulAbilityReady = true;

                            soulAbilityCharge = SOUL\_ABILITY\_CHARGE\_REQUIRED;

                        }

                        if (targets[j].userData.hp <= 0) {

                            handleEnemyDeath(targets[j]);

                        }

                    }

                    scene.remove(p);

                    projectiles.splice(i, 1);

                    break;

                }

            }

        }

    }

}

function updatePickups(dt) {

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

        let p = pickups[i];

        p.userData.time += dt;

        p.rotation.y += dt \* 2;

        p.position.y = 1.2 + Math.sin(p.userData.time \* 2) \* 0.3;

    }

}

function updateParticles(dt) {

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

        let p = particles[i];

        p.position.add(p.userData.dir.clone().multiplyScalar(p.userData.speed \* dt));

        p.userData.lifetime -= dt;

        if (p.userData.lifetime <= 0) {

            scene.remove(p);

            particles.splice(i, 1);

        }

    }

}

function stepWave() {

    if (enemies.length === 0) {

        ++wave;

        spawnEnemies();

    }

}

function setGameOver() {

    state = GAME\_STATE.GAMEOVER;

    updateUI();

}

function animatePlayer(dt) {

    let t = clock.getElapsedTime();

    player.children[0].position.y = Math.sin(t \* 3) \* 0.07;

    player.children[1].position.y = 1.8 + Math.sin(t \* 3 + 2) \* 0.07;

    player.userData.hair.position.y = 2.5 + Math.sin(t \* 4 + 1.5) \* 0.06;

    if (player.userData.aura.visible) {

        player.userData.aura.scale.set(1 + Math.sin(t \* 7) \* 0.08, 1 + Math.sin(t \* 8) \* 0.08, 1 + Math.sin(t \* 9) \* 0.08);

    }

}

function animateEnemies(dt) {

    let t = clock.getElapsedTime();

    for (let i = 0; i < enemies.length; ++i) {

        enemies[i].children[0].position.y = Math.sin(t \* 3 + i) \* 0.05;

        enemies[i].children[1].position.y = (enemies[i].userData.isBoss ? 2.2 : 1.5) + Math.sin(t \* 3 + i \* 2) \* 0.05;

    }

}

function animateCamera(dt) {

    let camDist = 12 + (input['Scroll'] || 0);

    camDist = clamp(camDist, 8, 22);

    cameraTarget.copy(player.position).add(new THREE.Vector3(0, 2.8, 0));

    let camOffset = new THREE.Vector3(0, 0, camDist);

    camOffset.applyAxisAngle(new THREE.Vector3(1, 0, 0), cameraPitch);

    camOffset.applyAxisAngle(new THREE.Vector3(0, 1, 0), cameraYaw);

    camera.position.copy(cameraTarget).add(camOffset);

    camera.lookAt(cameraTarget);

}

function gameLoop(ts) {

    requestAnimationFrame(gameLoop);

    let dt = clock.getDelta();

    if (dt > 0.1) dt = 0.1;

    if (state === GAME\_STATE.LOADING) {

        loadingProgress += 0.015;

        if (!loadingDone && loadingProgress >= 1.0) {

            loadingDone = true;

            setTimeout(() => {

                state = GAME\_STATE.MENU;

                updateUI();

            }, 600);

        }

        updateUI();

        renderer.render(scene, camera);

        return;

    }

    if (state === GAME\_STATE.MENU) {

        updateUI();

        renderer.render(scene, camera);

        return;

    }

    if (state === GAME\_STATE.PAUSE) {

        updateUI();

        renderer.render(scene, camera);

        return;

    }

    if (state === GAME\_STATE.GAMEOVER) {

        updateUI();

        renderer.render(scene, camera);

        return;

    }

    handleInput(dt);

    handleAttacks(dt);

    updatePlayer(dt);

    updateEnemies(dt);

    updateProjectiles(dt);

    updatePickups(dt);

    updateParticles(dt);

    stepWave();

    animatePlayer(dt);

    animateEnemies(dt);

    animateCamera(dt);

    updateUI();

    renderer.render(scene, camera);

}

function setupInput() {

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

        if (state === GAME\_STATE.PLAY) {

            input[e.code] = true;

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

                state = GAME\_STATE.PAUSE;

                updateUI();

            }

        } else if (state === GAME\_STATE.MENU) {

            if (e.code === 'ArrowUp' || e.code === 'KeyW') {

                menuSelection = (menuSelection + 2) % 3;

                updateUI();

            }

            if (e.code === 'ArrowDown' || e.code === 'KeyS') {

                menuSelection = (menuSelection + 1) % 3;

                updateUI();

            }

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

                if (menuSelection === 0) {

                    resetGame();

                    state = GAME\_STATE.PLAY;

                    updateUI();

                } else if (menuSelection === 1) {

                    ui.center.textContent = 'Controls:\nWASD: Move\nSpace: Fly\nShift: Dash\nMouse: Camera\nLClick: Melee\nRClick: Ki Blast\nQ: Transform\nE: Soul Ability\nF: Absorb Soul\nESC: Pause';

                    ui.menu.textContent = 'Press any key...';

                } else {

                    window.close();

                }

            }

        } else if (state === GAME\_STATE.PAUSE) {

            if (e.code === 'ArrowUp' || e.code === 'KeyW') {

                pauseSelection = (pauseSelection + 2) % 3;

                updateUI();

            }

            if (e.code === 'ArrowDown' || e.code === 'KeyS') {

                pauseSelection = (pauseSelection + 1) % 3;

                updateUI();

            }

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

                if (pauseSelection === 0) {

                    state = GAME\_STATE.PLAY;

                    updateUI();

                } else if (pauseSelection === 1) {

                    state = GAME\_STATE.MENU;

                    updateUI();

                } else {

                    window.close();

                }

            }

        } else if (state === GAME\_STATE.GAMEOVER) {

            if (e.code === 'ArrowUp' || e.code === 'KeyW') {

                gameOverSelection = (gameOverSelection + 1) % 2;

                updateUI();

            }

            if (e.code === 'ArrowDown' || e.code === 'KeyS') {

                gameOverSelection = (gameOverSelection + 1) % 2;

                updateUI();

            }

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

                if (gameOverSelection === 0) {

                    resetGame();

                    state = GAME\_STATE.PLAY;

                    updateUI();

                } else if (gameOverSelection === 1) {

                    state = GAME\_STATE.MENU;

                    updateUI();

                }

            }

        } else if (state === GAME\_STATE.LOADING) {

        }

    });

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

        input[e.code] = false;

    });

    document.addEventListener('mousedown', e => {

        if (e.button === 0) input['Mouse1'] = true;

        if (e.button === 2) input['Mouse2'] = true;

    });

    document.addEventListener('mouseup', e => {

        if (e.button === 0) input['Mouse1'] = false;

        if (e.button === 2) input['Mouse2'] = false;

    });

    document.addEventListener('mousemove', e => {

        if (state !== GAME\_STATE.PLAY) return;

        if (!pointerLocked && document.pointerLockElement === renderer.domElement) pointerLocked = true;

        if (pointerLocked) {

            cameraYaw -= e.movementX \* 0.008;

            cameraPitch -= e.movementY \* 0.006;

            cameraPitch = clamp(cameraPitch, -0.36, 0.46);

        }

    });

    renderer.domElement.addEventListener('click', () => {

        if (!pointerLocked) renderer.domElement.requestPointerLock();

    });

    document.addEventListener('pointerlockchange', () => {

        pointerLocked = document.pointerLockElement === renderer.domElement;

    });

    renderer.domElement.addEventListener('wheel', e => {

        input['Scroll'] = (input['Scroll'] || 0) + (e.deltaY > 0 ? 1 : -1);

        input['Scroll'] = clamp(input['Scroll'], -3, 4);

    });

}

function setup() {

    renderer = new THREE.WebGLRenderer({ antialias: true });

    renderer.setClearColor(0x181828);

    renderer.setSize(window.innerWidth, window.innerHeight);

    renderer.shadowMap.enabled = true;

    document.body.appendChild(renderer.domElement);

    scene = new THREE.Scene();

    camera = new THREE.PerspectiveCamera(65, window.innerWidth / window.innerHeight, 0.1, 400);

    clock = new THREE.Clock();

    let light = new THREE.DirectionalLight(0xffffff, 1.1);

    light.position.set(20, 32, 10);

    light.castShadow = true;

    scene.add(light);

    let ambient = new THREE.AmbientLight(0x8888bb, 0.38);

    scene.add(ambient);

    skybox = createSkybox();

    arena = createArena();

    player = createPlayer();

    createUI();

    updateUI();

    setupInput();

    loadingProgress = 0.1;

    setArenaSkybox();

    requestAnimationFrame(gameLoop);

}

setup();