```
/**
 * CM1005 - mid-term work
 */

// ===== ゲーム物理パラメータ =====
const SKATEBOARD_PHYSICS_GRAVITY = 2.2;
const SKATEBOARD_OLLIE_POWER = 95;
const SKATEBOARD_CRUISE_SPEED = 4.8;
const RIDER_COLLISION_RADIUS = 22;
const DECK_ELEVATION_HEIGHT = 11;

// ===== キャラクター描画パラメータ =====
const SPRITE_SCALE_FACTOR = 3;
const RIDER_STANCE_Y_ADJUST = -8;
const RIDER_AIRBORNE_Y_ADJUST = -8;

// ===== ゲームプレイ調整値 =====
const COIN_MAGNETISM_RANGE = 42;
const CANYON_WALL_THICKNESS = 6;

// ===== ビジュアルカラーパレット =====
const ATMOSPHERE_SKY = [95, 150, 250];
const TERRAIN_GRASS = [10, 145, 15];
const HUMAN_SKIN_TONE = [250, 208, 175];
const URBAN_JACKET = [15, 15, 180];
const STREET_DENIM = [10, 10, 120];
const EXPRESSION_LIPS = [240, 10, 10];
const VOID_CANYON_DEPTH = [25, 35, 75];
const VOID_CANYON_WALLS = [65, 75, 115];
const TREASURE_GOLD_OUTER = [250, 210, 5];
const TREASURE_GOLD_INNER = [255, 250, 10];
const DEBUG_OVERLAY_TEXT = [255, 255, 255];

// ===== 環境装飾カラーセット =====
const PEAK_SHADOW_TONE = [85, 85, 105];
const PEAK_HIGHLIGHT_TONE = [135, 135, 155];
const PEAK_SNOW_CAP = [235, 235, 245];
const PEAK_SNOW_SHADOW = [215, 215, 230];
const BARK_BROWN_BASE = [96, 62, 28];
const BARK_SHADOW_DARK = [65, 40, 15];
const FOLIAGE_GREEN_BASE = [29, 134, 29];
const FOLIAGE_SHADOW_DARK = [15, 95, 15];
const FOLIAGE_HIGHLIGHT_BRIGHT = [55, 175, 55];
const EVERGREEN_NEEDLE_COLOR = [5, 95, 5];
const VEGETATION_SHADOW = [5, 75, 5, 95];

// ===== 大気効果カラー =====
const CUMULUS_WHITE = [255, 255, 255, 195];
const CUMULUS_SHADOW = [225, 225, 235, 145];

// ===== 個人実装: スケートボード統合システム =====
const DECK_WOOD_COLOR = [134, 64, 14];
```

```
const DECK_GRIP_TAPE = [155, 77, 40];
const WHEEL_URETHANE = [45, 45, 45];
const TRUCK_METAL = [123, 123, 123];
// ===== 個人実装終了: スケートボード統合システム =====

// ===== ゲーム状態管理変数 =====
let skaterPositionX;
let skaterPositionY;
let groundLevelY;

// ===== プレイヤー入力状態 =====
let movingLeftward;
let movingRightward;
let inMidAir;
let fallingIntoVoid;

// ===== ゲームオブジェクト配列 =====
let voidCanyons;
let treasureCoins;

// ===== 背景環境オブジェクト =====
let mountainPeaks;
let forestTrees;
let treePositionsArray;  // 中間課題要件準拠
let atmosphericClouds;

// ===== 個人実装: カメラ制御システム =====
let cameraWorldX = 0;
const WORLD_BOUNDARY_WIDTH = 2048;
const CAMERA_FOLLOW_MARGIN = 380;
// ===== 個人実装終了: カメラ制御システム =====

// デバッグモード制御フラグ
let debugModeActive = false;

function setup()
{
        createCanvas(1024, 576);
        groundLevelY = height * 3/4;
        skaterPositionX = 200;
        skaterPositionY = groundLevelY - DECK_ELEVATION_HEIGHT;

        movingLeftward = false;
        movingRightward = false;
        inMidAir = false;
        fallingIntoVoid = false;

        // 危険な峡谷の配置
        voidCanyons = [
                {
                        worldX: 500,
                        gapWidth: 120
```

```
        },
        {
                worldX: 1300,
                gapWidth: 140
        }
];

// 収集可能な宝物コインの配置
treasureCoins = [
        {
                worldX: 150,
                worldY: groundLevelY,
                diameter: 30,
                collected: false
        },
        {
                worldX: 380,
                worldY: groundLevelY - 60,
                diameter: 30,
                collected: false
        },
        {
                worldX: 560,
                worldY: groundLevelY - 90,
                diameter: 30,
                collected: false
        },
        {
                worldX: 720,
                worldY: groundLevelY,
                diameter: 30,
                collected: false
        },
        {
                worldX: 1050,
                worldY: groundLevelY - 70,
                diameter: 30,
                collected: false
        },
        {
                worldX: 1250,
                worldY: groundLevelY - 50,
                diameter: 30,
                collected: false
        },
        {
                worldX: 1420,
                worldY: groundLevelY - 85,
                diameter: 30,
                collected: false
        },
        {
                worldX: 1600,
                worldY: groundLevelY,
```

```
                diameter: 30,
                collected: false
        }
    ];

    // 遠景の山々の初期化
    mountainPeaks = [
        {
            worldX: 100,
            baseSpan: 400,
            peakSpan: 200,
            elevation: 180
        },
        {
            worldX: 800,
            baseSpan: 500,
            peakSpan: 250,
            elevation: 220
        },
        {
            worldX: 1400,
            baseSpan: 350,
            peakSpan: 150,
            elevation: 160
        }
    ];

    // 中間課題要件: treePositionsArray配列でx座標を管理
    treePositionsArray = [80, 200, 320, 650, 800, 950, 1150,
1350, 1550];

    // 詳細な森林オブジェクトの初期化
    forestTrees = [
        {
            worldX: 80,
            baseY: groundLevelY,
            trunkWidth: 12,
            trunkHeight: 40,
            canopySize: 45,
            species: 'oak',
            scale: 'large'
        },
        {
            worldX: 200,
            baseY: groundLevelY,
            trunkWidth: 10,
            trunkHeight: 35,
            canopySize: 40,
            species: 'oak',
            scale: 'medium'
        },
        {
            worldX: 320,
            baseY: groundLevelY,
```

```
            trunkWidth: 8,
            trunkHeight: 30,
            canopySize: 35,
            species: 'oak',
            scale: 'medium'
        },
        {
            worldX: 650,
            baseY: groundLevelY,
            trunkWidth: 11,
            trunkHeight: 38,
            canopySize: 42,
            species: 'oak',
            scale: 'medium'
        },
        {
            worldX: 800,
            baseY: groundLevelY,
            trunkWidth: 10,
            trunkHeight: 35,
            canopySize: 40,
            species: 'oak',
            scale: 'medium'
        },
        {
            worldX: 950,
            baseY: groundLevelY,
            trunkWidth: 14,
            trunkHeight: 45,
            canopySize: 50,
            species: 'oak',
            scale: 'large'
        },
        {
            worldX: 1150,
            baseY: groundLevelY,
            trunkWidth: 13,
            trunkHeight: 42,
            canopySize: 48,
            species: 'oak',
            scale: 'large'
        },
        {
            worldX: 1350,
            baseY: groundLevelY,
            trunkWidth: 9,
            trunkHeight: 32,
            canopySize: 38,
            species: 'oak',
            scale: 'medium'
        },
        {
            worldX: 1550,
            baseY: groundLevelY,
```

```
                        trunkWidth: 11,
                        trunkHeight: 38,
                        canopySize: 42,
                        species: 'oak',
                        scale: 'medium'
                }
        ];

        // 大気中の雲の初期化
        atmosphericClouds = [
                {
                        worldX: 150,
                        worldY: 120,
                        size: 60,
                        type: 'medium'
                },
                {
                        worldX: 450,
                        worldY: 80,
                        size: 80,
                        type: 'large'
                },
                {
                        worldX: 750,
                        worldY: 100,
                        size: 50,
                        type: 'small'
                },
                {
                        worldX: 1100,
                        worldY: 90,
                        size: 70,
                        type: 'medium'
                },
                {
                        worldX: 1400,
                        worldY: 110,
                        size: 55,
                        type: 'small'
                },
                {
                        worldX: 1700,
                        worldY: 70,
                        size: 85,
                        type: 'large'
                }
        ];
}

function draw()
{
        drawBackground();

        // Following midterm instructions: push/translate/pop for
```

```
scrolling
        push();
        translate(-cameraWorldX, 0);

        drawClouds();
        drawMountains();
        drawTrees();
        renderVoidGaps();
        manageTreasureCoins();

        pop();

        // Character drawn outside translate to maintain correct
screen position
        renderSkateboardRider();
        processVoidCollision();
        updateCamera();
        updateSkaterPhysics();

        if(debugModeActive) {
                drawDebugInfo();
        }
}

function drawBackground() {
        background(ATMOSPHERE_SKY);
        noStroke();
        fill(TERRAIN_GRASS);
        rect(0, groundLevelY, WORLD_BOUNDARY_WIDTH, height -
groundLevelY);
}

function manageTreasureCoins() {
        for(let i = 0; i < treasureCoins.length; i++) {
                if(!treasureCoins[i].collected) {
                        drawCollectable(treasureCoins[i]);
                        checkCollectable(treasureCoins[i]);
                }
        }
}

// ===== START: Personal Code - Enhanced Collision Detection =====
function calculateDirectionalOffset() {
        if(movingLeftward) {
                return -RIDER_COLLISION_RADIUS;
        } else if(movingRightward) {
                return RIDER_COLLISION_RADIUS;
        }
        return 0;
}
// ===== END: Personal Code - Enhanced Collision Detection =====

function renderVoidGaps() {
        for(let i = 0; i < voidCanyons.length; i++) {
```

```
                drawCanyon(voidCanyons[i]);
        }
}

function drawCanyon(canyon) {
        // translate()を使用しているため、実際の画面座標を計算
        let screenX = canyon.worldX - cameraWorldX;

        // 画面外判定を修正
        if(screenX + canyon.gapWidth < -50 || screenX > width + 50)
{
                return;
        }

        noStroke();
        fill(VOID_CANYON_DEPTH);
        rect(canyon.worldX, groundLevelY, canyon.gapWidth, height -
groundLevelY);

        fill(VOID_CANYON_WALLS);
        rect(canyon.worldX, groundLevelY, CANYON_WALL_THICKNESS,
height - groundLevelY);
        rect(canyon.worldX + canyon.gapWidth -
CANYON_WALL_THICKNESS, groundLevelY, CANYON_WALL_THICKNESS, height -
groundLevelY);
}

function processVoidCollision() {
        let offsetX = calculateDirectionalOffset();

        // Check for falling into canyon
        for(let i = 0; i < voidCanyons.length; i++) {
                let canyon = voidCanyons[i];
                if(skaterPositionX + offsetX > canyon.worldX +
RIDER_COLLISION_RADIUS &&
                   skaterPositionX + offsetX < canyon.worldX +
canyon.gapWidth - RIDER_COLLISION_RADIUS &&
                   skaterPositionY >= groundLevelY -
DECK_ELEVATION_HEIGHT) {
                        fallingIntoVoid = true;
                        break;
                }
        }

        // Wall collision when falling
        if(fallingIntoVoid) {
                for(let i = 0; i < voidCanyons.length; i++) {
                        let canyon = voidCanyons[i];
                        if(skaterPositionX + offsetX >=
canyon.worldX - RIDER_COLLISION_RADIUS &&
                           skaterPositionX + offsetX <=
canyon.worldX + canyon.gapWidth + RIDER_COLLISION_RADIUS) {

adjustCharacterPositionInCanyon(canyon);
```

```
                        }
                }
        }
}

function adjustCharacterPositionInCanyon(canyon) {
        if(movingLeftward) {
                if(skaterPositionX < canyon.worldX +
RIDER_COLLISION_RADIUS) {
                        skaterPositionX = canyon.worldX +
RIDER_COLLISION_RADIUS;
                }
        } else if(movingRightward) {
                if(skaterPositionX > canyon.worldX + canyon.gapWidth
- RIDER_COLLISION_RADIUS) {
                        skaterPositionX = canyon.worldX +
canyon.gapWidth - RIDER_COLLISION_RADIUS;
                }
        } else {
                if(skaterPositionX < canyon.worldX +
RIDER_COLLISION_RADIUS/2) {
                        skaterPositionX = canyon.worldX +
RIDER_COLLISION_RADIUS/2;
                } else if(skaterPositionX > canyon.worldX +
canyon.gapWidth - RIDER_COLLISION_RADIUS/2) {
                        skaterPositionX = canyon.worldX +
canyon.gapWidth - RIDER_COLLISION_RADIUS/2;
                }
        }
}

function drawCollectable(collectable) {
        let screenX = collectable.worldX;
        let displayY = collectable.worldY - collectable.diameter/2;

        noStroke();
        fill(TREASURE_GOLD_OUTER);
        ellipse(screenX, displayY, collectable.diameter,
collectable.diameter);
        fill(TREASURE_GOLD_INNER);
        ellipse(screenX, displayY, collectable.diameter * 0.7,
collectable.diameter * 0.7);
}

function checkCollectable(collectable) {
        let offsetX = calculateDirectionalOffset();
        let displayY = collectable.worldY - collectable.diameter/2;
        let distance = dist(skaterPositionX + offsetX,
skaterPositionY, collectable.worldX, displayY);

        if(distance < COIN_MAGNETISM_RANGE) {
                collectable.collected = true;
        }
}
```

```
function renderSkateboardRider() {
        let direction = 'front';
        let yOffset = RIDER_STANCE_Y_ADJUST;
        let isJumping = false;

        if (movingLeftward && inMidAir) {
                direction = 'left';
                yOffset = RIDER_AIRBORNE_Y_ADJUST;
                isJumping = true;
        }
        else if (movingRightward && inMidAir) {
                direction = 'right';
                yOffset = RIDER_AIRBORNE_Y_ADJUST;
                isJumping = true;
        }
        else if (movingLeftward) {
                direction = 'left';
        }
        else if (movingRightward) {
                direction = 'right';
        }
        else if (inMidAir || fallingIntoVoid) {
                yOffset = RIDER_AIRBORNE_Y_ADJUST;
                isJumping = true;
        }

        let screenX = skaterPositionX - cameraWorldX;

        // Draw skateboard first (behind character)
        if (!fallingIntoVoid) {
                drawSkateboard(screenX, skaterPositionY);
        }

        drawCharacter(screenX, skaterPositionY, yOffset, direction,
isJumping);
}

function updateSkaterPhysics() {
        if(movingLeftward && skaterPositionX > 0) {
                skaterPositionX -= SKATEBOARD_CRUISE_SPEED;
        }

        if(movingRightward && skaterPositionX < WORLD_BOUNDARY_WIDTH
- RIDER_COLLISION_RADIUS) {
                skaterPositionX += SKATEBOARD_CRUISE_SPEED;
        }

        if(skaterPositionY < groundLevelY - DECK_ELEVATION_HEIGHT) {
                skaterPositionY += SKATEBOARD_PHYSICS_GRAVITY;
                inMidAir = true;
        } else {
                inMidAir = false;
        }
```

```
        if(fallingIntoVoid) {
                skaterPositionY += SKATEBOARD_PHYSICS_GRAVITY * 2;
        }
}

// ===== START: Personal Code - Camera System Implementation =====
function updateCamera() {
        // Following midterm instructions: cameraWorldX controls
camera position
        // Keep character centered on screen
        cameraWorldX = skaterPositionX - width/2;

        // Boundary limits
        let maxCameraX = WORLD_BOUNDARY_WIDTH - width;
        if(cameraWorldX > maxCameraX) {
                cameraWorldX = maxCameraX;
        }

        if(cameraWorldX < 0) {
                cameraWorldX = 0;
        }
}
// ===== END: Personal Code - Camera System Implementation =====

// ===== START: Personal Code - Debug System =====
function drawDebugInfo() {
        fill(DEBUG_TEXT_COLOR);
        noStroke();
        textSize(14);

        text("Character: " + Math.floor(skaterPositionX) + ", " +
Math.floor(skaterPositionY), 20, 20);
        text("Camera: " + Math.floor(cameraWorldX), 20, 40);
        text("Screen Character: " + Math.floor(skaterPositionX -
cameraWorldX), 20, 60);
        text("States: movingLeftward=" + movingLeftward + ",
movingRightward=" + movingRightward +
             ", inMidAir=" + inMidAir + ", fallingIntoVoid=" +
fallingIntoVoid, 20, 80);

        text("Canyons:", 20, 100);
        for(let i = 0; i < voidCanyons.length; i++) {
                text("  #" + (i+1) + ": " + voidCanyons[i].worldX +
" to " + (voidCanyons[i].worldX + voidCanyons[i].gapWidth), 20, 120
+ i*20);
        }

        text("Collectables:", 20, 160);
        for(let i = 0; i < treasureCoins.length; i++) {
                let displayY = treasureCoins[i].worldY -
treasureCoins[i].diameter/2;
                let distance = dist(skaterPositionX,
skaterPositionY, treasureCoins[i].worldX, displayY);
```

```
                text("  #" + (i+1) + ": pos=" +
Math.floor(treasureCoins[i].worldX) + "," +
                         Math.floor(displayY) + " | collected=" +
treasureCoins[i].collected +
                         " | distance=" + Math.floor(distance),
                         20, 180 + i*20);
        }

        text("Controls: A/D=move, W=jump, Z=toggle debug", 20,
height - 20);
}
// ===== END: Personal Code - Debug System =====

// ===== START: Personal Code - Background Decorations =====
function drawClouds() {
        for(let i = 0; i < atmosphericClouds.length; i++) {
                drawCloud(atmosphericClouds[i]);
        }
}

function drawCloud(cloud) {
        // translate()を使用しているため、実際の画面座標を計算
        let screenX = cloud.worldX - cameraWorldX;

        // 画面外判定を修正
        if(screenX + cloud.size * 2 < -100 || screenX > width + 100)
{
                return;
        }

        noStroke();

        // Generate circles based on cloud type
        let circles = [];
        if(cloud.type === 'small') {
                circles = [
                        {x: 0, y: 0, size: cloud.size},
                        {x: -cloud.size * 0.4, y: cloud.size * 0.1,
size: cloud.size * 0.7},
                        {x: cloud.size * 0.3, y: cloud.size * 0.15,
size: cloud.size * 0.8}
                ];
        } else if(cloud.type === 'medium') {
                circles = [
                        {x: 0, y: 0, size: cloud.size},
                        {x: -cloud.size * 0.5, y: cloud.size * 0.1,
size: cloud.size * 0.8},
                        {x: cloud.size * 0.4, y: cloud.size * 0.12,
size: cloud.size * 0.9},
                        {x: -cloud.size * 0.2, y: -cloud.size * 0.3,
size: cloud.size * 0.6},
                        {x: cloud.size * 0.15, y: -cloud.size *
0.25, size: cloud.size * 0.65}
```

```
                ];
        } else if(cloud.type === 'large') {
                circles = [
                        {x: 0, y: 0, size: cloud.size},
                        {x: -cloud.size * 0.6, y: cloud.size * 0.1,
size: cloud.size * 0.85},
                        {x: cloud.size * 0.5, y: cloud.size * 0.08,
size: cloud.size * 0.9},
                        {x: -cloud.size * 0.3, y: -cloud.size *
0.35, size: cloud.size * 0.7},
                        {x: cloud.size * 0.2, y: -cloud.size * 0.3,
size: cloud.size * 0.75},
                        {x: -cloud.size * 0.8, y: cloud.size * 0.25,
size: cloud.size * 0.6},
                        {x: cloud.size * 0.7, y: cloud.size * 0.2,
size: cloud.size * 0.65}
                ];
        }

        // Shadow layer
        fill(CUMULUS_SHADOW[0], CUMULUS_SHADOW[1],
CUMULUS_SHADOW[2], CUMULUS_SHADOW[3]);
        for(let circle of circles) {
                ellipse(cloud.worldX + circle.x + 3, cloud.worldY +
circle.y + 2, circle.size, circle.size);
        }

        // Main layer
        fill(CUMULUS_WHITE[0], CUMULUS_WHITE[1], CUMULUS_WHITE[2],
CUMULUS_WHITE[3]);
        for(let circle of circles) {
                ellipse(cloud.worldX + circle.x, cloud.worldY +
circle.y, circle.size, circle.size);
        }

        // Highlight layer
        fill(255, 255, 255, 120);
        for(let i = 0; i < circles.length; i++) {
                if(i < circles.length / 2) {
                        let circle = circles[i];
                        ellipse(cloud.worldX + circle.x -
circle.size * 0.2,
                                        cloud.worldY + circle.y -
circle.size * 0.15,
                                        circle.size * 0.4,
circle.size * 0.4);
                }
        }
}

function drawMountains() {
        for(let i = 0; i < mountainPeaks.length; i++) {
                drawMountain(mountainPeaks[i]);
        }
```

```
}

function drawMountain(mountain) {
        // translate()を使用しているため、実際の画面座標を計算
        let screenX = mountain.worldX - cameraWorldX;

        // 画面外判定を修正
        if(screenX + mountain.baseSpan < 0 || screenX > width) {
                return;
        }

        let peakY = groundLevelY - mountain.elevation;
        let baseY = groundLevelY;
        let leftPeakX = mountain.worldX + (mountain.baseSpan -
mountain.peakSpan) / 2;
        let rightPeakX = mountain.worldX + (mountain.baseSpan +
mountain.peakSpan) / 2;

        noStroke();

        // Shadow side (left)
        fill(PEAK_SHADOW_TONE);
        beginShape();
        vertex(mountain.worldX, baseY);
        vertex(leftPeakX, peakY);
        vertex(leftPeakX + mountain.peakSpan / 2, peakY);
        vertex(mountain.worldX + mountain.baseSpan / 2, baseY);
        endShape(CLOSE);

        // Light side (right)
        fill(PEAK_HIGHLIGHT_TONE);
        beginShape();
        vertex(mountain.worldX + mountain.baseSpan / 2, baseY);
        vertex(leftPeakX + mountain.peakSpan / 2, peakY);
        vertex(rightPeakX, peakY);
        vertex(mountain.worldX + mountain.baseSpan, baseY);
        endShape(CLOSE);

        // Ridge line
        stroke(PEAK_SHADOW_TONE);
        strokeWeight(2);
        line(leftPeakX, peakY, rightPeakX, peakY);
        noStroke();

        // Snow cap
        fill(PEAK_SNOW_CAP);
        let snowHeight = mountain.elevation * 0.35;
        beginShape();
        vertex(leftPeakX, peakY);
        vertex(rightPeakX, peakY);
        vertex(rightPeakX - mountain.peakSpan * 0.15, peakY +
snowHeight);
        vertex(leftPeakX + mountain.peakSpan * 0.15, peakY +
snowHeight);
```

```
        endShape(CLOSE);

        // Snow shadow
        fill(PEAK_SNOW_SHADOW);
        beginShape();
        vertex(leftPeakX, peakY);
        vertex(leftPeakX + mountain.peakSpan * 0.4, peakY);
        vertex(leftPeakX + mountain.peakSpan * 0.3, peakY +
snowHeight * 0.8);
        vertex(leftPeakX + mountain.peakSpan * 0.15, peakY +
snowHeight);
        endShape(CLOSE);
}
// ===== END: Personal Code - Background Decorations =====

function drawTrees() {
        for(let i = 0; i < forestTrees.length; i++) {
                drawTree(forestTrees[i]);
        }
}

function drawTree(tree) {
        // translate()を使用しているため、実際の画面座標を計算
        let screenX = tree.worldX - cameraWorldX;

        // 画面外判定を修正
        if(screenX + tree.canopySize < -50 || screenX > width + 50)
{
                return;
        }

        if(tree.species === 'oak') {
                drawOakTree(tree, tree.worldX);
        } else if(tree.species === 'pine') {
                drawPineTree(tree, tree.worldX);
        }
}

function drawOakTree(tree, screenX) {
        noStroke();

        // Ground shadow
        fill(VEGETATION_SHADOW[0], VEGETATION_SHADOW[1],
VEGETATION_SHADOW[2], VEGETATION_SHADOW[3]);
        ellipse(screenX + 5, tree.baseY, tree.canopySize * 0.8,
tree.canopySize * 0.3);

        // Trunk shadow
        fill(BARK_SHADOW_DARK);
        rect(screenX - tree.trunkWidth/2 + 2,
                 tree.baseY - tree.trunkHeight,
                 tree.trunkWidth * 0.4,
                 tree.trunkHeight);
```

```
        // Main trunk
        fill(BARK_BROWN_BASE);
        rect(screenX - tree.trunkWidth/2,
                 tree.baseY - tree.trunkHeight,
                 tree.trunkWidth,
                 tree.trunkHeight);

        // Branches
        stroke(BARK_BROWN_BASE);
        strokeWeight(3);
        line(screenX, tree.baseY - tree.trunkHeight * 0.7,
                 screenX - tree.canopySize * 0.3, tree.baseY -
tree.trunkHeight * 0.9);
        line(screenX, tree.baseY - tree.trunkHeight * 0.6,
                 screenX + tree.canopySize * 0.25, tree.baseY -
tree.trunkHeight * 0.8);
        noStroke();

        let leavesY = tree.baseY - tree.trunkHeight -
tree.canopySize/2;

        // Leaves shadow
        fill(FOLIAGE_SHADOW_DARK);
        ellipse(screenX + 3, leavesY + 3, tree.canopySize,
tree.canopySize);
        ellipse(screenX - tree.canopySize/3 + 2, leavesY +
tree.canopySize/4 + 2,
                        tree.canopySize * 0.7, tree.canopySize *
0.7);

        // Main leaves
        fill(FOLIAGE_GREEN_BASE);
        ellipse(screenX, leavesY, tree.canopySize, tree.canopySize);

        // Side leaves
        ellipse(screenX - tree.canopySize/3, leavesY +
tree.canopySize/4,
                        tree.canopySize * 0.7, tree.canopySize *
0.7);
        ellipse(screenX + tree.canopySize/3, leavesY +
tree.canopySize/4,
                        tree.canopySize * 0.7, tree.canopySize *
0.7);

        // Top leaves
        ellipse(screenX, leavesY - tree.canopySize/3,
                        tree.canopySize * 0.6, tree.canopySize *
0.6);

        // Highlights
        fill(FOLIAGE_HIGHLIGHT_BRIGHT);
        ellipse(screenX - tree.canopySize/6, leavesY -
tree.canopySize/6,
                        tree.canopySize * 0.4, tree.canopySize *
```

```
0.4);
        ellipse(screenX + tree.canopySize/5, leavesY +
tree.canopySize/8,
                        tree.canopySize * 0.3, tree.canopySize *
0.3);
}

function drawPineTree(tree, screenX) {
        noStroke();

        // Ground shadow
        fill(VEGETATION_SHADOW[0], VEGETATION_SHADOW[1],
VEGETATION_SHADOW[2], VEGETATION_SHADOW[3]);
        ellipse(screenX + 3, tree.baseY, tree.canopySize * 0.6,
tree.canopySize * 0.2);

        // Trunk shadow
        fill(BARK_SHADOW_DARK);
        rect(screenX - tree.trunkWidth/2 + 1,
                 tree.baseY - tree.trunkHeight,
                 tree.trunkWidth * 0.4,
                 tree.trunkHeight);

        // Main trunk
        fill(BARK_BROWN_BASE);
        rect(screenX - tree.trunkWidth/2,
                 tree.baseY - tree.trunkHeight,
                 tree.trunkWidth,
                 tree.trunkHeight);

        // Pine layers - triangular sections
        fill(EVERGREEN_NEEDLE_COLOR);
        let layerHeight = tree.trunkHeight / 3;
        let currentY = tree.baseY - tree.trunkHeight;
        let currentSize = tree.canopySize;

        for(let i = 0; i < 4; i++) {
                // Shadow layer
                fill(FOLIAGE_SHADOW_DARK);
                triangle(screenX + 2, currentY + 2,
                                 screenX - currentSize/2 + 2,
currentY - layerHeight + 2,
                                 screenX + currentSize/2 + 2,
currentY - layerHeight + 2);

                // Main layer
                fill(EVERGREEN_NEEDLE_COLOR);
                triangle(screenX, currentY,
                                 screenX - currentSize/2, currentY -
layerHeight,
                                 screenX + currentSize/2, currentY -
layerHeight);

                // Highlight
```

```
                fill(FOLIAGE_HIGHLIGHT_BRIGHT);
                triangle(screenX - 2, currentY - layerHeight * 0.3,
                                screenX - currentSize/4, currentY -
layerHeight * 0.7,
                                screenX + currentSize/6, currentY -
layerHeight * 0.5);

                currentY -= layerHeight * 0.7;
                currentSize *= 0.8;
        }
}

function keyControl(event) {
        console.log(event.type + ": " + key + " (" + keyCode + ")");

        // ===== START: Personal Code - Debug Toggle =====
        if(key === 'z' && event.type === 'keydown') {
                debugModeActive = !debugModeActive;
                return;
        }
        // ===== END: Personal Code - Debug Toggle =====

        const isKeyPressed = event.type === 'keydown';

        // Prevent control when character is falling into canyon
        if(fallingIntoVoid) {
                return;
        }

        switch(key) {
                case 'a':
                        movingLeftward = isKeyPressed;
                        break;
                case 'd':
                        movingRightward = isKeyPressed;
                        break;
                case 'w':
                        // Prevent double jumping
                        if(isKeyPressed && !inMidAir && !
fallingIntoVoid) {
                                skaterPositionY -=
SKATEBOARD_OLLIE_POWER;
                        }
                        break;
        }
}


function keyPressed() {
        keyControl({ type: 'keydown' });
}

function keyReleased() {
        keyControl({ type: 'keyup' });
```

```
}

function drawCharacter(x, y, yOffset, direction, isJumping) {
        drawHair(x, y, yOffset, direction);
        drawFace(x, y, yOffset, direction);
        drawClothes(x, y, yOffset, direction);
        drawLegs(x, y, yOffset, direction);
}

function drawHair(x, y, yOffset, direction) {
        fill(0);

        if (direction === 'front') {
                rect(x - 4*SPRITE_SCALE_FACTOR, y -
12*SPRITE_SCALE_FACTOR + yOffset, 8*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
                rect(x - 5*SPRITE_SCALE_FACTOR, y -
11*SPRITE_SCALE_FACTOR + yOffset, 10*SPRITE_SCALE_FACTOR,
3*SPRITE_SCALE_FACTOR);
                rect(x - 6*SPRITE_SCALE_FACTOR, y -
10*SPRITE_SCALE_FACTOR + yOffset, 12*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'left') {
                rect(x - 6*SPRITE_SCALE_FACTOR, y -
13*SPRITE_SCALE_FACTOR + yOffset, 8*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
                rect(x - 7*SPRITE_SCALE_FACTOR, y -
12*SPRITE_SCALE_FACTOR + yOffset, 9*SPRITE_SCALE_FACTOR,
3*SPRITE_SCALE_FACTOR);
                rect(x - 8*SPRITE_SCALE_FACTOR, y -
11*SPRITE_SCALE_FACTOR + yOffset, 10*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'right') {
                rect(x - 2*SPRITE_SCALE_FACTOR, y -
13*SPRITE_SCALE_FACTOR + yOffset, 8*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
                rect(x - 2*SPRITE_SCALE_FACTOR, y -
12*SPRITE_SCALE_FACTOR + yOffset, 9*SPRITE_SCALE_FACTOR,
3*SPRITE_SCALE_FACTOR);
                rect(x - 2*SPRITE_SCALE_FACTOR, y -
11*SPRITE_SCALE_FACTOR + yOffset, 10*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
        }
}

function drawFace(x, y, yOffset, direction) {
        fill(HUMAN_SKIN_TONE);

        if (direction === 'front') {
                rect(x - 4*SPRITE_SCALE_FACTOR, y -
8*SPRITE_SCALE_FACTOR + yOffset, 8*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
```

```
                rect(x - 3*SPRITE_SCALE_FACTOR, y -
4*SPRITE_SCALE_FACTOR + yOffset, 6*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                // Eyes
                fill(255);
                rect(x - 3*SPRITE_SCALE_FACTOR, y -
7*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);
                rect(x + 1*SPRITE_SCALE_FACTOR, y -
7*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(0);
                rect(x - 2*SPRITE_SCALE_FACTOR, y -
7*SPRITE_SCALE_FACTOR + yOffset, 1*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);
                rect(x + 1*SPRITE_SCALE_FACTOR, y -
7*SPRITE_SCALE_FACTOR + yOffset, 1*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);

                // Mouth
                fill(EXPRESSION_LIPS);
                rect(x - 1*SPRITE_SCALE_FACTOR, y -
2*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'left') {
                rect(x - 7*SPRITE_SCALE_FACTOR, y -
9*SPRITE_SCALE_FACTOR + yOffset, 7*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);
                rect(x - 6*SPRITE_SCALE_FACTOR, y -
5*SPRITE_SCALE_FACTOR + yOffset, 5*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(255);
                rect(x - 6*SPRITE_SCALE_FACTOR, y -
8*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(0);
                rect(x - 5*SPRITE_SCALE_FACTOR, y -
8*SPRITE_SCALE_FACTOR + yOffset, 1*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);

                fill(EXPRESSION_LIPS);
                rect(x - 4*SPRITE_SCALE_FACTOR, y -
3*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'right') {
                rect(x, y - 9*SPRITE_SCALE_FACTOR + yOffset,
7*SPRITE_SCALE_FACTOR, 4*SPRITE_SCALE_FACTOR);
                rect(x + 1*SPRITE_SCALE_FACTOR, y -
```

```
5*SPRITE_SCALE_FACTOR + yOffset, 5*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(255);
                rect(x + 4*SPRITE_SCALE_FACTOR, y -
8*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(0);
                rect(x + 4*SPRITE_SCALE_FACTOR, y -
8*SPRITE_SCALE_FACTOR + yOffset, 1*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);

                fill(EXPRESSION_LIPS);
                rect(x + 2*SPRITE_SCALE_FACTOR, y -
3*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
1*SPRITE_SCALE_FACTOR);
        }
}

function drawClothes(x, y, yOffset, direction) {
        fill(URBAN_JACKET);

        if (direction === 'front') {
                rect(x - 3*SPRITE_SCALE_FACTOR, y + yOffset,
6*SPRITE_SCALE_FACTOR, 4*SPRITE_SCALE_FACTOR);

                fill(255);
                rect(x - 1*SPRITE_SCALE_FACTOR, y + yOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);

                fill(STREET_DENIM);
                rect(x - 3*SPRITE_SCALE_FACTOR, y +
4*SPRITE_SCALE_FACTOR + yOffset, 6*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'left') {
                rect(x - 6*SPRITE_SCALE_FACTOR, y -
1*SPRITE_SCALE_FACTOR + yOffset, 6*SPRITE_SCALE_FACTOR,
4*SPRITE_SCALE_FACTOR);

                fill(255);
                rect(x - 4*SPRITE_SCALE_FACTOR, y -
1*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(STREET_DENIM);
                rect(x - 6*SPRITE_SCALE_FACTOR, y +
3*SPRITE_SCALE_FACTOR + yOffset, 6*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'right') {
                rect(x, y - 1*SPRITE_SCALE_FACTOR + yOffset,
6*SPRITE_SCALE_FACTOR, 4*SPRITE_SCALE_FACTOR);
```

```
                fill(255);
                rect(x + 2*SPRITE_SCALE_FACTOR, y -
1*SPRITE_SCALE_FACTOR + yOffset, 2*SPRITE_SCALE_FACTOR,
2*SPRITE_SCALE_FACTOR);

                fill(STREET_DENIM);
                rect(x, y + 3*SPRITE_SCALE_FACTOR + yOffset,
6*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
        }
}

function drawLegs(x, y, yOffset, direction) {
        fill(0);

        const legOffset = y + 6*SPRITE_SCALE_FACTOR + yOffset;

        if (direction === 'front') {
                rect(x - 3*SPRITE_SCALE_FACTOR, legOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
                rect(x + 1*SPRITE_SCALE_FACTOR, legOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'left') {
                rect(x - 6*SPRITE_SCALE_FACTOR, legOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
                rect(x - 1*SPRITE_SCALE_FACTOR, legOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
        }
        else if (direction === 'right') {
                rect(x - 1*SPRITE_SCALE_FACTOR, legOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
                rect(x + 4*SPRITE_SCALE_FACTOR, legOffset,
2*SPRITE_SCALE_FACTOR, 2*SPRITE_SCALE_FACTOR);
        }
}

// ===== START: Personal Code - Skateboard Implementation =====
function drawSkateboard(x, y) {
        noStroke();

        // Skateboard dimensions
        const boardWidth = 40;
        const boardHeight = 8;
        const wheelSize = 6;
        const truckWidth = 3;

        // Position skateboard so wheels touch the ground
        let boardY = y + DECK_ELEVATION_HEIGHT;

        // Skateboard deck shadow
        fill(DECK_WOOD_COLOR[0] - 30, DECK_WOOD_COLOR[1] - 20,
DECK_WOOD_COLOR[2] - 10);
        rect(x - boardWidth/2 + 2, boardY + 2, boardWidth,
```

```
boardHeight, 3);

        // Main skateboard deck
        fill(DECK_WOOD_COLOR);
        rect(x - boardWidth/2, boardY, boardWidth, boardHeight, 3);

        // Deck top surface (lighter)
        fill(DECK_GRIP_TAPE);
        rect(x - boardWidth/2 + 1, boardY + 1, boardWidth - 2,
boardHeight - 3, 2);

        // Trucks (metal parts)
        fill(TRUCK_METAL);
        rect(x - boardWidth/3, boardY + boardHeight - 1, truckWidth,
4);
        rect(x + boardWidth/3 - truckWidth, boardY + boardHeight -
1, truckWidth, 4);

        // Wheels
        fill(WHEEL_URETHANE);
        ellipse(x - boardWidth/3, boardY + boardHeight + 2,
wheelSize, wheelSize);
        ellipse(x + boardWidth/3, boardY + boardHeight + 2,
wheelSize, wheelSize);
        ellipse(x - boardWidth/3 + truckWidth, boardY + boardHeight
+ 2, wheelSize, wheelSize);
        ellipse(x + boardWidth/3 - truckWidth, boardY + boardHeight
+ 2, wheelSize, wheelSize);

        // Wheel highlights
        fill(80, 80, 80);
        ellipse(x - boardWidth/3 - 1, boardY + boardHeight + 1,
wheelSize/2, wheelSize/2);
        ellipse(x + boardWidth/3 - 1, boardY + boardHeight + 1,
wheelSize/2, wheelSize/2);
        ellipse(x - boardWidth/3 + truckWidth - 1, boardY +
boardHeight + 1, wheelSize/2, wheelSize/2);
        ellipse(x + boardWidth/3 - truckWidth - 1, boardY +
boardHeight + 1, wheelSize/2, wheelSize/2);
}
// ===== END: Personal Code - Skateboard Implementation =====
```