```
//The Game Project

var floorPos_y;

var gameChar_x;
var gameChar_y;

var canyon;
var collectable;

var mountains;
var trees_x;
var clouds;
var dinosaurs;
var dinosaurColors;


// Game character interaction variables
var isLeft;
var isRight;
var isFalling;
var isPlummeting;

// Game state variable for death
var isDead;


function setup() {
    createCanvas(1024, 576);
    floorPos_y = 432;

    gameChar_x = 700;
    gameChar_y = floorPos_y;
    // Game Character: The anchor point for the character is gameChar_x
(horizontal center) and gameChar_y (bottom of the feet).

    // Initialize arrays with objects

    trees_x = [
        { pos_x: width / 6, pos_y: floorPos_y - 50 },
        { pos_x: width / 4, pos_y: floorPos_y - 50 },
        { pos_x: width / 8, pos_y: floorPos_y - 50 },
        { pos_x: width / 1.5, pos_y: floorPos_y - 50 },
        { pos_x: width / 1.2, pos_y: floorPos_y - 50 }
    ];
    // Tree: The anchor point for a tree object is its base (center of
the trunk at the bottom).

    clouds = [
        { pos_x: random(10, width), pos_y: random(20, 100), size:
random(50, 80) },
        { pos_x: random(10, width), pos_y: random(100, 200), size:
random(50, 80) },
        { pos_x: random(10, width), pos_y: random(200, 250), size:
random(50, 80) },
        { pos_x: random(10, width), pos_y: random(50, 150), size:
random(60, 90) }
    ];
```

```
    // Cloud: The anchor point for a cloud object is the center of its
main (largest) ellipse.

    // Define a selection of dinosaur colors
    dinosaurColors = [
        [70, 100, 70],   // Dark green
        [100, 70, 70],   // Brownish red
        [70, 70, 100],   // Bluish purple
        [120, 90, 50],   // Earthy brown
        [60, 80, 60]     // Muted green
    ];

    // Initialize dinosaurs
    dinosaurs = [
        { pos_x: random(width, width * 1.5), pos_y: random(50, 150),
size: random(80, 120), speed: random(1, 2), color: random(dinosaurColors)
},
        { pos_x: random(width * 1.5, width * 2), pos_y: random(100, 200),
size: random(70, 110), speed: random(0.8, 1.5), color:
random(dinosaurColors) },
        { pos_x: random(width * 2, width * 2.5), pos_y: random(150, 250),
size: random(90, 130), speed: random(1.2, 2.5), color:
random(dinosaurColors) }
    ];
    // Dinosaur: The anchor point for drawing is the center of its main
body (0,0) after translation.

    // Corrected mountain y-positions to be level with the ground
    mountains = [
        { pos_x: 300, pos_y: floorPos_y - (250 / 2), height: 250, width:
100 },
        { pos_x: 150, pos_y: floorPos_y - (150 / 2), height: 150, width:
50 },
        { pos_x: 50, pos_y: floorPos_y - (350 / 2), height: 350, width:
200 }
    ];
    // Mountain: The anchor point for a mountain object is the horizontal
center of its base.

    // Game character interaction setup
    isLeft = false;
    isRight = false;
    isFalling = false;
    isPlummeting = false;

    // Define the canyon
    canyon = { x_pos: 450, width: 100, y_pos: floorPos_y };
    // Canyon: The anchor point for the canyon is its top-left corner
(x_pos, y_pos).

    // Collectable - Position adjusted so the drumstick sits on the
ground
    collectable = { x_pos: 200, y_pos: floorPos_y - 20, size: 10,
isFound: false };
    // Collectable: The anchor point for the collectable (drumstick) is
approximately the horizontal center and near the top of its main body.

    // Initialize death state
```

```
    isDead = false;
}


function draw() {
    background(100, 155, 255); // Fill the sky blue

    // Camera Follow Logic
    push(); // Save the current transformation state
    // Shift the canvas by an amount that keeps the character visually
centered
    translate(width / 2 - gameChar_x, 0);

    drawGround(); // Draw the ground

    // Draw mountains using a loop
    for (let i = 0; i < mountains.length; i++) {
        drawMountain(mountains[i]);
    }

    // Animate and draw clouds using a loop
    for (let i = 0; i < clouds.length; i++) {
        animateCloud(clouds[i]);
        drawCloud(clouds[i]);
    }

    // Animate and draw dinosaurs
    for (let i = 0; i < dinosaurs.length; i++) {
        animateDinosaur(dinosaurs[i]);
        drawDinosaur(dinosaurs[i]);
    }

    // Draw trees using a loop
    for (let i = 0; i < trees_x.length; i++) {
        drawTree(trees_x[i]);
    }

    // Draw the canyon (pit)
    drawCanyon(canyon);

    // Character Update Logic (movement and gravity)
    // Only update character if not dead
    if (!isDead) {
        // Apply gravity
        gameChar_y += 2;

        // Determine if the character is over a canyon
        var isOverCanyon = (gameChar_x > canyon.x_pos && gameChar_x <
canyon.x_pos + canyon.width);

        // If the character is below or at floor level AND NOT over a
canyon,
        // then they are on solid ground and should stop falling.
        if (gameChar_y >= floorPos_y && !isOverCanyon) {
            gameChar_y = floorPos_y; // Clamp to floor
            isFalling = false;
        } else {
```

```
            // If they are above the floor, or over a canyon, they are
falling.
            isFalling = true;
        }

        // Handle character horizontal movement
        if (isLeft) {
            gameChar_x -= 5;
        } else if (isRight) {
            gameChar_x += 5;
        }

        // Check if character fell into the canyon
        checkCanyon(canyon);

    } else {
        // If dead, ensure character falls indefinitely (plummeting)
        gameChar_y += 5; // Faster fall when dead
        isPlummeting = true;
        isFalling = true;
    }


    // Character Drawing (After position is updated)
    drawCharacter();

    // Collectable Logic (After character update)
    checkIfGameCharInCollectableRange();
    drawCollectable();

    pop(); // Restore the previous transformation state

    // "Game Over" Text
    // This text should be drawn after pop() so it's not affected by
camera translation
    if (isDead) {
        fill(255, 0, 0); // Red color for game over text
        textSize(60);
        textAlign(CENTER, CENTER);
        text("GAME OVER!", width / 2, height / 2); // Display in the
center of the screen
        textSize(20);
        text("Press space to restart", width / 2, height / 2 + 50);
    }
}


// Character Drawing Helper Functions

function drawCharacterBody() {
    fill(255, 128, 0); // Orange body
    // Game Character Body: Top-left corner is gameChar_x - 12,
gameChar_y - 60.
    rect(gameChar_x - 12, gameChar_y - 60, 25, 50);
}

function drawCharacterHead() {
    fill(255, 204, 153); // Skin tone head
```

```
    // Game Character Head: Center is gameChar_x, gameChar_y - 69.
    ellipse(gameChar_x, gameChar_y - 69, 17, 17);
}

function drawPolkaDots() {
    fill(0, 0, 0); // Black polka dots
    // Polka Dots: Each ellipse's center is relative to gameChar_x,
gameChar_y.
    ellipse(gameChar_x - 7, gameChar_y - 50, 5, 5);
    ellipse(gameChar_x + 7, gameChar_y - 45, 5, 5);
    ellipse(gameChar_x - 5, gameChar_y - 38, 5, 5);
    ellipse(gameChar_x + 5, gameChar_y - 30, 5, 5);
    ellipse(gameChar_x - 8, gameChar_y - 25, 5, 5);
}


function drawCharacterArms(leftArmOffsetX, leftArmOffsetY,
rightArmOffsetX, rightArmOffsetY) {
    fill(255, 204, 153); // Skin tone arms
    // Game Character Arms: Each arm's top-left corner is offset from
gameChar_x, gameChar_y.
    rect(gameChar_x + leftArmOffsetX, gameChar_y + leftArmOffsetY, 10,
30); // Left arm
    rect(gameChar_x + rightArmOffsetX, gameChar_y + rightArmOffsetY, 10,
30); // Right arm
}

function drawCharacterFeet() {
    fill(255, 204, 153); // Skin tone feet
    // Game Character Feet: The middle vertex of the triangle is at
gameChar_x, gameChar_y.
    triangle(gameChar_x - 12, gameChar_y - 10, gameChar_x, gameChar_y,
gameChar_x + 12, gameChar_y - 10);
}


function drawCharacter() {
    drawCharacterHead(); // Always draw head

    // Reverted: Body and polka dots are now always drawn, even when
falling.
    drawCharacterBody();
    drawPolkaDots();


    drawCharacterFeet(); // Always draw feet

    // Adjust character arms based on state
    if (isLeft && isFalling) {
        // Adjusted right arm Y-offset to prevent it from drawing below
the feet
        drawCharacterArms(-1, -80, 0, -60);
    }
    else if (isRight && isFalling) {
        // Adjusted right arm Y-offset to prevent it from drawing below
the feet
        drawCharacterArms(-8, -80, 0, -60);
    }
```

```
    else if (isLeft) { // Moving left - arms for walking
        drawCharacterArms(-22, -60, 13, -60);
    }
    else if (isRight) { // Moving right - arms for walking
        drawCharacterArms(-22, -60, 13, -60);
    }
    else if (isFalling || isPlummeting) { // General falling or
plummeting state
        drawCharacterArms(-22, -60, 13, -60);
    }
    else { // Default pose (standing)
        drawCharacterArms(-22, -60, 13, -60);
    }
}


// Collectable Functions

function checkIfGameCharInCollectableRange() {
    var d = dist(gameChar_x, gameChar_y, collectable.x_pos,
collectable.y_pos);
    if (d < 30) {
        collectable.isFound = true;
    }
}

function drawCollectable() {
    if (!collectable.isFound) {
        // Drumstick
        fill(255, 236, 214);
        // The main rect starts at collectable.x_pos - 5,
collectable.y_pos - 20.
        rect(collectable.x_pos - 5, collectable.y_pos - 20, 10, 40);
        ellipse(collectable.x_pos - 5, collectable.y_pos - 20, 10);
        ellipse(collectable.x_pos + 3, collectable.y_pos - 20, 10);

        // Meat
        fill(205, 69, 0);
        ellipse(collectable.x_pos, collectable.y_pos + 10, 35, 50);
    }
}


// Canyon Functions

function drawCanyon(t_canyon) {
    noStroke();
    fill(100, 155, 255); // Dark purple/blue for the pit
    // Canyon: The anchor point for the canyon is its top-left corner
(t_canyon.x_pos, t_canyon.y_pos).
    rect(t_canyon.x_pos, t_canyon.y_pos, t_canyon.width, height -
t_canyon.y_pos);
}

function checkCanyon(t_canyon) {
    // Check if character's horizontal position is within the canyon's x-
range
```

```
    // and if the character has fallen significantly below the
floorPos_y.
    // '10' is a small threshold to ensure they are truly "in" the pit
and not just past floorPos_y from a jump.
    if (gameChar_x > t_canyon.x_pos && gameChar_x < t_canyon.x_pos +
t_canyon.width) {
        if (gameChar_y > floorPos_y + 10) {
            isDead = true;
        }
    }
}


// Background Element Drawing and Animation Functions

function animateCloud(t_cloud) { // Animates a single cloud object
    t_cloud.pos_x++; // Move cloud right
    // Reset cloud if it moves off screen to create continuous movement
    if (t_cloud.pos_x > width + t_cloud.size * 1.2) {
        t_cloud.pos_x = -t_cloud.size * 1.2; // Start from left of screen
    }
}

function drawClouds() {
    // Use a loop to draw all clouds
    for (let i = 0; i < clouds.length; i++) {
        drawCloud(clouds[i]);
    }
}

function drawCloud(t_cloud) {
    fill(255); // White clouds
    // Cloud: The anchor point for drawing is t_cloud.pos_x,
t_cloud.pos_y (center of the main ellipse).
    ellipse(t_cloud.pos_x, t_cloud.pos_y, t_cloud.size * 1.2,
t_cloud.size * 1.2);
    ellipse(t_cloud.pos_x - 40, t_cloud.pos_y, t_cloud.size,
t_cloud.size);
    ellipse(t_cloud.pos_x + 40, t_cloud.pos_y, t_cloud.size,
t_cloud.size);
}

// Dinosaur animation function
function animateDinosaur(t_dinosaur) {
    t_dinosaur.pos_x -= t_dinosaur.speed; // Move dinosaur left
    // Reset dinosaur if it moves off screen to create continuous
movement
    if (t_dinosaur.pos_x < -t_dinosaur.size * 1.5) { // Adjusted reset
point
        t_dinosaur.pos_x = width + t_dinosaur.size * 1.5; // Start from
right of screen
        t_dinosaur.pos_y = random(50, 250); // Randomize vertical
position
        t_dinosaur.speed = random(1, 2.5); // Randomize speed
        t_dinosaur.color = random(dinosaurColors); // Assign a new random
color when reset
    }
}
```

```
// Dinosaur drawing function
function drawDinosaur(t_dinosaur) {
    push(); // Isolate transformation for each dinosaur
    translate(t_dinosaur.pos_x, t_dinosaur.pos_y);
    scale(t_dinosaur.size / 100); // Scale based on the 'size' property

    fill(t_dinosaur.color[0], t_dinosaur.color[1], t_dinosaur.color[2]);
// Use the dinosaur's assigned color
    noStroke();

    // Body
    ellipse(0, 0, 70, 30); // Main body

    // Neck
    ellipse(-30, -20, 20, 25);

    // Head
    ellipse(-45, -30, 20, 15);

    // Beak/Mouth (simple triangle)
    triangle(-55, -30, -45, -35, -45, -25);

    // Wing (simple triangle)
    triangle(10, -10, 50, -35, 30, -5);

    // Tail
    triangle(35, 0, 50, 10, 60, 5);

    pop(); // Restore previous transformation state
}


function drawTrees() {
    // Use a loop to draw all trees
    for (let i = 0; i < trees.length; i++) {
        drawTree(trees_x[i]);
    }
}

function drawTree(t_tree) {
    noStroke();
    fill(198, 117, 20); // Brown
    rectMode(CENTER);
    // Tree Trunk: The anchor point for the tree trunk drawing is
t_tree.pos_x, t_tree.pos_y (when rectMode is CENTER).
    rect(t_tree.pos_x, t_tree.pos_y, 40, 100);
    rectMode(CORNER); // Set back to default mode

    fill(66, 105, 47); // Green
    // Tree Leaves: The anchor points for the triangle (leaves) are its
three vertices, defined relative to t_tree.pos_x, t_tree.pos_y.
    var x1 = t_tree.pos_x - 80;
    var y1 = t_tree.pos_y - 50;
    var x2 = t_tree.pos_x;
    var y2 = t_tree.pos_y - 150;
    var x3 = t_tree.pos_x + 80;
    var y3 = t_tree.pos_y - 50;
```

```
    triangle(x1, y1, x2, y2, x3, y3);
}


function drawMountains() {
    // Use a loop to draw all mountains
    for (let i = 0; i < mountains.length; i++) {
        drawMountain(mountains[i]);
    }
}

function drawMountain(t_mountain) {
    // Mountain: The anchor point for drawing the triangle is defined by
its three vertices,
    // calculated from t_mountain.pos_x (horizontal center of base) and
t_mountain.pos_y (vertical center of height).
    var x1 = t_mountain.pos_x - t_mountain.width / 2;
    var y1 = t_mountain.pos_y + t_mountain.height / 2;
    var x2 = t_mountain.pos_x;
    var y2 = t_mountain.pos_y - t_mountain.height / 2;
    var x3 = t_mountain.pos_x + t_mountain.width / 2;
    var y3 = t_mountain.pos_y + t_mountain.height / 2;
    fill(82, 92, 95); // Grey
    triangle(x1, y1, x2, y2, x3, y3);
}

function drawGround() {
    noStroke();
    fill(101, 35, 2); // Brownish ground
    // Ground: The anchor point is its top-left corner (0, floorPos_y).
    rect(0, floorPos_y, width, height - floorPos_y);
}


// Keyboard Interaction Functions

function keyPressed() {
    // Only allow movement and jump if not dead
    if (!isDead) {
        if (keyCode == 37) { // Left arrow
            console.log("left arrow pressed");
            isLeft = true;
        }
        else if (keyCode == 39) { // Right arrow
            console.log("right arrow pressed");
            isRight = true;
        }
        else if (keyCode == 38) { // Up arrow
            if (gameChar_y >= floorPos_y) { // Only jump if on the ground
                console.log("up arrow pressed");
                gameChar_y -= 100; // Increased jump height slightly
            }
        }
    }
    // Allow restart with spacebar if dead
    else if (keyCode == 32) { // Spacebar
        setup(); // Call setup to reset the game state
    }
```

```
}

function keyReleased() {
    if (keyCode == 37) { // Left arrow
        console.log("left arrow released");
        isLeft = false;
    }
    if (keyCode == 39) { // Right arrow
        console.log("right arrow released");
        isRight = false;
    }
}
```