```
// global game constants
const middleLane = 0;
const particleCount = 20;
const initRollingSpeed = 0.005;

let sceneWidth;
let sceneHeight;
let camera;
let scene;
let renderer;
let rollingGroundSphere;
let heroSphere;
let rollingSpeed = initRollingSpeed;
let sphericalHelper;
let bounceValue = 0.1;
let currentLane;
let treeClock;
let levelClock;
let jumping;
let treesInPath;
let treesPool;
let scoreText;
let score;
let hasCollided;
let gameOverFlag;
let distanceCounter;
let distanceMeter;
let isPaused;
let globalRenderID;
let levelCounter;
let scheduler;
let w orld;
let hero;
let sun;
let explosion;

init();

function init() {
// set up the scene
createScene();

// instructions to play game
gameInstructions();
}

function createScene() {
// variables
distanceCounter = 0;
isPaused = false;
gameOverFlag = false;
hasCollided = false;
score = 0;
levelCounter = 1;
currentLane = middleLane;
jumping = false;
```

```
treesInPath = [];
treesPool = [];

// objects
treeClock = new THREE.Clock();
treeClock.start();
levelClock = new THREE.Clock();
levelClock.start();
scheduler = new Scheduler();
sphericalHelper = new THREE.Spherical();
scene = new THREE.Scene(); // the 3d scene
scene.fog = new THREE.FogExp2(0xf0fff0, 0.14);
world = new World(scene);
hero = new Hero(scene);
sun = new Light(scene);
explosion = new Explosion(scene, particleCount);
rollingGroundSphere = world.body;
heroSphere = hero.body;

// perspective camera
sceneWidth = window.innerWidth;
sceneHeight = Math.round(window.innerHeight * 0.99);
camera = new THREE.PerspectiveCamera(60, sceneWidth / sceneHeight, 0.1, 1000);
renderer = new THREE.WebGLRenderer({ alpha: true }); // renderer with transparent backdrop
renderer.setClearColor(0xfffafa, 1);
renderer.shadowMap.enabled = true; // enable shadow
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
renderer.setSize(sceneWidth, sceneHeight);

const dom = document.getElementById('TutContainer');
dom.appendChild(renderer.domElement);

createTreesPool();
addWorldTrees();

camera.position.z = 6.5;
camera.position.y = 3.5;
// helper to rotate around in scene
const orbitControl = new THREE.OrbitControls(camera, renderer.domElement);
orbitControl.addEventListener('change', render);
orbitControl.enableKeys = false;
orbitControl.enablePan = true;
orbitControl.enableZoom = true;
orbitControl.minPolarAngle = 1.1;
orbitControl.maxPolarAngle = 1.1;
orbitControl.minAzimuthAngle = -0.2;
orbitControl.maxAzimuthAngle = 0.2;

window.addEventListener('resize', onWindowResize, false); // resize callback

document.onkeydown = handleKeyDown;

scoreText = document.createElement('div');
scoreText.setAttribute('id', 'scoreBoard');
scoreText.innerHTML = score.toString();
document.body.appendChild(scoreText);
```

```
const infoText = document.createElement('div');
infoText.setAttribute('id', 'infoBoard');
infoText.innerHTML = 'UP - Jump, Left/Right - Move
"M" un/mute';
document.body.appendChild(infoText);

distanceMeter = document.createElement('div');
distanceMeter.setAttribute('id', 'distanceBoard');
distanceMeter.innerHTML = '0m';
document.body.appendChild(distanceMeter);
}

function gameInstructions() {
const instructionsDiv = document.createElement('div');
instructionsDiv.id = 'instructionsDiv';
instructionsDiv.innerHTML = '

How far can you go?
Your first hit is your last.
UP - Jump, Left/Right - Move
Press "m" to un/mute sound

Start Game ';
$(document).mousemove((e) => {
$('#image').css({ left: e.pageX, top: e.pageY });
});
document.body.appendChild(instructionsDiv);
}

function startGame() {
document.getElementById('instructionsDiv').remove();

// call game loop
update();
}

function createTreesPool() {
const maxTreesInPool = 10;
let new Tree;
for (let i = 0; i < maxTreesInPool; i += 1) {
new Tree = new Tree().body;
treesPool.push(new Tree);
}
}

function handleKeyDow n(keyEvent) {
const leftLane = -1;
const rightLane = 1;

if (jumping) return;
let validMove = true;

if (keyEvent.keyCode === 77) {
// 'M' key
const soundElement = document.getElementById('track');
soundElement.muted = !soundElement.muted;
}
// if (keyEvent.keyCode === 80) pause();
if (keyEvent.keyCode === 37) {
// left
if (currentLane === middleLane) {
currentLane = leftLane;
} else if (currentLane === rightLane) {
currentLane = middleLane;
} else {
validMove = false;
}
} else if (keyEvent.keyCode === 39) {
// right
if (currentLane === middleLane) {
currentLane = rightLane;
} else if (currentLane === leftLane) {
currentLane = middleLane;
} else {
validMove = false;
}
} else {
if (keyEvent.keyCode === 38) {
// up, jump
bounceValue = 0.1;
jumping = true;
}
validMove = false;
}
// heroSphere.position.x = currentLane;
if (validMove) {
jumping = true;
bounceValue = 0.06;
}
}

function addPathTree() {
const options = [0, 1, 2];
let lane = Math.floor(Math.random() * 3);
addTree(true, lane);
options.splice(lane, 1);
if (Math.random() > 0.5) {
lane = Math.floor(Math.random() * 2);
addTree(true, options[lane]);
}
}

function addWorldTrees() {
const numTrees = 36;
const gap = 6.28 / 36;
for (let i = 0; i < numTrees; i += 1) {
addTree(false, i * gap, true);
addTree(false, i * gap, false);
}
}

function addTree(inPath, row, isLeft) {
let new Tree;
```

```
const pathAngleValues = [1.52, 1.57, 1.62];

if (inPath) {
if (treesPool.length === 0) return;
new Tree = treesPool.pop();
new Tree.visible = true;
treesInPath.push(new Tree);
sphericalHelper.set(
w orld.radius - 0.3,
pathAngleValues[row ],
-rollingGroundSphere.rotation.x + 4,
);
} else {
new Tree = new  Tree().body;
let forestAreaAngle = 0; // [1.52,1.57,1.62];
if (isLeft) {
forestAreaAngle = 1.68 + Math.random() * 0.1;
} else {
forestAreaAngle = 1.46 - Math.random() * 0.1;
}
sphericalHelper.set(w orld.radius - 0.3, forestAreaAngle, row );
}
new Tree.position.setFromSpherical(sphericalHelper);
const rollingGroundVector = rollingGroundSphere.position.clone().normalize();
const treeVector = new Tree.position.clone().normalize();
new Tree.quaternion.setFromUnitVectors(treeVector, rollingGroundVector);
new Tree.rotation.x += Math.random() * ((2 * Math.PI) / 10) + -Math.PI / 10;

rollingGroundSphere.add(new Tree);
}

function update() {
const gravity = 0.005;
const treeReleaseInterval = 0.5;
const levelUpdateInterval = 30;

if (gameOverFlag) return;

if (levelClock.getElapsedTime() > levelUpdateInterval) {
// update level & game speed
levelClock.start();
rollingSpeed += 0.001;
levelCounter += 1;
// update ground color using scheduler
rollingGroundSphere.material.color.setHex(scheduler.getNextColor());
notifyLevel(levelCounter);
}
rollingGroundSphere.rotation.x += rollingSpeed;
hero.update();
if (heroSphere.position.y <= 2 hero.basey) { jumping="false;" bouncevalue="Math.random()" * 0.04 + 0.005; }
herosphere.position.y herosphere.position.x="THREE.Math.lerp(" herosphere.position.x, currentlane, treeclock.getdelta(),
treeclock.getelapsedtime(), ); -="gravity;" if (treeclock.getelapsedtime()> treeReleaseInterval) {
treeClock.start();
addPathTree();
distanceCounter += 1;
if (!hasCollided) {
score += 2 * treeReleaseInterval;
scoreText.innerHTML = score.toString();
distanceMeter.innerHTML = Completed: ${distanceCounter}m<br>Highest: ${localStorage.getItem('newscore')}m;
if (distanceCounter > localStorage.getItem('new score')) {
localStorage.setItem('new score', distanceCounter);
}
} else {
gameOver();
}
}
doTreeLogic();
explosion.logic();
render();
globalRenderID = requestAnimationFrame(update); // request next update
}

function doTreeLogic() {
let oneTree;
const treePos = new THREE.Vector3();
const treesToRemove = [];
treesInPath.forEach((element, index) => {
oneTree = treesInPath[index];
treePos.setFromMatrixPosition(oneTree.matrixWorld);
if (treePos.z > 6 && oneTree.visible) {
// gone out of our view zone
treesToRemove.push(oneTree);
} else if (treePos.distanceTo(heroSphere.position) <= 0.6) { hascollided="true;" explosion.explode(herosphere); } }); let
fromw here; treestoremove.foreach((_, index)> {
oneTree = treesToRemove[index];
fromWhere = treesInPath.indexOf(oneTree);
treesInPath.splice(fromWhere, 1);
treesPool.push(oneTree);
oneTree.visible = false;
});
}

function render() {
renderer.render(scene, camera);
}

function gameOver() {
const gameOverDiv = document.createElement('div');
gameOverDiv.id = 'gameOverDiv';
gameOverDiv.innerHTML = <p id='gameOverText'> GAME OVER WITH SCORE OF: ${score} </p> <button id='restart'
onClick='restart()'>Restart Game</button>;
document.body.appendChild(gameOverDiv);

score = 0;
scoreText.innerHTML = score.toString();

distanceCounter = 0;
rollingGroundSphere.rotation.x = 0;
rollingSpeed = 0;
scheduler.reset();
levelClock.stop();
gameOverFlag = true;

cancelAnimationFrame(globalRenderID);
```

```
}

function restart() {
gameOverFlag = false;
hasCollided = false;
score = 0;
const parent = document.getElementById('gameOverDiv').parentElement;
parent.removeChild(document.getElementById('gameOverDiv'));
rollingSpeed = initRollingSpeed;
levelCounter = 1;
scheduler.reset();
rollingGroundSphere.material.color.setHex(scheduler.getNextColor());
levelClock.stop();
update();
}

function pause() {
isPaused = !isPaused;
if (isPaused) {
rollingGroundSphere.rotation.x = 0;
rollingSpeed = 0;
} else {
rollingSpeed = initRollingSpeed;
update();
}
}

function notifyLevel(level) {
const levelUpDiv = document.createElement('div');
levelUpDiv.id = 'levelUpDiv';
levelUpDiv.innerHTML = <p id='levelUpText'> Level ${level} </p>;
document.body.appendChild(levelUpDiv);
window.setTimeout(() => {
document.getElementById('levelUpDiv').remove();
}, 500);
}

function onWindowResize() {
// resize & align
sceneHeight = window.innerHeight;
sceneWidth = window.innerWidth;
renderer.setSize(sceneWidth, sceneHeight);
camera.aspect = sceneWidth / sceneHeight;
camera.updateProjectionMatrix();
}
```