**Game Programming**

**Assignment 2**

**Mini Game Development**

**17BCE0421**

**Kowshik S**

**Title of the game:** Jerry’s Escape Ride

**Story of the game:**

Jerry will be trapped in a house, in search of way out it finds a jetpack. On the jetpack it will be written that “complete the collection of 500 coins to escape from the house”. So, the mice wear the jetpack and collects the 500 coins through lasers spinning to find a way out. At the end it finds that Tom(cat) has setup this trap to kill him.

**Description of design and development of the game:**

Design of the game completely involves 2d objects. I used background to be repeating until the game is finished. Character can fly, run, fall and die. Background is normal wall picture. Some decorations such as bookshelves and mouse holes are used.

**Game playing procedure:**

This game can be played with touch screen as well as using a mouse. The player needs to click mouse to jump and to needs to do it timely to avoid obstacles and collect coins.

**Specify in which language, tool, Game Engine and etc. you developed the game:**

C sharp is used as language, tool is Microsoft visual studio, engine is unity.

**Source codes:**

**Code for controlling mouse:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class MouseController : MonoBehaviour {

public float jetpackForce = 75.0f;

private Rigidbody2D playerRigidbody;

public float forwardMovementSpeed = 3.0f;

public Transform groundCheckTransform;

private bool isGrounded;

public LayerMask groundCheckLayerMask;

private Animator mouseAnimator;

public ParticleSystem jetpack;

private bool isDead = false;

private uint coins = 0;

public Text coinsCollectedLabel;

public Button restartButton;

public AudioClip coinCollectSound;

public AudioSource jetpackAudio;

public AudioSource footstepsAudio;

public ParallaxScroll parallax;

void Start () {

playerRigidbody = GetComponent<Rigidbody2D>();

mouseAnimator = GetComponent<Animator>();

}

public void RestartGame()

{

SceneManager.LoadScene("RocketMouse");

}

void AdjustJetpack(bool jetpackActive)

{

var jetpackEmission = jetpack.emission;

jetpackEmission.enabled = !isGrounded;

if (jetpackActive)

{

jetpackEmission.rateOverTime = 300.0f;

}

else

{

jetpackEmission.rateOverTime = 75.0f;

}

}

void CollectCoin(Collider2D coinCollider)

{

coins++;

coinsCollectedLabel.text = coins.ToString();

Destroy(coinCollider.gameObject);

AudioSource.PlayClipAtPoint(coinCollectSound, transform.position);

}

void UpdateGroundedStatus()

{

isGrounded = Physics2D.OverlapCircle(groundCheckTransform.position, 0.1f, groundCheckLayerMask);

mouseAnimator.SetBool("isGrounded", isGrounded);

}

void OnTriggerEnter2D(Collider2D collider)

{

if (collider.gameObject.CompareTag("Coins"))

{

CollectCoin(collider);

}

else

{

HitByLaser(collider);

}

}

void HitByLaser(Collider2D laserCollider)

{

if (!isDead)

{

AudioSource laserZap = laserCollider.gameObject.GetComponent<AudioSource>();

laserZap.Play();

}

isDead = true;

mouseAnimator.SetBool("isDead", true);

}

void FixedUpdate()

{

bool jetpackActive = Input.GetButton("Fire1");

jetpackActive = jetpackActive && !isDead;

if (jetpackActive)

{

playerRigidbody.AddForce(new Vector2(0, jetpackForce));

}

if (!isDead)

{

Vector2 newVelocity = playerRigidbody.velocity;

newVelocity.x = forwardMovementSpeed;

playerRigidbody.velocity = newVelocity;

}

UpdateGroundedStatus();

AdjustJetpack(jetpackActive);

if (isDead && isGrounded)

{

restartButton.gameObject.SetActive(true);

}

AdjustFootstepsAndJetpackSound(jetpackActive);

parallax.offset = transform.position.x;

}

void AdjustFootstepsAndJetpackSound(bool jetpackActive)

{

footstepsAudio.enabled = !isDead && isGrounded;

jetpackAudio.enabled = !isDead && !isGrounded;

if (jetpackActive)

{

jetpackAudio.volume = 1.0f;

}

else

{

jetpackAudio.volume = 0.5f;

}

}

}

**Description:**

This will be the force applied to the mouse when the jetpack is on. When the game starts, you retain a reference to the player’s Rigid body. Fixed Update, you check if the Fire1 button is currently pressed. Add Force simply applies the force to the rigid body.

**Forward Movement Speed** will define how fast the mouse moves forward.

**Code for lasers:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LaserScript : MonoBehaviour {

public Sprite laserOnSprite;

public Sprite laserOffSprite;

public float toggleInterval = 0.5f;

public float rotationSpeed = 0.0f;

private bool isLaserOn = true;

private float timeUntilNextToggle;

private Collider2D laserCollider;

private SpriteRenderer laserRenderer;

void Start () {

timeUntilNextToggle = toggleInterval;

laserCollider = gameObject.GetComponent<Collider2D>();

laserRenderer = gameObject.GetComponent<SpriteRenderer>();

}

void Update () {

timeUntilNextToggle -= Time.deltaTime;

if (timeUntilNextToggle <= 0)

{

isLaserOn = !isLaserOn;

laserCollider.enabled = isLaserOn;

if (isLaserOn)

{

laserRenderer.sprite = laserOnSprite;

}

else

{

laserRenderer.sprite = laserOffSprite;

}

timeUntilNextToggle = toggleInterval;

}

transform.RotateAround(transform.position, Vector3.forward, rotationSpeed \* Time.deltaTime);

}

}

**Description:**

1. The Laser has two states: On and Off, and there is a separate image for each state. You will specify each state image in just a moment.
2. These properties allow you to add a bit of random fluctuation. You can set a different toggleInterval so that all lasers on the level don’t work exactly the same. By setting a low interval, you create a laser that will turn on and off quickly, and by setting a high interval you will create a laser that will stay in one state for some time. The rotationSpeed variable serves a similar purpose and specifies the speed of the laser rotation.
3. These two private variables are used to toggle the laser’s state.
4. These two private variables hold references to the laser collider and renderer so that their properties can be adjusted.
5. This will set the time until the laser should toggle its state for the first time.
6. Here we save references to the collider and renderer as you will need to adjust their properties during their lifetime.

**Code for Camera flow:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraFlow : MonoBehaviour {

public GameObject targetObject;

private float distanceToTarget;

void Start () {

distanceToTarget = transform.position.x - targetObject.transform.position.x;

}

void Update () {

float targetObjectX = targetObject.transform.position.x;

Vector3 newCameraPosition = transform.position;

newCameraPosition.x = targetObjectX + distanceToTarget;

transform.position = newCameraPosition;

}

}

**Description:**

This code simply takes the x-coordinate of the target object and moves the camera to that position.

**Code for ParallaxScroll:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ParallaxScroll : MonoBehaviour {

public Renderer background;

public Renderer foreground;

public float backgroundSpeed = 0.02f;

public float foregroundSpeed = 0.06f;

public float offset = 0.0f;

void Start () {

}

void Update () {

float backgroundOffset = offset \* backgroundSpeed;

float foregroundOffset = offset \* foregroundSpeed;

background.material.mainTextureOffset = new Vector2(backgroundOffset, 0);

foreground.material.mainTextureOffset = new Vector2(foregroundOffset, 0);

}

}

**Description:**

1. The Renderer variables will hold a reference to the **Mesh Renderer**component of each of the quads so that you can adjust their texture properties.
2. The backgroundSpeed and foregroundSpeed just define the speed for each background.
3. The offset will be provided by the player’s position. This will enable you to couple the mouse’s movement to the movement of the parallax background. If your pick up a power up and boost forward, the background will move quickly. If the player dies, the movement stops.
4. This code increases the texture offset of each of the quad’s textures with the offset, thus moving it. The resulting speed is different since the script uses the backgroundSpeed and foregroundSpeed coefficients.

**Code for generatorscripts:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GeneratorScript : MonoBehaviour {

public GameObject[] availableRooms;

public List<GameObject> currentRooms;

private float screenWidthInPoints;

public GameObject[] availableObjects;

public List<GameObject> objects;

public float objectsMinDistance = 5.0f;

public float objectsMaxDistance = 10.0f;

public float objectsMinY = -1.4f;

public float objectsMaxY = 1.4f;

public float objectsMinRotation = -45.0f;

public float objectsMaxRotation = 45.0f;

void Start () {

float height = 2.0f \* Camera.main.orthographicSize;

screenWidthInPoints = height \* Camera.main.aspect;

StartCoroutine(GeneratorCheck());

}

void AddObject(float lastObjectX)

{

int randomIndex = Random.Range(0, availableObjects.Length);

GameObject obj = (GameObject)Instantiate(availableObjects[randomIndex]);

float objectPositionX = lastObjectX + Random.Range(objectsMinDistance, objectsMaxDistance);

float randomY = Random.Range(objectsMinY, objectsMaxY);

obj.transform.position = new Vector3(objectPositionX, randomY, 0);

float rotation = Random.Range(objectsMinRotation, objectsMaxRotation);

obj.transform.rotation = Quaternion.Euler(Vector3.forward \* rotation);

objects.Add(obj);

}

void GenerateObjectsIfRequired()

{

float playerX = transform.position.x;

float removeObjectsX = playerX - screenWidthInPoints;

float addObjectX = playerX + screenWidthInPoints;

float farthestObjectX = 0;

List<GameObject> objectsToRemove = new List<GameObject>();

foreach (var obj in objects)

{

float objX = obj.transform.position.x;

farthestObjectX = Mathf.Max(farthestObjectX, objX);

if (objX < removeObjectsX)

{

objectsToRemove.Add(obj);

}

}

foreach (var obj in objectsToRemove)

{

objects.Remove(obj);

Destroy(obj);

}

if (farthestObjectX < addObjectX)

{

AddObject(farthestObjectX);

}

}

void AddRoom(float farthestRoomEndX)

{

int randomRoomIndex = Random.Range(0, availableRooms.Length);

GameObject room = (GameObject)Instantiate(availableRooms[randomRoomIndex]);

float roomWidth = room.transform.Find("floor").localScale.x;

float roomCenter = farthestRoomEndX + roomWidth \* 0.5f;

room.transform.position = new Vector3(roomCenter, 0, 0);

currentRooms.Add(room);

}

private void GenerateRoomIfRequired()

{

List<GameObject> roomsToRemove = new List<GameObject>();

bool addRooms = true;

float playerX = transform.position.x;

float removeRoomX = playerX - screenWidthInPoints;

float addRoomX = playerX + screenWidthInPoints;

float farthestRoomEndX = 0;

foreach (var room in currentRooms)

{

float roomWidth = room.transform.Find("floor").localScale.x;

float roomStartX = room.transform.position.x - (roomWidth \* 0.5f);

float roomEndX = roomStartX + roomWidth;

if (roomStartX > addRoomX)

{

addRooms = false;

}

if (roomEndX < removeRoomX)

{

roomsToRemove.Add(room);

}

farthestRoomEndX = Mathf.Max(farthestRoomEndX, roomEndX);

}

foreach (var room in roomsToRemove)

{

currentRooms.Remove(room);

Destroy(room);

}

if (addRooms)

{

AddRoom(farthestRoomEndX);

}

}

private IEnumerator GeneratorCheck()

{

while (true)

{

GenerateRoomIfRequired();

GenerateObjectsIfRequired();

yield return new WaitForSeconds(0.25f);

}

}

void Update () {

}

}

**Description:**

The availableRooms field will contain an array of Prefabs, which the script can generate.

The currentRooms list will store instanced rooms, so that it can check where the last room ends and if it needs to add more rooms.

The screenWidthInPoints field is just required to cache the screen size in points.

his method adds a new room using the farthestRoomEndX point, which is the rightmost point of the level so far.

The while loop will ensure any code will continue to be executed whilst the game is running and the GameObject is active. Operations involving List<> can be performance limiting; therefore, a yield statement is used to add a 0.25 second pause in execution between each iteration of the loop. GenerateRoomIfRequired is only executed as often as it is required.

**Screen Shots:**

