**Desarrollo de app VR nativa en Unity**

Índice:

-Setup Básico.

-Manejo de objeto (Acción-reacción).

-Teletransportación.

-Movimiento continuo.

-Interacción de sujetar.

Anexo:

-LocomotionController

- ContinuousMovement

-Disparador simple

-Disparador completo

Setup Básico

-Usar template 3D para generar proyecto.

**Instalación del XR interaction plugin**

-Windows>Package Manager

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media-Tipo de package: Unity Registry // Configuración: Advanced Project Settings

-Marcar checkbox: Enable Preview Packages

-Ubicar en la lista de packages: **XR Interaction Toolkit** // Instalar

-Edit>>Project Settings>>XR Plugin Management>> Install XR Plugin Management

-Seleccionar dispositivos en checkbox:

Para PC-Oculus & Windows Mixed Reality

Para Android-Oculus

-Crear plano y asignarle material

**Crear Referencia de origen XR**

-Desde la zona de Jerarquía presionar clic derecho: XR>>XR Origin (Action-based)

-En el inspector remover: Tracker Pose Driver (New Input System)

-En el inspector agregar: Tracked Pose Driver

-Seleccionando Left y Right controller, remover del inspector los componentes XR Interact Visual, Line Rendever, XR Ray Interactor y XR Controller (Action Based).

-Seleccionando Left y Right controller, agregar en el inspector los componentes:

Una captura de pantalla de una computadora

Descripción generada automáticamente-XR Controller (Device-Based). Configurar en Controller node para cada Hand (Left para izquierda y Right para derecha).

-XR Direct Interactor

-Sphere Collider: ajustar el radio a 0.15; activar checkbox: Is Trigger.

**Visualización de manos**

-Importar a la carpeta Assets el paquete de Unity: XR Toolkit Hand Presence

-Assets>Models>Oculus Hands>Materials. En el inspector cambiar Rendering Mode a Opaque. Es posible ajustar el color de las manos pero es algo opcional.

-Arrastrar los prefabs de cada mano hasta el componente que le corresponde en la jerarquía.

Pantalla de una computadora

Descripción generada automáticamente con confianza mediaAcción recomendada: Presionar Play en la parte superior y con los lentes conectados verificar el funcionamiento de las manos y la visualización mirando hacia los lados.

Manejo de objeto (Acción-reacción)

Experimentando con un disparador que pueda ser sujetado en cada mano y que la acción de disparar suceda al presionar un botón.

Requerimientos:

-Mesa para colocar el objeto.

-Prefab de bala (Con componente Rigdbody y el Use Gravity activado).

-Asset del disparador.

Recomendaciones para configurar el disparador:

-Agregar al elemento principal del disparador los siguientes componentes:

-Rigidbody: agregar en la opción Interpolate = Interpolate.

-XR Grab Interactable: asegurando agregar como collider los elementos principales del disparador.

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

-Crear Grab Point ubicado en la zona donde de sujeta el disparador e incluirlo en XR Grab Interactable>Force Gravity On Detach>Attach transform

-Crear punto de disparo (tener en cuenta la ubicación y la rotación)

-Nuevo script del disparador e incluyendo los elementos de la bala y del punto de disparo

Captura de pantalla de computadora

Descripción generada automáticamente-Crear nueva interacción, desde Interactable Events>Activate>Activated (ActivateEventArgs), arrastrar componente del script del disparador hasta el objeto de la interacción. Agregar la función: Gun(nombre del script deñ disparador)>Shoot()

Captura de pantalla de computadora

Descripción generada automáticamenteAcción recomendada: Probar funcionamiento del manejo de los objetos para ambas manos y acciones de interacción programadas al presionar un botón.

Teletransportación

Funcionalidad para desplazarse sobre cualquier plano apuntando con el cursor la zona deseada.

**Asignando rayo de referencia**

Desde la sección de jerarquía seleccionar VR Ring y agregar los siguientes componentes:

-Locomotion System

-Teleportation Provider.

En VR Rig>>Camera Offset generar el asset XR>>Ray Interactor. Este nos servirá para tener un indicador visual del punto a donde se realiza la teletransportación.

-Se deben generar un asset para cada mano, realizando el ajuste en el componente XR Controller>>Configuration>>Controller Node; Acá debe estar asignado Right para la mano derecha y Left Hand para la mano izquierda.

-Se deben ajustar las coordinadas de posición para que todas se encuentren en 0.

**Zona de teletransporte**

Existen 2 maneras de generar zonas de teletransportación usando un Área o un Ancla(Anchor). En este caso usaremos un área, ya que nos permite teletransportarnos a cualquier punto dentro esa zona.

-En la zona de jerarquía generar el asset XR>>Teleportation zone. Y asigna la posición deseada.

-También se le puede asignar zonas de teletransporte a objetos 3D, Como por ejemplo en un cubo es suficiente con agregar el componente Teleportation Area.

**Recomendación:** En este punto se puede probar la correcta funcionalidad de teletransporte señalando con el mando algún punto en la zona de teletransporte. Nótese que en las zonas habilitadas el haz de luz debe ser verde y en las zonas no habilitadas debe verse de color rojo.

Para evitar teletransportaciones a través de superficies sólidas es necesario ajustar en el Teleport Ray ubicado en la zona de jerarquía el componente XR Controller lo siguiente:

-Select Usage: Trigger.

-Axis To Press Threshold: 0.2

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**-Requiere revisión-**

**Tip sobre manejo de objetos:**  uno de los inconvenientes que puede presentarse en la teletransportación es cuando se tiene un objeto sujetado. Para solucionarlo se debe agregar una capa al objeto para que se active solo cuando se desea sujetar. Para esto realizamos lo siguiente:

-Seleccionar el asset de la esfera en la zona de jerarquía para poder inspeccionarlo.

-En la pestaña de Layer seleccionar Add Layer.

-Usar uno de los slots disponibles para crear la Layer Grab.

Interfaz de usuario gráfica

Descripción generada automáticamente-Seleccionar nuevamente el asset de la esfera y asignar el Layer Grab recién creado.

En el Teleport Ray también es necesario realizar un ajuste para evitar las teletransportaciones inconscientes, el ajuste es el siguiente, al componente XR Ray Interactor:

-Asignando en Raycast Mask el valor Grab.

-Selection Action Trigger el valor State.

Para que la referencia de teletransporte sea adecuada para planos de diferentes niveles se requiere modificar el Teleport Ray con los siguientes ajustes al componente XR Ray Interactor:

-XR Ray Interactor>>Line type = Projectile Curve

-Velocity=0.8

-Desactivar el Checkbox en Enable Interaction with UI

**Punto de teletransporte**

Para tener una ayuda visual de donde se realiza la teletransportación se agrega un elemento que permita identificar claramente el punto final.

-Crear elemento 3D del tipo cilindro con una escala pequeña para que se comprenda que es solo una acción de posicionamiento.

-En la zona de jerarquía seleccionar el Teleport Ray y en su inspector ir al componente XR Interactor Line Visual. Arrastrar el asset del cilindro recién creado hasta elemento Reticle del mencionado componente.

Recomendación: realizar test de funcionalidad donde se aprecie el elemento de referencia y la correcta teletransportación.

Imagen que contiene pasto, tabla

Descripción generada automáticamente

**Visualización de referencia solo al ejecutar acción**

Para evitar que la referencia de teletransporte este siempre activada se requiere utilizar un script que controle la visualización y ejecución de teletransporte solo cuando se presionen los botones adecuados**.**

-En la zona de jerarquía seleccionar el VR Rig y en el inspector se debe generar un nuevo script que en este caso será llamado: LocomotionController (Detalle del script en anexo).

-Una vez implementado se debe arrastrar al inspector los elementos con los que interactúa (Left Teleport Ray & Right Teleport Ray) en sus slots correspondientes.

Captura de pantalla de un celular

Descripción generada automáticamente-Dentro del componente del script recién creado asignar la activación configurando el Teleport Activation Button con el valor Trigger.

Recomendación: Realizar prueba de funcionalidad donde el elemento de teletransportación solo se visualice cuando se presione el botón de acción sin tener ningún objeto sujetado en las manos.

Movimiento continuo

Esta funcionalidad permitirá poder realizar un desplazamiento lineal sobre cualquier plano. Además de eso también se explicará la integración para girar la vista sobre un eje vertical.

**Configurando Script de movimiento**

-En la zona de jerarquía agregar VR Rig un nuevo componente, un nuevo script llamado: ContinuousMovement.

-Una de las primeras cosas que se debe agregar en una variable pública que luego desde el inspector dse deb configurar para que el Input source sea (en este caso) Left Hand.

En esta etapa se recomienda unas el script “ContinuousMovement Parte I” disponible en los anexos, que contiene la base de desplazamiento, actualización de estado y lectura de dispositivo.

-En VR Rig agregar componente Character Controller, que permitirá tener un cilindro de referencia de la ubicación del personaje. El radio y las proporciones varían dependiendo de la necesidad de la experiencia, en este caso se usará un radio de 0.15 y el centro se posicionará en las coordenadas (0,1,0).

Recomendación: en este punto se pueda realizar una prueba para verificar el correcto desplazamiento teniendo en cuenta. Desde el game scene se debe apreciar el cilindro y el desplazamiento.

Gráfico

Descripción generada automáticamente

Completando funciones adicionales al script se agregan los métodos para agregar gravedad al desplazamiento, así como también la verificación del seguimiento de la cápsula a la posición principal del personaje correspondiente a los cascos VR.

-Verificar en el script las secciones Gravity, CapsuleFollowHeadset y CheckIfGrounded (“ContinuousMovement Parte II”)

Para mejorar la suavidad en los fotogramas se debe ajustar el tiempo de duración.

-En la zona superior ir a Edit>>Project settings>>Time>>FIxed Timestep = 0.011111.

**Agregar rotación**

-En la zona de jerarquía agregar VR Rig un nuevo componente llamado: Snap Turn Provider.

-Controllers>>Size = 1.

-Element 0 = Right Hand (XRController)

-Turno Amount (Ángulo de giro) = 30.

-Activation Timeout = 0.2

Interacción de sujetar

-Ajuste de colisión con el cuerpo

-Tipos de movimiento

-Vibración al seleccionar

-Bloquear teletransportación al sujetar objetos (actualización del script locomotioncontroller)

Rayo de interacción

-Creación y ajuste de rayo

-

Anexo

LocomotionController

-

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.XR.Interaction.Toolkit;

public class LocomotionController : MonoBehaviour

{

public XRController leftTeleportRay;

public XRController rightTeleportRay;

public InputHelpers.Button teleportActivationButton;

public float activationThreshold = 0.1f;

// Update is called once per frame

void Update()

{

if (leftTeleportRay)

{

leftTeleportRay.gameObject.SetActive(CheckIfActivated(leftTeleportRay));

}

if (rightTeleportRay)

{

rightTeleportRay.gameObject.SetActive(CheckIfActivated(rightTeleportRay));

}

}

public bool CheckIfActivated(XRController controller)

{

InputHelpers.IsPressed(controller.inputDevice, teleportActivationButton, out bool isActivated, activationThreshold);

return isActivated;

}

}

-

ContinuousMovement

-Parte I

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.XR;

using UnityEngine.XR.Interaction.Toolkit;

using Unity.XR.CoreUtils;

public class ContinuousMovement : MonoBehaviour

{

public float speed = 1;

public XRNode inputSource;

private Vector2 inputAxis;

private CharacterController character;

private XROrigin origin;

// Start is called before the first frame update

void Start()

{

character = GetComponent<CharacterController>();

origin = GetComponent<XROrigin>();

}

// Update is called once per frame

void Update()

{

InputDevice device = InputDevices.GetDeviceAtXRNode(inputSource);

device.TryGetFeatureValue(CommonUsages.primary2DAxis, out inputAxis);

}

private void FixedUpdate()

{

Quaternion headYaw = Quaternion.Euler(0, origin.Camera.transform.eulerAngles.y, 0);

Vector3 direction = headYaw \* new Vector3(inputAxis.x, 0, inputAxis.y);

character.Move(direction \* Time.fixedDeltaTime \* speed);

}

}

-

-Parte II

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.XR;

using UnityEngine.XR.Interaction.Toolkit;

using Unity.XR.CoreUtils;

public class ContinuousMovement : MonoBehaviour

{

public float speed = 1;

public XRNode inputSource;

private Vector2 inputAxis;

private CharacterController character;

public float gravity = -9.81f;

private float fallingSpeed;

private XROrigin origin;

public LayerMask groundLayer;

public float additionalHeight = 0.2f;

// Start is called before the first frame update

void Start()

{

character = GetComponent<CharacterController>();

origin = GetComponent<XROrigin>();

}

// Update is called once per frame

void Update()

{

InputDevice device = InputDevices.GetDeviceAtXRNode(inputSource);

device.TryGetFeatureValue(CommonUsages.primary2DAxis, out inputAxis);

}

private void FixedUpdate()

{

CapsuleFollowHeadset();

Quaternion headYaw = Quaternion.Euler(0, origin.Camera.transform.eulerAngles.y, 0);

Vector3 direction = headYaw \* new Vector3(inputAxis.x, 0, inputAxis.y);

character.Move(direction \* Time.fixedDeltaTime \* speed);

//gravity

bool isGrounded = CheckIfGrounded();

if (isGrounded)

{

fallingSpeed = 0;

}

else

{

fallingSpeed += gravity \* Time.fixedDeltaTime;

}

character.Move(Vector3.up \* fallingSpeed \* Time.fixedDeltaTime);

}

void CapsuleFollowHeadset()

{

character.height = origin.CameraInOriginSpaceHeight + additionalHeight;

Vector3 capsuleCenter = transform.InverseTransformPoint(origin.Camera.transform.position);

character.center = new Vector3(capsuleCenter.x, character.height / 2 + character.skinWidth, capsuleCenter.z);

}

bool CheckIfGrounded()

{

//tells us if on ground

Vector3 rayStart = transform.TransformPoint(character.center);

float rayLength = character.center.y + 0.01f;

bool hasHit = Physics.SphereCast(rayStart, character.radius, Vector3.down, out RaycastHit hitInfo, rayLength, groundLayer);

return hasHit;

}

}

-

Versión para evitar el Assets\ContinuousMovement.cs(23,28): warning CS0618: 'XRRig' is obsolete: 'XRRig has been deprecated. Use the XROrigin component instead.'

-

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.XR;

using UnityEngine.XR.Interaction.Toolkit;

using Unity.XR.CoreUtils;

public class ContinuousMovement : MonoBehaviour

{

public float speed = 1;

public XRNode inputSource;

public float gravity = -9.81f;

public LayerMask groundLayer;

public float additionalHeight = 0.2f;

private float fallingSpeed;

private XROrigin origin;

private Vector2 inputAxis;

private CharacterController character;

// Start is called before the first frame update

void Start()

{

character = GetComponent<CharacterController>();

origin = GetComponent<XROrigin>();

}

// Update is called once per frame

void Update()

{

InputDevice device = InputDevices.GetDeviceAtXRNode(inputSource);

device.TryGetFeatureValue(CommonUsages.primary2DAxis, out inputAxis);

}

private void FixedUpdate()

{

CapsuleFollowHeadset();

Quaternion headYaw = Quaternion.Euler(0, origin.Camera.transform.eulerAngles.y, 0);

Vector3 direction = headYaw \* new Vector3(inputAxis.x, 0, inputAxis.y);

character.Move(direction \* Time.fixedDeltaTime \* speed);

//gravity

bool isGrounded = CheckIfGrounded();

if (isGrounded)

fallingSpeed = 0;

else

fallingSpeed += gravity \* Time.fixedDeltaTime;

character.Move(Vector3.up \* fallingSpeed \* Time.fixedDeltaTime);

}

void CapsuleFollowHeadset()

{

character.height = origin.CameraInOriginSpaceHeight + additionalHeight;

Vector3 capsuleCenter = transform.InverseTransformPoint(origin.Camera.transform.position);

character.center = new Vector3(capsuleCenter.x, character.height/2 + character.skinWidth, capsuleCenter.z);

}

bool CheckIfGrounded()

{

//tells us if on ground

Vector3 rayStart = transform.TransformPoint(character.center);

float rayLegth = character.center.y + 0.01f;

bool hasHit = Physics.SphereCast(rayStart, character.radius, Vector3.down, out RaycastHit hitInfo, rayLegth, groundLayer);

return hasHit;

}

}

-

- Versión depreciada

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.XR;

using UnityEngine.XR.Interaction.Toolkit;

public class ContinuousMovement : MonoBehaviour

{

public float speed =1;

public XRNode inputSource;

public float gravity = -9.81f;

public LayerMask groundLayer;

public float additionalHeight = 0.2f;

private float fallingSpeed;

private XRRig rig;

private Vector2 inputAxis;

private CharacterController character;

// Start is called before the first frame update

void Start()

{

character = GetComponent<CharacterController>();

rig = GetComponent<XRRig>();

}

// Update is called once per frame

void Update()

{

InputDevice device = InputDevices.GetDeviceAtXRNode(inputSource);

device.TryGetFeatureValue(CommonUsages.primary2DAxis, out inputAxis);

}

private void FixedUpdate()

{

CapsuleFollowHeadset();

Quaternion headYaw = Quaternion.Euler(0, rig.cameraGameObject.transform.eulerAngles.y, 0);

Vector3 direction = headYaw \* new Vector3(inputAxis.x, 0, inputAxis.y);

character.Move(direction \* Time.fixedDeltaTime \* speed);

//gravity

bool isGrounded = CheckIfGrounded();

if (isGrounded)

{

fallingSpeed = 0;

}

else

{

fallingSpeed += gravity \* Time.fixedDeltaTime;

}

character.Move(Vector3.up \* fallingSpeed \* Time.fixedDeltaTime);

}

void CapsuleFollowHeadset()

{

character.height = rig.cameraInRigSpaceHeight + additionalHeight;

Vector3 capsuleCenter = transform.InverseTransformPoint(rig.cameraGameObject.transform.position);

character.center = new Vector3(capsuleCenter.x, character.height/2 + character.skinWidth, capsuleCenter.z);

}

bool CheckIfGrounded()

{

//tells us if on ground

Vector3 rayStart = transform.TransformPoint(character.center);

float rayLength = character.center.y + 0.01f;

bool hasHit = Physics.SphereCast(rayStart, character.radius, Vector3.down, out RaycastHit hitInfo, rayLength, groundLayer);

return hasHit;

}

}

-

Disparador simple

-

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class gun : MonoBehaviour

{

[SerializeField] GameObject bulletPrefab, shootPoint;

public void Shoot()

{

Instantiate(bulletPrefab, shootPoint.transform.position, Quaternion.identity);

}

}

-

Disparador completo

-

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class gun : MonoBehaviour

{

public float speed = 40;

public GameObject bullet;

public Transform barrel;

public void Fire()

{

GameObject spawnedBullet = Instantiate(bullet, barrel.position, barrel.rotation);

spawnedBullet.GetComponent<Rigidbody>().velocity = speed \* barrel.forward;

Destroy(spawnedBullet, 2);

}

}

-