UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

DONANT VIDA A LA REALITAT VIRTUAL

Desenvolupament d'un braç amb moviment IK en Unity per una experiència de joc interactiva

---

Ruben Gimbert Roldán

Directors: Ignasi Cos Aguilera i Marc Soler Bages
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 13 de juny de 2023

Índex

1. Introducció i motivació

Actualment, la Realitat Virtual (VR) ha guanyat un considerable impuls en l'àmbit dels videojocs i les experiències interactives. Els avenços en tecnologia i hardware han permès als usuaris submergir-se en entorns virtuals i gaudir d'experiències immersives. No obstant això, un dels reptes clau en el desenvolupament de jocs de VR és aconseguir una interacció natural i realista entre els usuaris i els objectes virtuals. Dins d'aquest context, el present treball es centra en el desenvolupament d'una experiència immersiva, amb la qual l'usuari pugui tenir una sensació el més realista possible pel que fa als moviments i interaccions amb els objectes virtuals.

El desenvolupament d'experiències de joc interactives en VR ha generat un gran interès tant en la indústria de l'entreteniment com en l'àmbit de la investigació. Aquestes ofereixen un potencial significatiu per explorar noves formes d'entreteniment, educació i entrenament. Però, com bé hem dit, la implementació d'interaccions realistes i naturals segueix sent un repte important. En aquest sentit, es volen estudiar els mètodes que s'utilitzen per generar-les, encara que sigui en un primer nivell, i així plasmar unes bases per tal de poder realitzar altres projectes de major envergadura.

A més dels aspectes tècnics i de la investigació prèviament mencionats, existeix una motivació personal per la realització d'aquest treball. Es basa en el fascinant potencial dels mons virtuals i les experiències immersives per transformar la manera amb la qual interactuem i experimentem la realitat. Des d'una perspectiva més personal, els mons virtuals i experiències immersives sempre han despertat la nostra curiositat i passió per explorar noves formes de comunicació, aprenentatge i entreteniment. Aquests entorns digitals ofereixen una plataforma única per escapar de les limitacions de la realitat i submergir-se en univers creatius sense fronteres. Aquesta capacitat de crear i explorar mons virtuals interactius ens fascina, ja que ofereix la oportunitat de viure experiències que d'altra manera serien impossibles, inaccessibles o potser massa perilloses a la vida real. A través d'aquest treball, volem tenir l'oportunitat d'aprendre i aplicar les bases de la Realitat Virtual, per tal de continuar amb aquesta motivació en un futur, al llarg de la nostra carrera professional.

Hem utilitzat Unity pel desenvolupament del projecte. Unity és un motor de desenvolupament de videojocs multiplataforma que permet la creació i disseny de jocs o experiències interactives en 2D, 3D i realitat virtual (VR). És utilitzat pels desenvolupadors per crear jocs, aplicacions i simulacions per diverses plataformes, com computadores, consoles, dispositius mòbils i dispositius de realitat virtual. Unity ofereix una amplia gama d'eines i recursos, incloent un editor visual, un motor de física, suport per animacions, scripting amb C#, principalment, i una gran quantitat d'assets i plugins disponibles en el seu propi ecosistema. Amb la versatilitat i facilitat del seu ús, Unity s'ha convertit en l'opció més popular tant per desenvolupadors independents com per grans estudis de desenvolupament de videojocs. És per això que hem optat per utilitzar-lo.

2. Objectius

Com hem dit a l'apartat anterior, l'objectiu principal d'aquest Treball de Fi de Grau és desenvolupar una experiència de joc interactiva en realitat virtual. Ho farem mitjançant la creació del model d'un braç virtual amb el qual es pugui interactuar de diverses formes i amb diferents objectes, de manera que puguem tenir una experiència immersiva el més natural i realista possible.

D'aquesta manera, també complirem l'altre objectiu, que és aprendre les bases del desenvolupament en l'entorn de la Realitat Virtual. És un sector en ple apogeu que té infinites possibilitats i un potencial increïble. Amb la qual cosa, aquest és un projecte perfecte per obtenir uns primers coneixements, però molt útils, sobre aquesta emocionant tecnologia. Com hem dit prèviament, utilitzarem Unity pel desenvolupament del treball i, per tant, també és el nostre objectiu aprendre a utilitzar una de les eines, per no dir la més utilitzada en el sector de la Realitat Virtual i els videojocs.

3. Planificació

Una planificació (ideal) del treball a realitzar sempre és una bona eina, o més bé imprescindible, per tal d'assolir al màxim els objectius proposats d'un projecte de certa envergadura com aquest. La nostra manera de fer-ho va ser definint uns objectius a curt, mig i llarg termini.

En primer lloc, ens interessava dur a terme la creació del braç i implementació del seu moviment. Per fer-ho, vam decidir que primer havíem de documentar-nos sobre Unity, ja que no ho havíem utilitzat mai. Una vegada feta la documentació del programa, el següent pas seria la creació d'un model molt simple d'un braç i mirar de com implementar-li un moviment el més natural i realista possible en 2D. Vam pensar que era millor començar per un moviment en 2D, ja que seria molt més senzill d'entendre i implementar, que fer-ho directament en 3D. Aquests objectius a curt termini els teníem pensat assolir en el primer mes i mig (fins la primera meitat de març).

Una vegada fet això, procediríem a ampliar la implementació del moviment a la tercera dimensió. Aquest, el vam definir com un objectiu a mig termini, sempre i quan ens ho permetés el temps per tal de complir amb els objectius establerts. Aquest objectiu no era imprescindible aconseguir-ho, ja que potser estàvem abastant massa feina pel temps que tindríem. Vam pensar tenir-lo fet en un mes a partir d'haver aconseguit els primers objectius (fins la primera meitat d'abril).

Finalment, vam definir com objectiu a llarg termini, poder interactuar amb el braç (o braços) en un entorn de realitat virtual, en el qual poguéssim manipular objectes de diferents maneres. Aquesta interacció la faríem a través d'unes ulleres i comandaments de realitat virtual. Així doncs, aquest sí que el definiríem com un objectiu imprescindible per tal d'assolir els objectius finals del projecte. Li dedicaríem la segona meitat d'abril i el maig per fer-ho.

Després, aquesta planificació que vam idear no va ser la que vam seguir realment, encara que s'acosta força. Els objectius a curt termini ens van portar més temps de l'esperat, ja que ens va ser una mica difícil començar a adaptar-nos a Unity. No obstant això, vam aconseguir realitzar-los en uns 2 mesos aproximadament (febrer i març). Una vegada fet això, vam pensar en continuar amb el moviment en 3D i ens vam documentar per saber com implementar-ho. Vam veure que el pas de 2D a 3D ens comportaria massa temps, primer per aprendre com modificar els càlculs existents i segon per tal d'adaptar-lo al món virtual (encara que el més complicat era entendre i aplicar els càlculs corresponents). Llavors, vam decidir no implementar el moviment en 3D i intentar estudiar les forces de rotació del braç. Vam pensar que els valors de les forces de rotació de les articulacions del braç les podríem calcular, per més endavant fer-ne estudis. Ens va passar el mateix que amb el moviment en 3D, vam estudiar què era i com obtenir-lo, però ja no ens quedava massa temps si volíem assolir els objectius principals. Vam dedicar gairebé tot l'abril per realitzar una documentació tant del moviment en 3D com de les forces de rotació de les articulacions del braç. Finalment, vam passar a l'objectiu final, que era desenvolupar un entorn virtual en el qual poder interactuar amb diversos objectes. Aquest sí que el vam poder assolir, al mateix temps que redactàvem aquesta memòria, durant el mes de maig i principis de juny.

4. Desenvolupament

El que farem a continuació és explicar amb molt més detall tot el procés de desenvolupament del treball. A mesura que anem explicant tots els passos que hem anat fent, també anirem mostrant, sempre que es pugui, els resultats que anem aconseguint. D'aquesta manera, creiem que quedaran més clares les explicacions i els motius de les decisions preses durant el procés.

4.1 Creació i moviment del braç

El primer objectiu que ens vam marcar va ser poder crear un braç a Unity i poder moure'l per interactuar amb els elements d'una escena. Documentant-nos, vam veure que hi havia diversos procediments per tal d'implementar el moviment, entre els quals el de Inverse Kinematics (Cinemàtica Inversa) era un dels més utilitzats i que millor s'adequava al nostre objectiu.

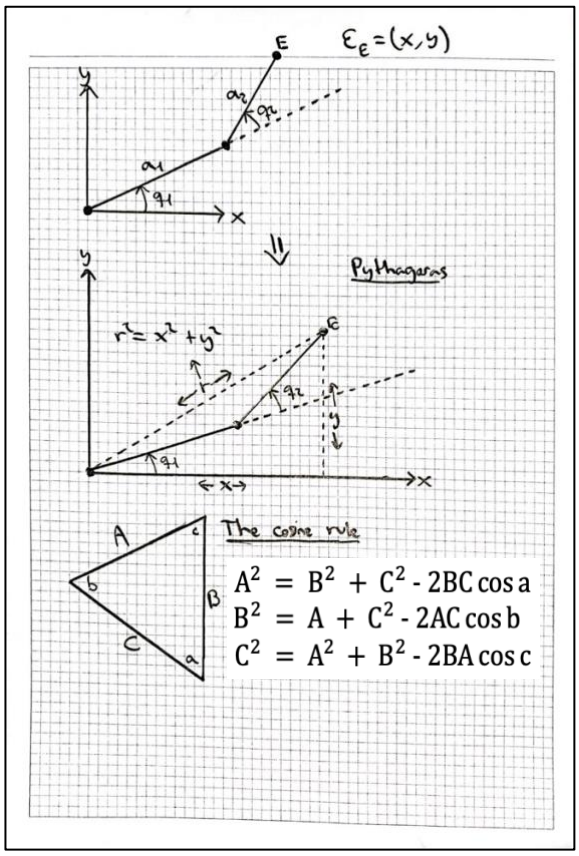
La cinemàtica inversa és una branca de la robòtica i la mecànica que es basa en determinar les posicions i orientacions de les articulacions per aconseguir una posició u orientació específica d'un extrem d'un sistema articulat.

En el nostre cas, tenim un braç articulat de dos segments en el qual hi ha dos articulacions que permeten el moviment del braç: l'espatlla i el colze. La IK permet, doncs, especificar una posició i orientació desitjada per l'extrem del braç i calcular els angles requerits en les articulacions per assolir aquella posició i orientació específica. Això s'aconsegueix per mitjà de diverses fórmules matemàtiques i equacions trigonomètriques per resoldre el sistema d'equacions associat. Típicament s'utilitza la llei del cosinus i les funcions trigonomètriques per calcular els angles mencionats.

El nostre objectiu el teníem clar, però la complexitat que tindria implementar un algoritme en 3D (ja que la nostra escena seria 3D) era molt alta com per fer-ho directament sense passar abans pel 2D. Llavors, vam decidir començar per implementar un algoritme IK que permetés moure el braç sobre un pla (en aquest cas el pla vertical), és a dir, en 2D. Més endavant, havent assolit aquest primer objectiu, ens podríem plantejar la implementació en 3D. Cal detallar que, quan parlem de 2D i 3D, parlem del moviment del braç només, ja que l'escena si que serà 3D independentment del moviment.

4.1.1 Càlculs matemàtics

A continuació, anem a plantejar el problema de manera teòrica. Observem la següent imatge, en la qual es pot visualitzar un esquema d'una possible configuració que podria adoptar el braç:

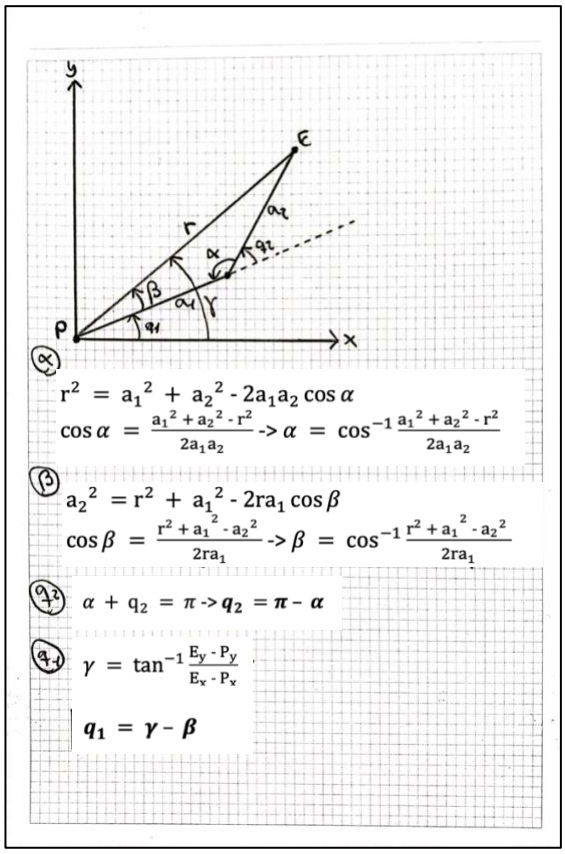


Imatge 1: Primer esquema dels càlculs d'IK

En el primer esquema es mostra, com hem dit, un exemple d'una posició que podria adoptar el braç per tal d'arribar al punt objectiu E. El braç està format per 2 segments, a_1 i a_2, i cadascun té un angle, q_1 i q_2, que determina la seva rotació per tal que l'extrem del braç coincideixi amb el punt E. La rotació de l'espatlla es correspondria amb l'angle q_1 i la del colze amb l'angle q_2.
En el segon esquema, que representa el mateix escenari, hem afegit unes línies discontínues per tal de representar els triangles amb els quals resoldrem les equacions. La distància entre l'espatlla i el punt E, l'hem anomenat r, que seria el tercer costat del triangle.

El tercer esquema es tracta de la Llei del Cosinus, que relaciona les longituds dels costats d'un triangle amb els angles oposats a aquests costats. Pel nostre problema, en el qual coneixem les longituds dels tres costats i volem calcular els angles, és la manera més senzilla.

Una vegada plantejat el problema, passem a la següent fase:

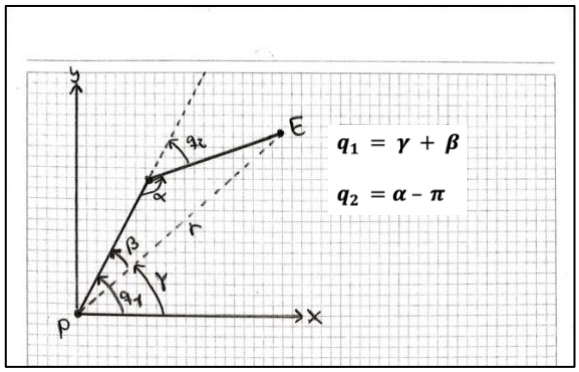


Imatge 2: Segon esquema dels càlculs d'IK

En aquesta imatge 2 es mostra un altre cop el mateix escenari que teníem abans, però anomenant i situant els angles necessaris per obtenir q_1 i q_2, que serien l'angle α, format per r i a_1, i el β, format per a_1 i a_2. Tal i com es mostra, podem obtenir els angles α i β aplicant la Llei del cosinus, ja que coneixem tots els costats del triangle.

Una vegada els tenim, podrem calcular q_1 i q_2. Començarem per l'angle q_2, que és el més senzill i intuïtiu. Si ens fixem en l'esquema, podem deduir que la suma d'α i q_2 és igual a Π (180º). Llavors, q_2 serà igual a la diferència de Π i α. Per calcular q_1, hem de pensar que podem utilitzar l'equació trigonomètrica de la tangent per tal d'obtenir l'angle γ, representat a l'esquema. Per obtenir aquest angle γ, només hem d'obtenir les dimensions de l'angle rectangle que forma el punt P (espatlla) i el punt E (objectiu). És a dir, hem de saber les components del vector PE i aplicar-les a la funció de la tangent, com es mostra al càlcul de q_1. Finalment, podrem obtenir q_1 calculant la diferència entre γ i β.

Cal destacar que aquesta és una de les possibles solucions que hi ha, ja que, com es mostra a la següent imatge, al tenir 2 graus de llibertat n'hi ha 2 solucions pel mateix escenari.



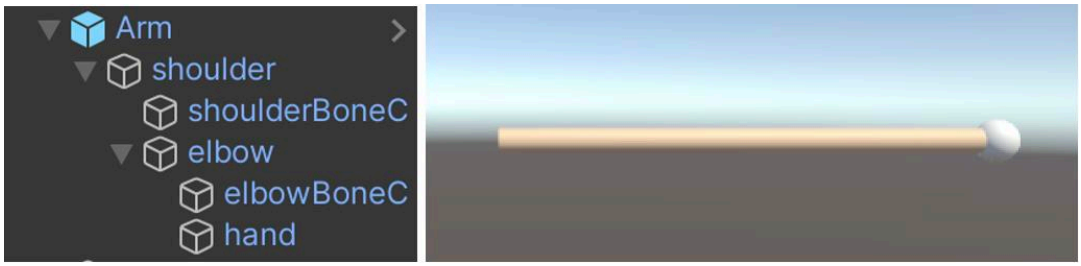
Imatge 3: Tercer esquema dels càlculs d'IK

Aquesta seria, doncs, l'altre possible solució pel mateix escenari. Els càlculs de q_1 i q_2 variaran depenent de l'enfoc que li vulguem donar a la solució.

4.1.2 Implementació a Unity

Una vegada estudiats els càlculs necessaris que involucren obtenir els angles de rotació dels segments del braç, passem a implementar-ho a un projecte de Unity. Per crear un projecte de Unity, primer hem d'instal·lar un editor, si no el tenim ja. La versió de l'editor que hem utilitzat per aquest projecte ha estat la 2021.3.18f1, que era el que ens recomanava pel sistema operatiu macOS. Una vegada el tenim instal·lat (s'instal·la des del propi Unity), cliquem a New Project i seleccionem la plantilla 3D, que ens crearà una escena 3D amb una llum i una càmera.

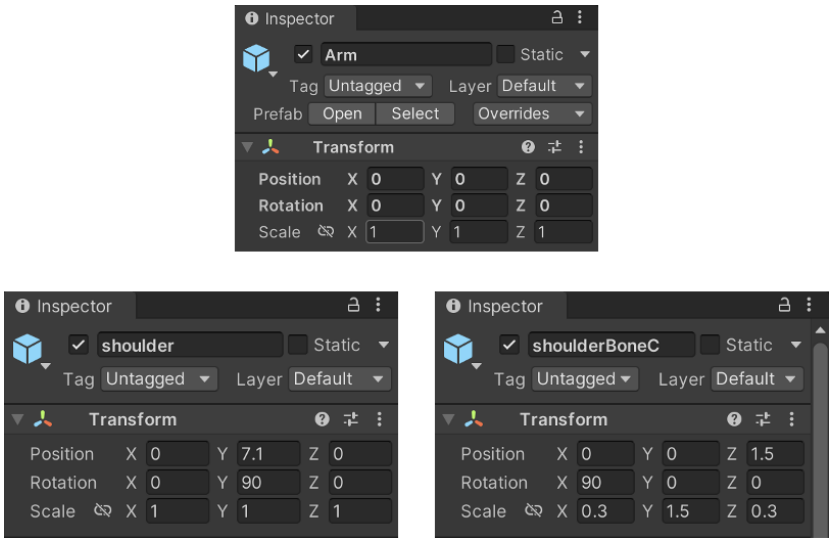
El que primer hem de fer és crear el model del braç. Havíem de pensar en la jerarquia que hauria de tenir el braç i els seus components per tal d'aplicar els càlculs de manera correcta. Hi ha diverses possibilitats per l'organització dels elements que formen el braç. Però, pel nostre problema i tal i com l'havíem plantejat, vam crear el braç seguint la següent jerarquia:



Imatge 4: A l'esquerra la jerarquia del model del braç, a la dreta una il·lustració del model

A Unity es poden crear objectes buits, on es poden afegir altres objectes, com podria ser un cub, una esfera o un cilindre. Llavors, mirant l'esquema de la imatge 4, Arm seria un objecte originalment buit, en el qual hem anat afegint altres objectes fills, representant

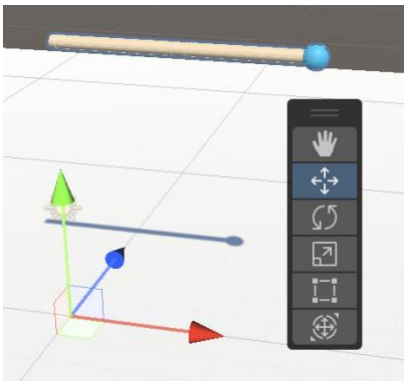
els components del braç. Pensant en el moviment d'un braç, podem deduir que el moviment i rotació de l'espatlla determinen també els de la resta del braç. És a dir, la rotació d'un primer segment, que representaria la part superior del braç, també implicaria que l'altre segment es mogui de la mateixa manera, ja que està lligat a ell. La rotació del segon segment, que representaria la part inferior del braç, en canvi, seria independent del primer segment. És per això que el segment inferior ha de ser fill del primer, perquè tingui el comportament explicat. Per representar els ossos i les seves articulacions, hem creat 2 objectes buits amb un cilindre que representaria el segment o os. Els hem configurat de manera que l'objecte buit, que representaria l'articulació de l'os, queda situat a un dels extrems del cilindre que conté, que representaria l'os. D'aquesta manera, quan rotem l'objecte buit, el cilindre, al seu torn, rotarà sobre un dels extrems, que és justament el que volíem aconseguir. Llavors, només hem de col·locar l'objecte que es correspon amb la part inferior del braç com a fill del que es correspon amb la part superior, tal i com es veu a la imatge 4. Hem afegit també una esfera simulant la mà. Cal destacar que a Unity existeixen les posicions, rotacions i escales en l'espai de món, i les posicions, rotacions i escales locals d'un objecte, que es poden veure i modificar des de la finestra Inspector, seleccionant-lo.



Imatge 5: Components Transform de Arm, shoulder i shoulderBoneC

A la imatge de dalt es poden els components Transform de Arm, shoulder i shoulderBoneC, en els quals es poden veure i editar les posicions, rotacions i escales dels objectes. Els atributs d'Arm, com no és fill de cap altre objecte, es tracta de la posició, rotació i escala en l'espai de món. En canvi, els atributs de shoulder i shoulderBoneC són locals, tenint com a referència el respectiu objecte pare. Perquè s'entengui millor, utilitzarem aquest mateix exemple. Arm està situat al punt (0, 0, 0) en l'espai de món i shoulder, que és fill d'Arm, està situat al punt local (0, 7.1, 0) respecte d'Arm. En aquest cas coincideix també amb la seva posició en l'espai de món, ja que Arm està situat a l'origen. Si ens fixem en shoulderBoneC, que és fill de shoulder, està situat al punt local (0, 0, 1.5), però realment està situat al punt (0, 7.1, 1.5) en l'espai de món. El que és el mateix, la posició en l'espai de món serà la suma de les posicions locals del propi objecte i dels pares. Passa el mateix amb la rotació i escala.

A part de poder modificar aquests valors des de la finestra Inspector, es pot modificar amb el ratolí, des de la finestra on es visualitza l'escena amb els objectes. A la següent imatge es pot veure els 3 eixos amb els quals es pot variar la posició del braç, directament arrossegant amb el ratolí:



Imatge 6: Eixos del braç (Arm) sobre l'escena virtual

Ara que ja tenim clar el procés de creació del model, passarem a explicar la implementació de l'algoritme IK pel moviment del braç.

Per tal de definir i controlar el comportament dels objectes a Unity, s'utilitzen scripts escrits en C# (també admet JavaScript i Boo, però C# és el que s'utilitza principalment). Aquests scripts defineixen el comportament dels objectes del joc a l'especificar com interactuen amb l'entorn, com responen a una entrada del jugador, com es mouen, col·lideixen, entre altres coses. Aquests scripts s'adjunten als objectes com a components i s'executen en resposta a esdeveniments o a cada quadre de la simulació del joc (frame).

En el nostre cas, el que volem és que s'executi a cada frame, ja que és necessari anar calculant les rotacions dels segments del braç el més ràpid possible per simular un moviment continu i natural. Per crear un script nou, només hem de fer clic dret a la carpeta on el volem crear (des de Unity), i seleccionar Create > C# Script. Si l'obrim, veurem que per defecte té les importacions mínimes necessàries i una classe anomenada com haguem anomenat l'script. A la classe, que en aquest cas es diu IK_2, ja venen definides dos funcions: Start() i Update(). La funció Start() és la que es crida una vegada abans del primer frame, i la funció Update() es crida una vegada a cada frame. A la següent imatge es poden veure les importacions per defecte i la declaració de la classe:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class IK_2 : MonoBehaviour
{

    [SerializeField]
    Transform elbow;
    [SerializeField]
    Transform elbowBone;

    [SerializeField]
    Transform shoulder;
    [SerializeField]
    Transform shoulderBone;

    [SerializeField]
    Transform hand;

    [SerializeField]
    Transform target;

    private Vector3 lastTargetPos;

    private float elbowBoneLength;
    private float shoulderBoneLength;
```

Imatge 7: Importacions i declaració de la classe de l'script IK_2

Si ens fixem en les variables declarades a la classe, veurem que hi ha algunes que tenen l'etiqueta [SerializeField] a sobre. Aquesta etiqueta el que fa és permetre que una variable privada sigui visible i editable des de l'Inspector, però mantenint l'encapsulació. Hem declarat variables d'aquest tipus per emmagatzemar tots els components del braç (articulacions, ossos i mà), a més de l'objecte al qual volem arribar amb la mà (target). Aquestes variables són de tipus Transform, ja que aquest component dels objectes és el que determina la seva posició, rotació i escala. Després, hem definit tres variables privades: dos per les longituds dels ossos i una per emmagatzemar la posició del frame anterior de l'objecte target.

Passem a explicar la funció Start():

```
    void Start()
    {
        lastTargetPos = target.position;

        // BONE 1
        elbowBoneLength = Vector2.Distance(new Vector2(elbow.position.x, elbow.position.y),
        new Vector2(hand.position.x, hand.position.y));

        // BONE 2
        shoulderBoneLength = Vector2.Distance(new Vector2(shoulder.position.x, shoulder.position.y),
        new Vector2(elbow.position.x, elbow.position.y));
    }
```

Imatge 8: Funció Start() de l'script IK_2

En aquesta funció, principalment hem fet els càlculs que necessitem calcular una sola vegada. Es tracta de les longituds dels ossos. Per calcular-les, hem utilitzat les posicions dels extrems dels cilindres, que les podem obtenir a partir de les variables definides. Per la part inferior del braç, utilitzem les components x i y de la posició del colze (elbow) i de la posició de la mà. Utilitzem aquestes 2 components, ja que, de moment, volem implementar l'algoritme en 2D. Per calcular la longitud entre aquests 2 punts, utilitzem la funció Distance de Vector2. Per la part superior del braç, farem el mateix, però agafant

com a punts l'espatlla i el colze. A més, emmagatzemarem la posició del target a la variable definida lastTargetPos, de la qual hem parlat abans.

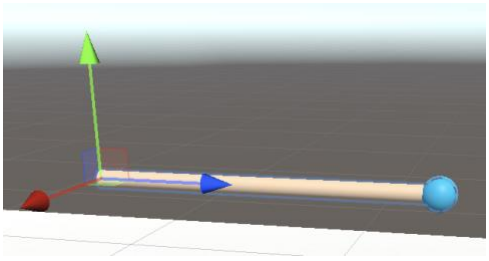
Ara, passem a explicar la funció Update():

```
42      void Update()
43      {
44          if(lastTargetPos != target.position){
45              float r = Vector2.Distance(new Vector2(shoulder.position.x, shoulder.position.y),
46              new Vector2(target.position.x, target.position.y));
47
48              if(r <= elbowBoneLength + shoulderBoneLength){
49                  float cos_beta = ((r * r) + (shoulderBoneLength * shoulderBoneLength) -
50                  (elbowBoneLength * elbowBoneLength)) / (2 * r * shoulderBoneLength);
51
52                  float beta = Mathf.Acos(cos_beta) * Mathf.Rad2Deg;
53
54                  float cos_alpha = ((elbowBoneLength * elbowBoneLength) + (shoulderBoneLength * shoulderBoneLength) -
55                  (r * r)) / (2 * elbowBoneLength * shoulderBoneLength);
56
57                  float alpha = Mathf.Acos(cos_alpha) * Mathf.Rad2Deg;
58
59                  Vector2 diff = new Vector2(target.position.x, target.position.y) -
60                  new Vector2(shoulder.position.x, shoulder.position.y);
61
62                  float gamma = Mathf.Atan2(diff.y, diff.x) * Mathf.Rad2Deg;
63
64                  float q1 = gamma - beta;
65                  float q2 = 180 - alpha;
66
67                  shoulder.localRotation = Quaternion.Euler(-q1, 90f, 0f);
68                  elbow.localRotation = Quaternion.Euler(-q2, 0f, 0f);
69
70              } else{
71                  shoulder.LookAt(target);
72                  elbow.LookAt(target);
73              }
74          }
75      }
```

Imatge 9: Funció Update() de l'script IK_2

En aquesta funció, que s'executarà una vegada a cada frame, es fan tots els càlculs que hem detallat a l'apartat anterior per obtenir els angles de rotació de les articulacions del braç. Per tal d'optimitzar una mica la funció, el que fem primer de tot es comprovar si el target ha variat la seva posició respecte l'anterior frame (línia 46), ja que sinó, la posició serà la mateixa i no farà falta cap càlcul. A continuació, el que fem és comprovar si el target està situat a una posició més allunyada que la longitud del braç totalment estirat. En cas afirmatiu, cridarem a la funció LookAt(), tant de l'espatlla com del colze, passant per paràmetre l'objecte target. El que farà, doncs, és apuntar el transform del qual es crida la funció cap a l'objecte passat per paràmetre, de manera que, si tant l'espatlla com el colze apunten al target, el braç quedarà totalment estirat i apuntant cap a ell (bloc else de la línia 72). En cas contrari, procedirem a fer els càlculs pertinents. Cal dir que hem utilitzat els mateixos noms per les variables que als esquemes explicats al punt anterior. La distància de l'espatlla al target, que hem calculat, l'hem guardat a una variable anomenada r. Seguidament, calcularem el cosinus de β per trobar β, i el cosinus d'α per trobar α (línies 49 – 57), tal i com hem explicat a l'apartat anterior. Després, passem a calcular q_1 i q_2. Per calcular q_1, necessitem calcular també l'angle γ, que ho farem a partir del vector que va de l'espatlla al target, calculant l'arctangent de l'angle rectangle que forma (línies 59 – 62). La diferència de γ i β es correspondrà amb l'angle q_1, i la diferència de Π i α es correspondrà amb l'angle q_2. Finalment, hem d'assignar les rotacions (locals) de les articulacions (línies 67 i 68). Perquè tingui el comportament esperat, hem hagut de mantenir la rotació tant en l'eix Y com en el Z de l'espatlla i del colze, ja que, en el nostre

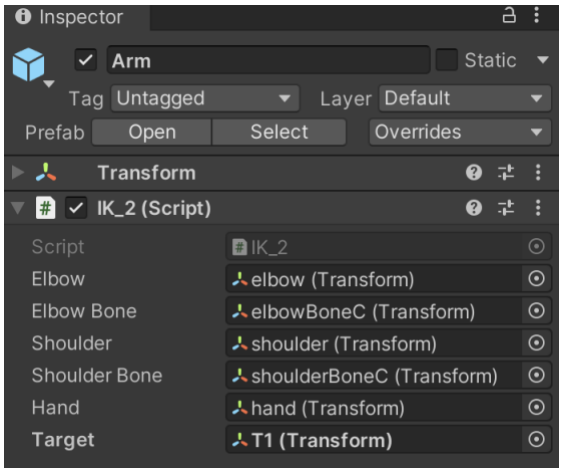
cas, les articulacions han de rotar sobre l'eix X. A més, hem hagut de canviar el signe dels angles q_1 i q_2 per obtenir el moviment esperat.



Imatge 10: Direccions dels eixos locals de l'espatlla (shoulder)

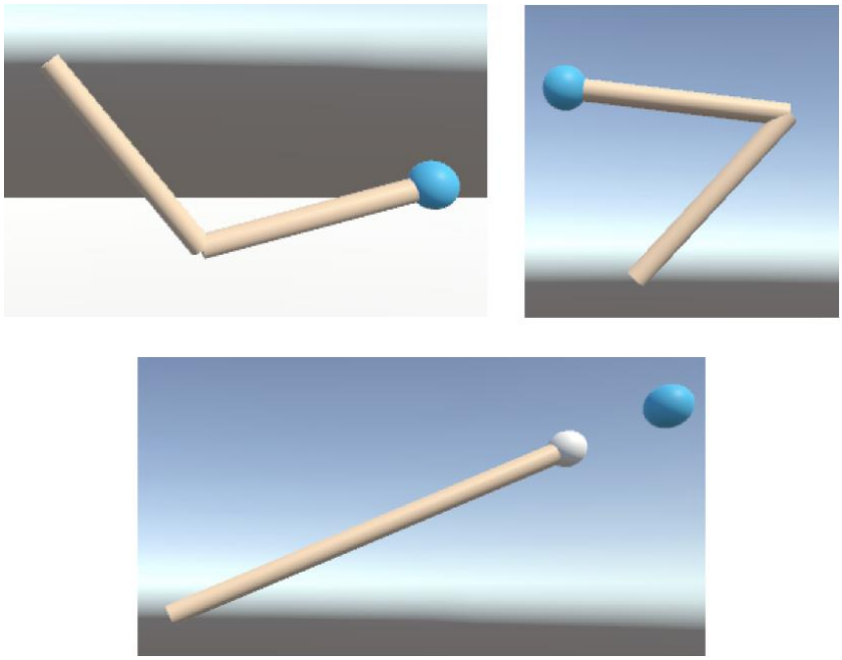
A la imatge de dalt es mostren les direccions dels eixos locals de l'espatlla, que seran les mateixes pel colze. El verd es l'eix local Y, el blau és el Z i el vermell el X. És per això que, en el nostre cas, l'eix sobre el qual han de rotar les articulacions és el X.

Una vegada tenim el script, el que hem de fer és adjuntar-lo al model del braç com un nou component. Seleccionem l'objecte Arm, que encapsula tot el braç, afegim un nou component des de l'Inspector i seleccionem l'script IK_2. Ara, veurem com apareixen els camps de les variables que tenien l'etiqueta [SerializeField]. Procedim a arrossegar o seleccionar els objectes corresponents i ja hauria de funcionar. Per l'objecte target, hem afegit a l'escena una esfera de les mateixes dimensions que la mà, anomenada T1. A la següent imatge podem veure com quedaria el component que hem afegit a Arm, amb tots els objectes seleccionats:



Imatge 11: Component IK_2 de l'objecte Arm

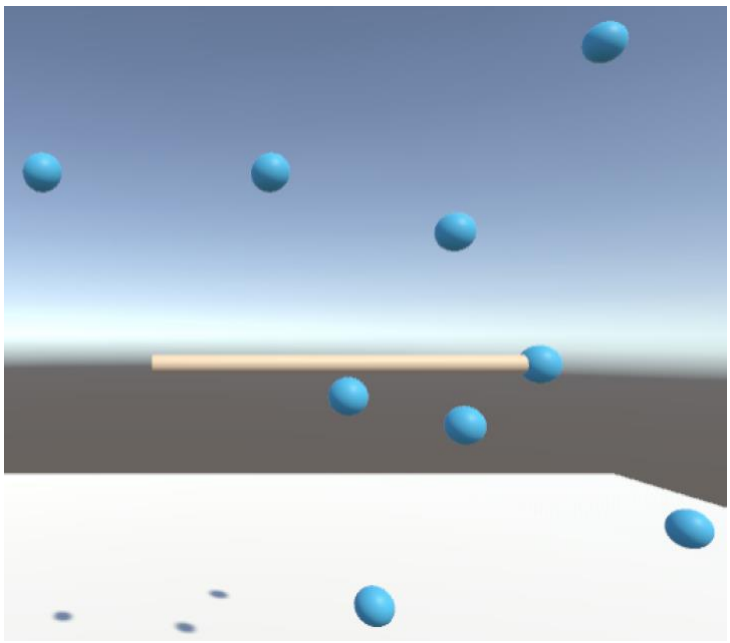
Ara, si provem a executar, podem moure l'esfera T1 i, en conseqüència, el braç es posicionarà de manera que la mà sempre es trobi el més a prop seu. A les següents imatges es pot visualitzar el moviment del braç respecte la posició de T1, en 3 configuracions diferents:



Imatge 12: Moviment del braç en 3 configuracions diferents

A la última imatge es pot veure com T1 es troba massa lluny com per poder arribar amb la mà i el que fa llavors, és apuntar amb el braç totalment estirat cap a l'objecte.

A continuació, el que anem a fer és construir una mena d'animació, per tal que, a l'executar el joc, l'esfera T1 es mogui sola i, per conseqüència, el braç també ho faci. Per fer-ho, el que hem fet ha sigut crear un objecte buit, anomenat Animation, en el qual hem afegit 9 rèpliques de T1 i les hem posicionat en llocs arbitraris de l'escena, en el mateix pla vertical en el qual es mou el braç. A la següent imatge es pot veure com estan posicionades totes les esferes target:



Imatge 13: Esferes d'Animation a l'escena

Perquè no es vegin les esferes, hem desactivat l'objecte Animation des de l'Inspector. La següent cosa que hem fet ha sigut crear un script per tal de moure T1 a cadascuna de les posicions de les esferes d'Animation, cada cert temps i amb una mena de retard, per tal de donar l'aspecte d'una animació. L'hem anomenat animation_target, i és el següent:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class animation_target : MonoBehaviour
{

    [SerializeField]
    Transform T1;

    [SerializeField]
    Transform[] targets;

    public float duration = 2.0f;
    private int currentTargetIndex = 0;
    private float startTime;
    // Start is called before the first frame update
    void Start()
    {
        startTime = Time.time;
    }
    // Update is called once per frame
    void Update()
    {
        float timeElapsed = Time.time - startTime;
        float t = Mathf.Clamp01(timeElapsed / duration);

        T1.position = Vector3.Lerp(T1.position, targets[currentTargetIndex].position, t);

        if (t >= 0.25f) {
            currentTargetIndex = (currentTargetIndex + 1) % targets.Length;
            startTime = Time.time;
        }
    }
}
```

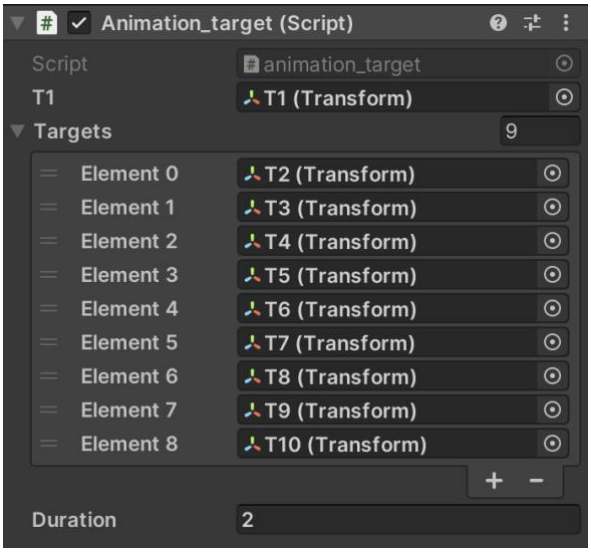
Imatge 14: Script animation_target

Hem declarat una variable de tipus Transform pel target que volem moure, en aquest cas T1. També hem declarat una altra variable per emmagatzemar totes les demés esferes de l'animació: es tracta d'una array de tipus Transform anomenada targets. Ambdues tenen l'etiqueta [SerializeField]. Després, hem declarat la variable pública duration per establir una duració per l'animació, per defecte amb valor de 2. Finalment, tenim dos variables privades: currentTargetIndex, per tal d'indicar l'índex de l'array de targets en el qual ens trobem, i startTime, per tal d'emmagatzemar el temps d'inici de cada animació.
A la funció Start(), inicialitzem la variable startTime amb Time.time, que indica el temps transcorregut des de l'inici de l'aplicació.

A la funció Update(), calculem el temps que ha transcorregut des de l'inici de l'animació amb la diferència del temps actual i el del startTime, i el guardem a una variable anomenada timeElapsed. Es calcula a continuació un valor t, normalitzat entre 0 i 1, dividint el temps transcorregut per la duració de l'animació, utilitzant Math.Clamp01 per

assegurar-se que el valor es trobi a dins del rang esmentat. A la línia 28 s'actualitza la posició de T1 utilitzant la funció Vector3.Lerp, que realitza una interpolació lineal entre la posició actual de T1 i la posició de l'objectiu actual de l'array de targets, utilitzant el valor de t com a factor d'interpolació. Gràcies a aquesta interpol·lació, T1 no canvia de posició directament a la del següent target, sinó que és progressiu. Després, si el valor de t és superior a 0.25f, s'actualitzen els valors de currentTargetIndex pel següent (si ens trobem a l'últim, es reinicia) i de startTime pel temps actual de l'aplicació.

Hem creat un component a Arm amb aquest script, tal i com hem fet abans pel IK, i hem omplert els camps amb els corresponents objectes. D'aquesta manera, quan executem el joc i tenint activat el component de l'animació, veurem com el braç es va movent seguint T1, que anirà variant la seva posició respecte el temps, de manera cíclica per totes les esferes invisibles de l'objecte Animation.



Imatge 15: Component animation_target d'Arm

A la imatge de dalt es pot veure el component creat a partir del script animation_target amb els camps omplerts. Si ens fixem, podem canviar el valor de duration, de manera que podrem variar el temps que triga a moure's T1 entre una posició i una altra, amb la qual cosa el braç també es mourà a una velocitat proporcional.

4.1.3 Moviment en 3D

Una vegada vam arribar a desenvolupar la creació del braç i la implementació de l'algorisme IK en 2D pel seu moviment, el següent repte que ens vam plantejar era implementar-ho perquè el braç es pugui moure en tot l'espai 3D. Ens vam documentar per saber com es podria implementar a partir del que teníem. En aquest punt vam veure que la complexitat que tenia implementar-ho en 3D era massa alta pel temps que disposàvem, ja que després també voldríem poder aprendre a interactuar, amb les ulleres i els comandaments VR, amb altres objectes de l'escena. Llavors, vam decidir que explicaríem el procés teòric per passar a la tercera dimensió, però la seva implementació la deixaríem com a pla de futur del projecte. Dit això, explicarem unes pinzellades del procés que hauríem de realitzar per passar a la tercera dimensió.

Primer de tot, hem de destacar que el problema que teníem en 2D tenia 2 graus de llibertat, ja que havíem de controlar 2 variables, en el nostre cas els angles de les articulacions. En estendre a la tercera dimensió, el problema passa a tenir 3 graus de llibertat, sempre i quan assumim que podem rotar lliurement el primer segment (part superior del braç). En canvi, el segon segment o part inferior del braç, queda de la mateixa manera que abans. Normalment es fa aquesta aproximació, que se'n diu braç robòtic "Leg-like", en la qual els braços tenen les articulacions dels malucs i dels genolls (els malucs poden rotar gairebé lliurement i els genolls només sobre un eix). Es fa així perquè, si les dos articulacions poguessin rotar lliurement, hi hauria 4 graus de llibertat. Llavors, el problema tindria una complexitat molt més elevada.

Com havíem explicat pel moviment en 2D, hi havia 2 possibles solucions pel mateix escenari. Ara, pel moviment en 3D, les solucions són infinites, ja que es pot rotar sobre l'eix de simetria representat per la recta que connecta l'origen del braç (o espatlla) amb el target, i seguiria sent igual de vàlida la solució. Ens hem de fixar en una de les solucions, com per exemple una que assumeixi el vector (0, 1, 0) com la direcció cap a dalt.

Llavors, podríem resoldre, conceptualment, el problema en 3 passos:

1. Rotar el punt del target sobre l'eix Y fins que quedi sobre el pla XY.
2. Moure el braç per assolir la posició del target com si fos en 2D.
3. Desfer la primera rotació rotant el braç sencer en la direcció oposada sobre l'eix Y.

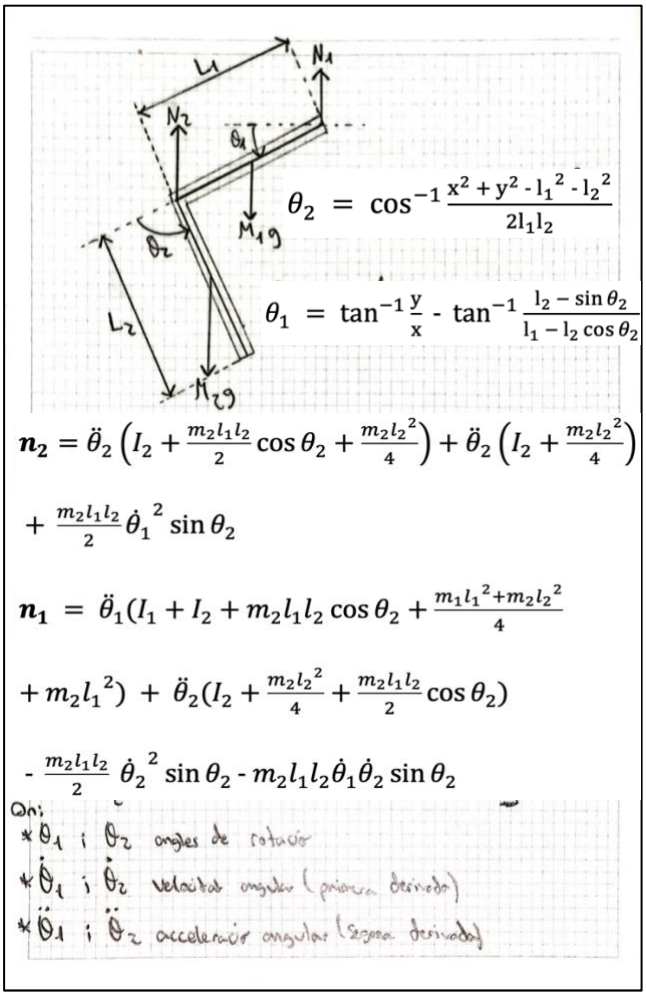
4.1.4 Torques

Abans de començar a estudiar com interactuar amb altres objectes d'una escena amb les ulleres VR, volem explicar una última cosa que hem estudiat respecte a les rotacions de les articulacions, encara que no ho hem acabat implementant.

El torque és una mesura de la força que actua sobre un objecte per rotar-lo al voltant d'un eix en específic, és a dir, és la força que s'aplica a un objecte per fer-lo girar. També se'l coneix com moment de força, i les seves unitats són Newton * metre en el Sistema Internacional.

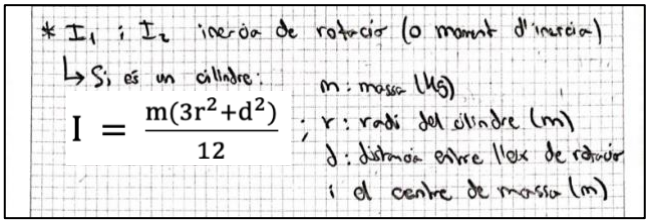
En el context del nostre escenari, d'un braç de 2 segments i 2 articulacions, el torque s'aplica a cadascuna d'elles i determina la capacitat del braç per moure's o per mantenir una posició específica. El torque depèn de diversos factors, com la longitud de cada segment, la força aplicada i la posició relativa de les articulacions. Quan s'aplica una força en el braç, el torque resultant dependrà de la distància entre el punt d'aplicació de la força i el punt de rotació de cada articulació. Si la força s'aplica més lluny del punt de rotació, es generarà un major torque i es facilitarà el moviment del braç.

A la següent imatge es mostra un esquema del braç de 2 segments amb cadascun dels torques de les articulacions, a més de les fórmules per calcular-los:



Imatge 16: Esquema i càlcul dels valors de torque d'un braç de 2 segments

Si ens fixem bé, θ_1 i θ_2 es correspondrien amb els angles q_1 i q_2 anteriors, que serien els angles de rotació. N_1 i N_2 serien els torques o força de rotació. Les equacions dinàmiques per obtenir les expressions dels torques N_1 i N_2 de l'espatlla i del colze, poden obtenir-se aplicant directament la cinemàtica, les equacions de Newton-Euler i el principi de d'Alembert. Cal aplicar la cinemàtica per obtenir les velocitats i acceleracions angulars, les equacions de Newton-Euler per relacionar les forces i els moments amb els moviments angulars, i el principi d'Alembert per equilibrar les forces i els moments aplicats. Cal esmentar que I_1 i I_2 són els moments d'inèrcia de cada segment, que depèn de la massa i la geometria de l'objecte. En el nostre cas, com hem utilitzat cilindres pels segments del braç, es podria calcular de la següent manera:



Imatge 17: Càlcul de la Inèrcia de rotació d'un cilindre que rota sobre un dels extrems

Per la mateixa raó que a l'apartat anterior, no hem implementat això al nostre projecte de Unity, ja que no disposàvem del temps necessari per fer-ho i realitzar-ne els estudis

pertinents. Per això vam decidir deixar-ho també com a pla de futur. Són diversos els estudis que es podrien realitzar sobre les forces de rotació, entre els quals tenim algunes idees:

- Anàlisis de la dinàmica del braç: es pot estudiar com els valors de torque varien en funció del temps, a mesura que el braç es mou.
- Optimització del control del braç: es pot estudiar com ajustar els valors de torque en funció de diferents objectius de control, com el seguiment d'una trajectòria, el manteniment d'una posició i la minimització de l'esforç o eficiència energètica (com afecta al consum d'energia).

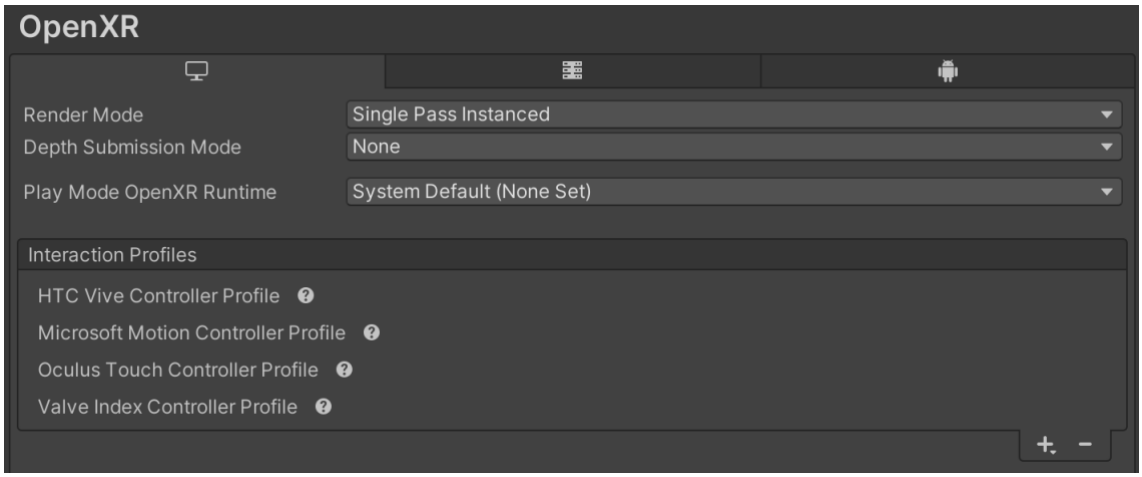
4.2 Creació del joc de Realitat Virtual

En aquest apartat, explicarem tot el procés que hem dut a terme per tal de crear un petit joc de Realitat Virtual. Explicarem diverses maneres que hi ha d'interactuar amb els diferents elements d'una escena, a través de les ulleres i comandaments de VR.

4.2.1 Configuració del projecte

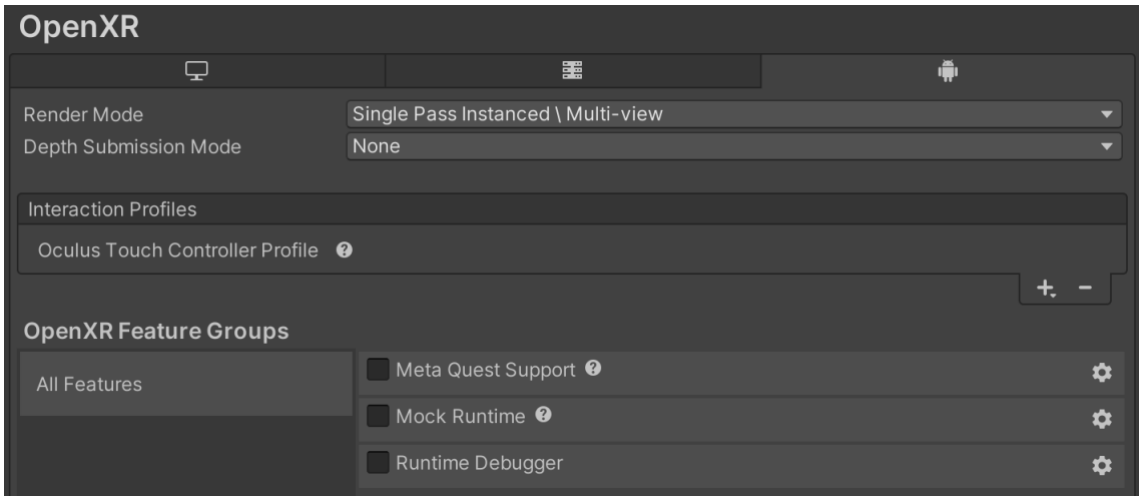
Crearem un nou projecte de Unity. Ens hem instal·lat un nou editor i l'hem afegit els mòduls Android Build Support i Windows Build Support, per tal de poder compilar i generar aplicacions i executables per ambdues plataformes (a més de l'opció en macOS, que és el Sistema Operatiu que estem utilitzant). La versió de l'editor que hem utilitzat en aquest cas és la 2021.3.25f1 (que ens la recomanava el propi Unity). A l'hora de crear el projecte, hem seleccionat la plantilla 3D (URP). Aquesta plantilla està dissenyada per proporcionar un renderitzat eficient i flexible per jocs en 3D, oferint un rendiment optimitzat i compatibilitat multiplataforma.

Una vegada creat el projecte, crearem una escena amb un pla que simuli el sòl. Cal instal·lar el connector XR Plug-In Management des de la Configuració del Projecte. Cal seleccionar les opcions d'Open XR tant a l'Escriptori com a l'Android. Ara, el que hem de fer és seleccionar el Perfil d'Interacció d'Open XR: a l'Escriptori cal afegir els Perfils de Controlador que es mostren a la següent imatge:



Imatge 18: Perfils de Controlador de OpenXR a l'Escriptori

I en Android el perfil Oculus Touch Controller Profile:



Imatge 19: Perfils de Controlador de OpenXR en Android

D'aquesta manera, tindrem compatibilitat amb la majoria dels dispositius de realitat virtual.

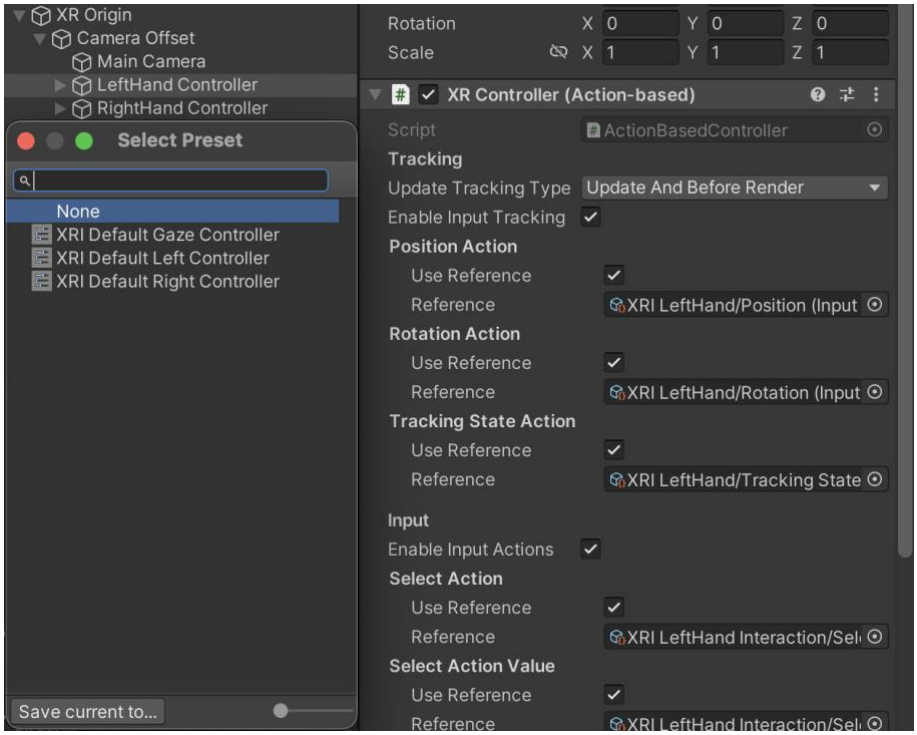
Per agilitzar el procés de creació del joc, utilitzarem l'eina XR Interaction Toolkit, que s'instal·la des de Window -> Package Manager -> Add package by name -> com.unity.xr.interaction.toolkit.

Ara, cal esborrar la càmera principal que hi ha a l'escena (main camera) i afegir un element XR Origin, que ja tindrà configurada la càmera per moure's amb les ulleres.

Cal canviar el Mode d'Origen de Seguiment (Tracking Origin Mode) de l'XR Origin a Floor per tenir en compte l'altura del personatge, com es mostra a la següent imatge.

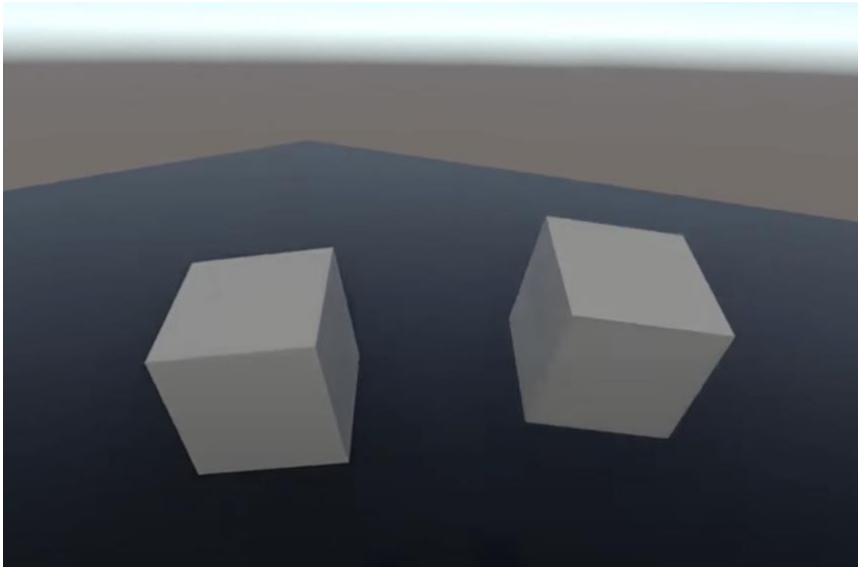
Imatge 20: Component XR Origin del'objecte XR Origin

Per defecte, en afegir l'XR Origin, ja ens han vingut els Hand Controllers, però si no és així, cal crear 2 buits (empty) dins del Camera Offset amb el component XR Controller (Action-Based). Per configurar totes les Accions d'Entrada (Input Actions) dels Hand Controllers sense fer-ho manualment, cal importar des de l'XR Interaction Toolkit (en el Package Manager) els Starter Assets i seleccionar en el component que hem afegit el preajustament (preset) corresponent per a cada mà: XRI Default Left Controller o XRI Default Right Controller:



Imatge 21: Configuració del preset del LeftHand Controller

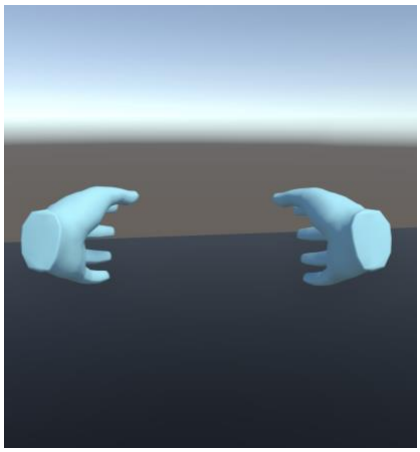
A continuació, cal dir-li a Unity que activi l'acció que estem utilitzant. Ho farem afegint el component Input Action Manager a l'XR Origin i arrossegant a l'atribut Action Assets el paquet XRI Default Input Actions del Starter Assets que hem importat prèviament. D'aquesta manera, ara podem comprovar que les mans es mouen en executar. Per veure-ho millor, cal afegir un objecte 3D als Hand Controllers, com pot ser un cub, una esfera o un model que tinguem (mans).



Imatge 22: Execució del joc en la qual es comprova que els Hand Controllers es mouen amb els comandaments

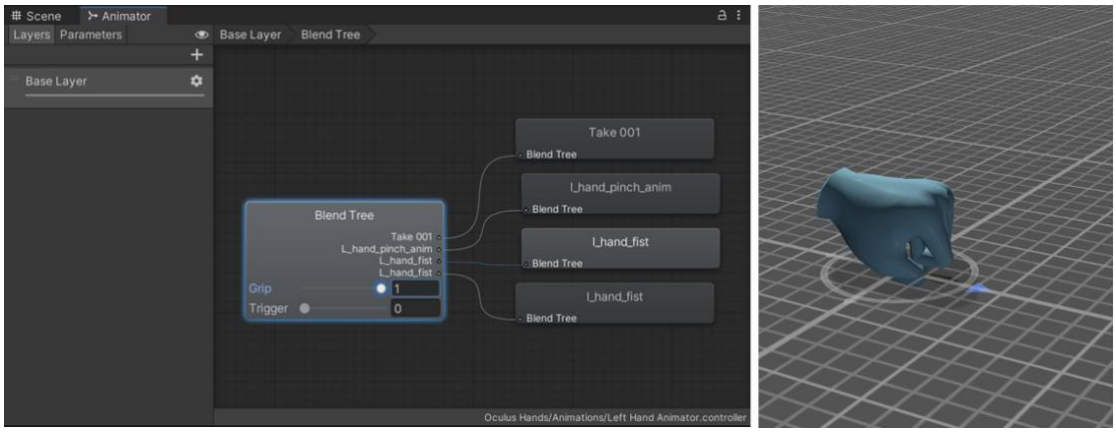
4.2.2 Entrada i presència de les mans

En aquesta part aprendrem com escoltar les entrades de la realitat virtual (VR) i utilitzar-les per executar animacions a les mans. Descarreguem un paquet d'Oculus que conté els models i les animacions per a les mans: model-mans-VR
Un cop importat, podem substituir els cubs que simulaven les mans pels models de les mans del paquet descarregat:



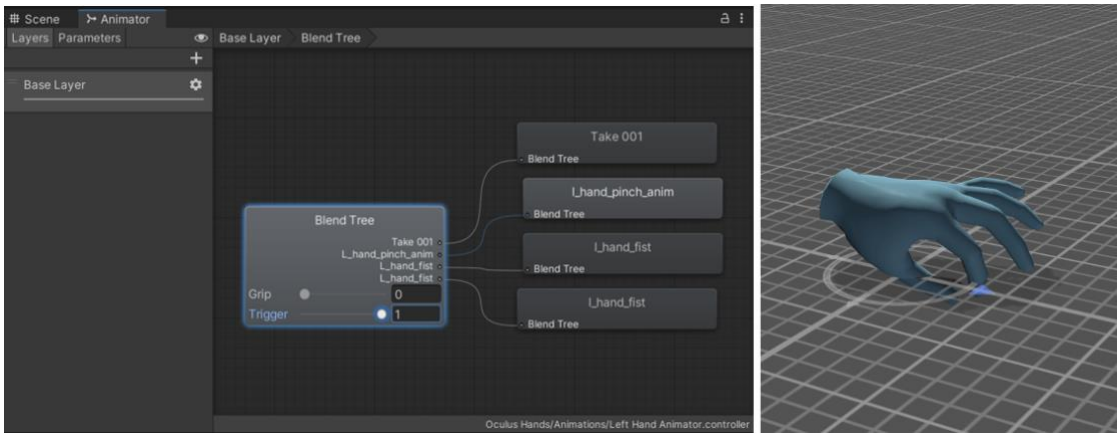
Imatge 23: Execució del joc en la qual s'han assignat models de mans pels Hand Controllers

El que volem fer ara és que, en prémer els botons dels comandaments, s'executin accions. Si seleccionem un dels models de les mans i anem a la finestra Animator, podem accedir a Blend Tree i provar com funcionen les animacions que hi ha, que són 2: una amb el botó Grip i una altra amb el botó Trigger.



Imatge 24: Animació del model de la mà en prémer el botó Grip

A les imatges de dalt es pot visualitzar el final de l'animació del botó Grip, que està representat per un valor màxim que oscil·la entre 0 i 1.



Imatge 25: Animació del model de la mà en prèmer el botó Trigger

En aquestes segones imatges, es pot veure el final de l'animació del botó Trigger, que també és representat per un valor màxim que oscil·la entre 0 i 1.

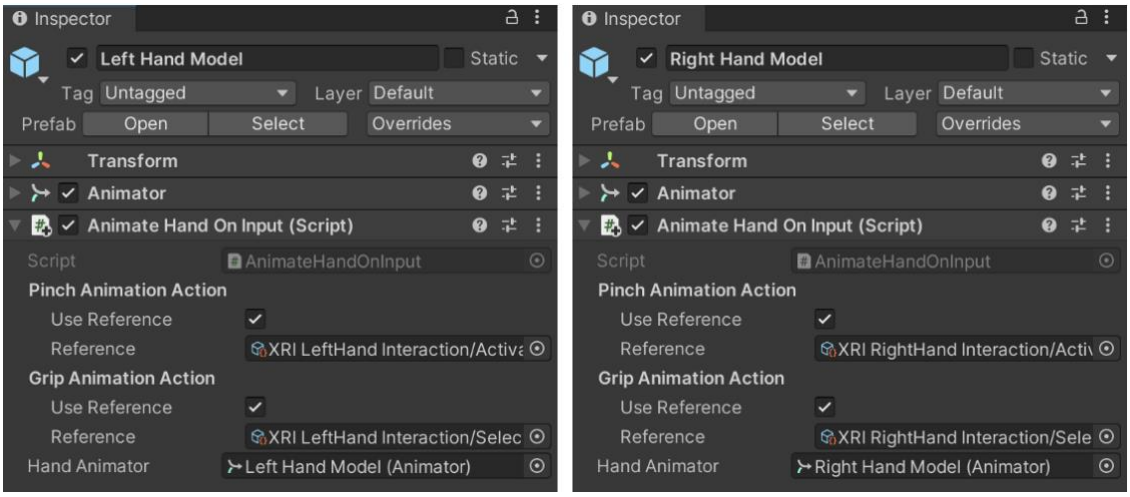
Així doncs, perquè les accions s'executin en prémer els botons, el que hem de fer és seleccionar un dels models de les mans, en aquest cas el de l'esquerra, i crear un nou component anomenat "AnimateHandOnInput", que crearà un script amb aquest nom.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class AnimateHandOnInput : MonoBehaviour
{
    public InputActionProperty pinchAnimationAction;
    public InputActionProperty gripAnimationAction;
    public Animator handAnimator;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // con esto leeremos el valor de apretar el boton trigger
        // y se lo asignaremos al handAnimator
        float triggerValue = pinchAnimationAction.action.ReadValue<float>();
        handAnimator.SetFloat("Trigger", triggerValue);

        // con esto leeremos el valor de apretar el boton grip
        // y se lo asignaremos al handAnimator
        float gripValue = gripAnimationAction.action.ReadValue<float>();
        handAnimator.SetFloat("Grip", gripValue);
    }
}
```

Imatge 26: Script del component creat AnimateHandOnInput

El que farem és definir una sèrie d'accions en l'script i seleccionar l'acció adequada des del paquet d'accions importat des de "Starter Assets". Les accions les hem anomenat "pinchAnimationAction" i "gripAnimationAction". També definirem un model de tipus

Animator (handAnimator). Aleshores, en el mètode "Update()", que s'executa en cada frame, recollirem els valors de prémer els botons Trigger i Grip del comandament (línies 23 i 28), que es corresponen amb els valors que hem mencionat abans en mostrar les animacions. Després, assignarem aquests valors a l'animator i, d'aquesta manera, en prémer el botó Trigger es visualitzarà l'animació corresponent (línies 24 i 29).
Finalment, afegirem el mateix script a l'altre model i omplirem els camps de les variables del script per a cada mà, tal com es mostra a les imatges següents.

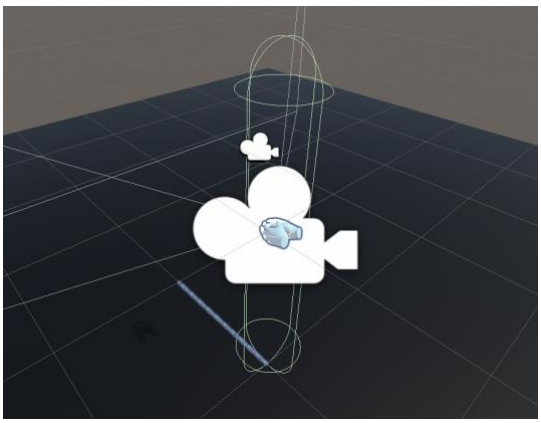


Imatge 27: Components AnimateHandOnInput dels models de les dos mans

4.2.3 Moviment continu

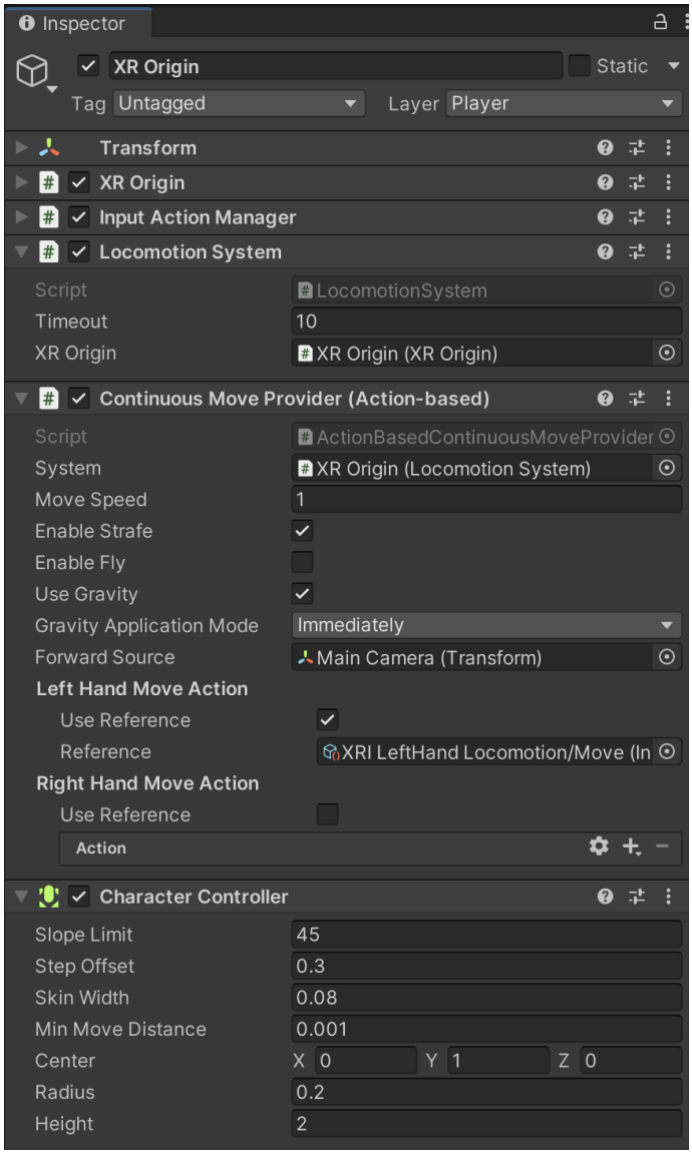
En aquest apartat farem les configuracions per realitzar un moviment continu. Per fer-ho, hem d'afegir alguns components al nostre XR Origin. Primerament, afegirem el component Locomotion System, que rebrà la sol·licitud de moviment que fem per desplaçar-nos. Afegirem també el component Continuous Move Provider (que fa que es mogui en la direcció de la càmera, és a dir, del casc), que és el component principal que escolta la nostra entrada per moure'ns. Finalment, afegirem també el Character Controller, on tindrem la configuració per saber com reaccionarà el nostre personatge davant un obstacle. Per exemple, l'atribut Slope Limit indica els graus màxims pels quals el nostre personatge pot escalar. Reduirem el valor de l'atribut Radius a 0.2 per percebre un moviment més precís i col·lidir amb els altres objectes a una distància menor. També posicionarem el centre en el punt (0, 1, 0), perquè el nostre personatge quedi sobre el pla i no caigui.

A la següent imatge es pot veure com quedaria la "càpsula" del nostre Character Controller:



Imatge 28: Càpsula del Character Controller

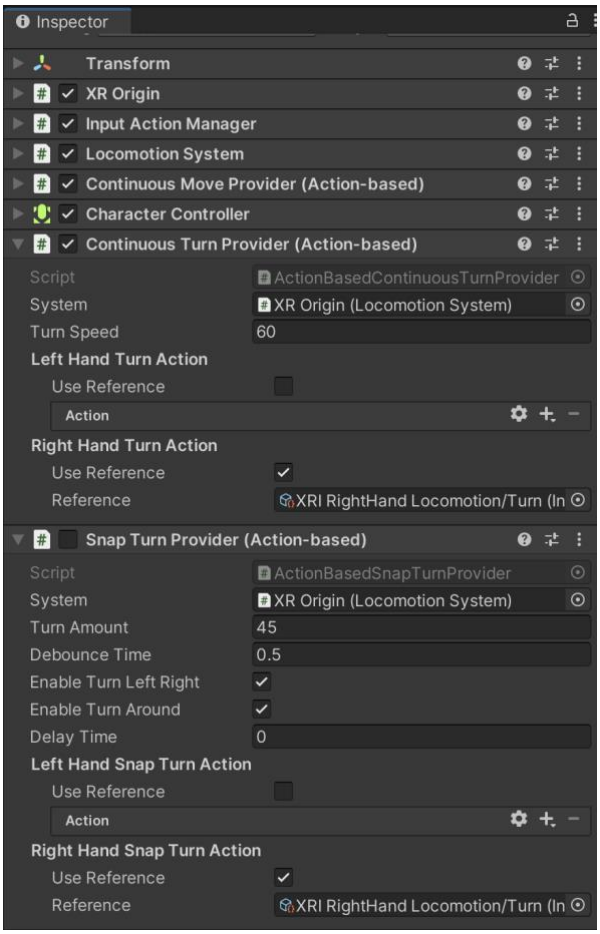
Les configuracions dels 3 components quedarien de la següent manera:



Imatge 29: Components Locomotion System, Continuous Move Provider i Character Controller de l'objecte XR Origin

Si ens fixem en la configuració del Continuous Move Provider, podem veure que podem canviar la velocitat de moviment, utilitzar o no la gravetat i, en cas afirmatiu, si seleccionem l'opció Attempting Move al camp Gravity Application Mode la gravetat no ens afectarà fins que comencem a moure'ns. Per això hem seleccionat l'opció Immediately, que farà que ens afecti immediatament en executar el joc. També podem destacar el Forward Source, que farà que el nostre personatge es mogui en la direcció de l'element que li assignem quan intentem avançar cap endavant. En aquest cas, hem assignat la Main Camera del nostre XR Origin perquè es mogui cap on el personatge estigui mirant. Finalment, en el camp Left Hand Move Action, assignarem l'acció per moure'ns del paquet descarregat XR Interaction Toolkit, és a dir, l'XRI LeftHand Locomotion/Move. Amb això, el nostre personatge ja es pot moure sobre el pla amb el comandament esquerre.

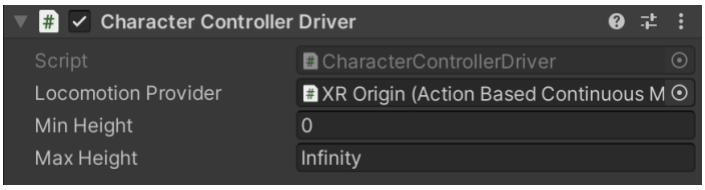
Ara configurarem la rotació, que la farem amb el comandament dret. Es pot escollir entre una rotació contínua i una rotació que gira una quantitat de graus d'un sol cop. En el futur, el jugador hauria de poder escollir entre una configuració o una altra, o depenent del joc que fem utilitzar una de les dues. Per fer-ho, afegirem els components Continuous Turn Provider i Snap Turn Provider. La configuració d'aquests components és molt semblant a la del Continuous Move Provider: cal indicar el Locomotion System i assignar-li l'acció apropiada a la mà o mans que vulguem. En aquest cas, com que la mà esquerra ja s'encarrega del moviment, li assignarem l'acció només a la mà dreta, és a dir, l'XRI RightHand Locomotion/Turn.



Imatge 30: Components Contiuous Turn Provider i Snap Turn Provider de l'objecte XR Origin

Si ens fixem en els components, podem veure que es pot modificar la velocitat de rotació en el Continuous Turn Provider i, en el Snap Turn Provider, els graus que volem girar d'un sol cop (Turn Amount).

Finalment, afegirem el component Character Controller Driver per solucionar el problema que es produeix quan movem el cap (headset) i la càpsula (Character Controller) es manté immòbil a menys que la moguem amb els comandaments. Dit d'una altra manera, gràcies a aquest component, ara la càpsula s'adapta al cap, tant en altura com en posició sobre el pla.

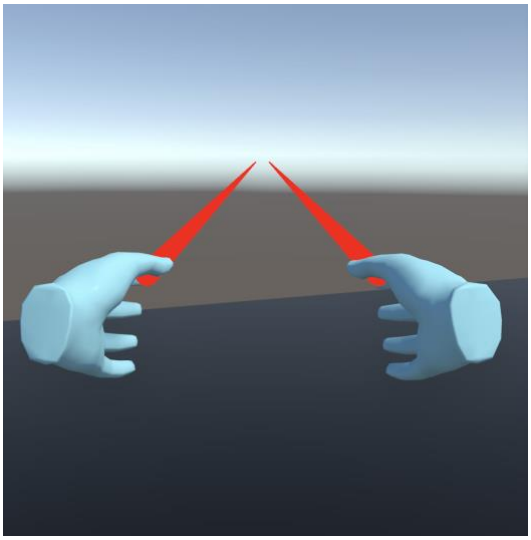


Imatge 31: Component Character Controller Driver de l'objecte XR Origin

A la imatge de dalt podem veure la configuració del component, en la qual hem de seleccionar el component Continuous Move Provider com a proveïdor de locomoció (Locomotion Provider).

4.2.4 Teletransportació

En aquesta part s'explica com implementar la teletransportació amb els comandaments. Per fer-ho, necessitem afegir al nostre XR Origin > Camera Offset, 2 XR > Ray Interactor (Action Based), un per a cada mà. Els anomenarem Left Teleportation Ray i Right Teleportation Ray. En el component XR Controller de cadascun, seleccionarem el preset corresponent a LeftHand o RightHand, tal com ja havíem fet anteriorment per als Hand Controllers. Ara, quan executem, veuríem dos raigs vermells que surten de les mans, encara que de moment no realitzen cap acció.

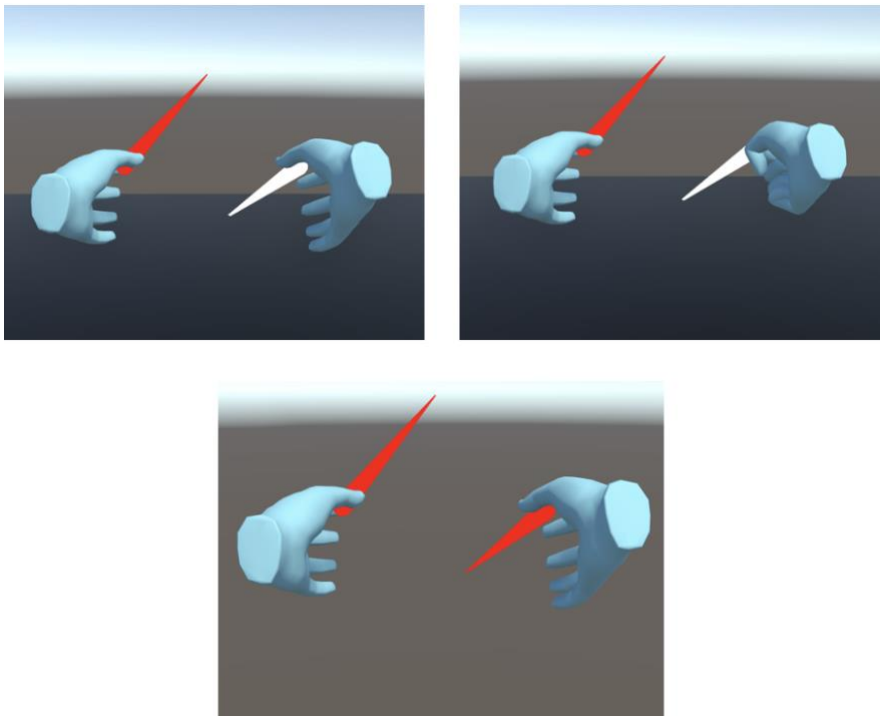


Imatge 32: Raigs d'interacció dels comandaments

Ara afegirem el component Teleportation Provider a l'XR Origin, i seleccionarem com a atribut System el Locomotion System que vam afegir anteriorment. Només ens falta

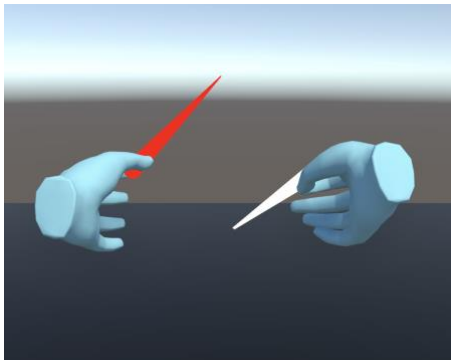
indicar l'àrea de teletransportació. Hi ha 2 tipus: Teleportation Area i Teleportation Anchor.

Pel primer tipus, seleccionarem el pla i li afegirem el component Teleportation Area. A la llista de col·lisionadors, afegirem el Mesh Collider del mateix pla. Ara ja hauríem de poder teletransportar-nos a qualsevol punt del pla al qual apuntem i premem el botó Grip de qualsevol dels comandaments.



Imatge 33: Execució del joc en la qual es mostra com el jugador pot teletransportar-se amb el Grip Button

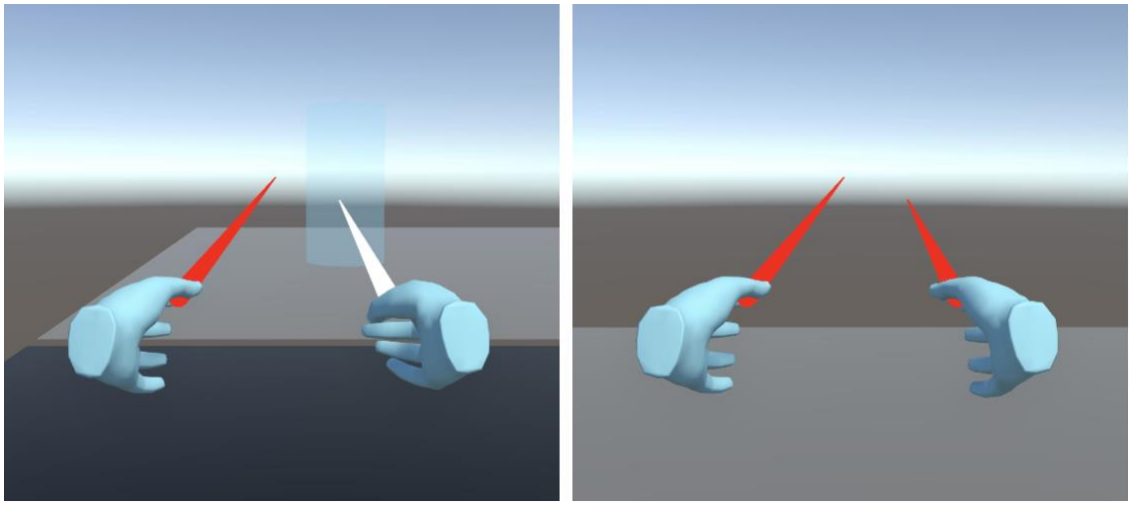
Si volem canviar el botó amb el qual ens teletransportarem, només hem de canviar la referència d'acció d'entrada (InputActionReference) de l'acció Select per l'acció Interaction/Activate (en comptes de l'acció Interaction/Select, que és la que ve per defecte) als Teleportation Ray en el seu component XR Controller. Ara, per teleportar-nos, utilitzarem el botó Trigger (Trigger Button).



Imatge 34: Execució en la qual es mostra com s'ha canviat el botó per teletransportar el jugador al Trigger Button

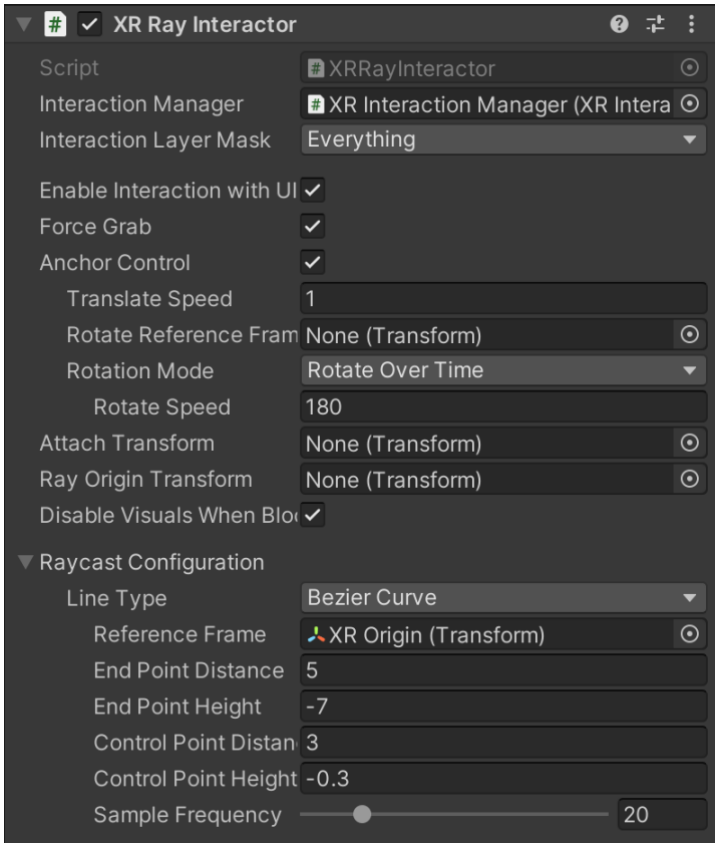
Pel segon tipus, crearem un objecte buit anomenat Teleportation Anchor en un altre pla i li afegirem un cilindre que posicionarem en el punt local (0, 1, 0), de manera que el centre de massa de l'objecte Teleportation Anchor estigui situat a la part inferior del

cilindre. Canviarem el material del cilindre per un de nou semitransparent (utilitzant el shader Universal Render Pipeline/Unlit). A continuació, afegirem a l'objecte Teleportation Anchor el component Teleportation Anchor i afegirem el cilindre a la seva llista de col·lisionadors. És important fixar-se en l'atribut Teleport Anchor Transform, que en aquest cas seria l'objecte que hem creat Teleportation Anchor, ja que aquí s'indica on ens teletransportarem. Duplicarem l'objecte Teleportation Anchor i el col·locarem separat de l'altre. Ara, en intentar teletransportar-nos a qualsevol dels cilindres (apuntant-lo i prement el botó Trigger), ens teletransportarem just a la posició del cilindre. Si volem indicar també la rotació en teletransportar-nos, podem seleccionar, en el component Teleportation Anchor, l'opció Target Up and Forward de l'atribut Match Orientation. D'aquesta manera, quan ens teletransportem al cilindre, la nostra càmera apuntarà en la direcció de l'eix Z, tot i que ho deixarem com estava. A les següents imatges podem veure com ens podem teletransportar a la posició d'un dels cilindres.



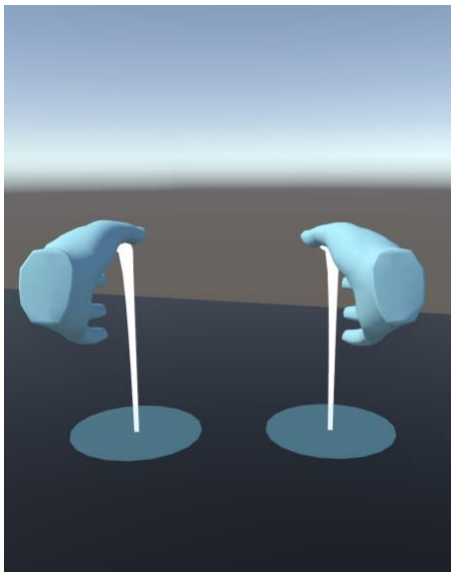
Imatge 35: Execució en la qual es mostra com el jugador pot teletransportar-se a la posició d'un dels cilindres

El que farem ara és canviar l'aparença dels raigs de teletransportació perquè siguin corbats i apuntin cap avall (d'aquesta manera, serà més fàcil apuntar a la posició desitjada). Ho farem indicant la següent configuració de Raycast Configuration en el component XR Ray Interactor de cada raig:



Imatge 36: Component XR Ray Interactor dels Teleportation Ray

Per donar-li una aparença encara millor, crearem un cilindre pla en cada raig (com un posavasos), eliminarem els seus components Capsule Collider, canviarem el material pel mateix material que hem utilitzat per als Teleportation Anchor (semi-transparent) i el desplaçarem a l'atribut Reticle del component XR Interactor Line Visual de cada raig. D'aquesta manera, el raig de teletransportació tindrà aquest cilindre pla al final del raig, a la posició de la seva col·lisió amb l'objecte, indicant de manera més clara la posició de teletransportació. A la següent imatge es pot visualitzar el resultat:



Imatge 37: Raigs de teletransportació actualitzats

4.2.5 Suspendre, agafar i utilitzar objectes interactius

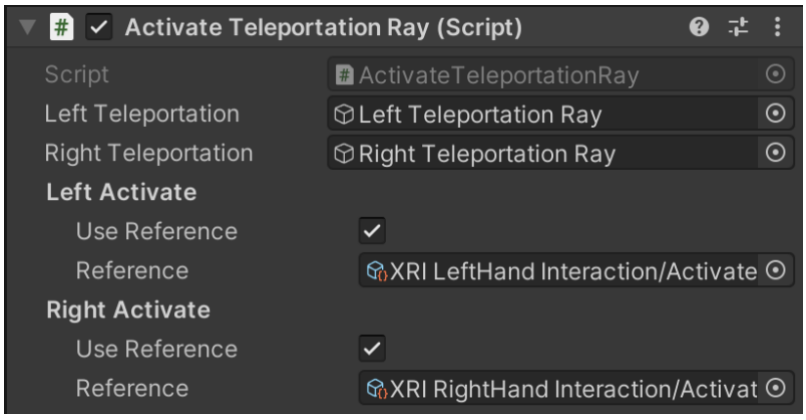
En aquesta part ensenyarem a com interactuar amb diferents objectes, ja sigui agafant-los, deixant-los anar o realitzant alguna acció amb ells (disparar una pistola).

Per tal que els raigs de teletransportació no estiguin visibles tot el temps, crearem un nou component en l'XR Origin per activar-los només quan es premi el botó corresponent per dur a terme l'acció (Trigger Button). L'script del qual és el següent:

```
using System.Collections;
using System. Collections.Generic;
using UnityEngine;
using UnityEngine.XR. Interaction. Toolkit;
using UnityEngine. InputSystem;

public class ActivateTeleportationRay : MonoBehaviour
{
    public GameObject leftTeleportation;
    public GameObject rightTeleportation;

    public InputActionProperty leftActivate;
    public InputActionProperty rightActivate;

    void Update()
    {
        leftTeleportation.SetActive(leftActivate.action.ReadValue<float>() > 0.1f);
        rightTeleportation.SetActive(rightActivate.action.ReadValue<float>() > 0.1f);
    }
}
```

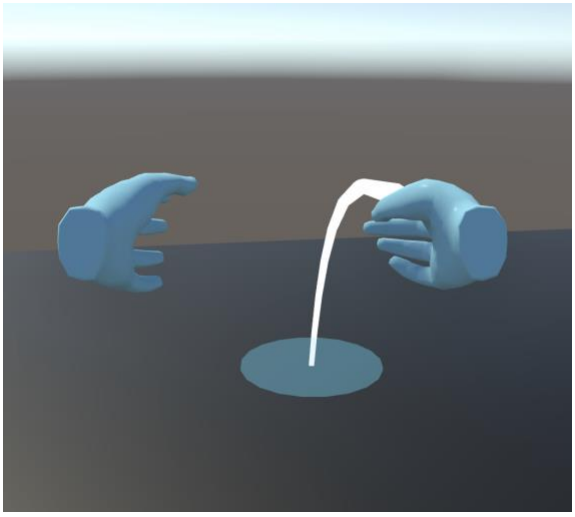
Imatge 38: Script del component ActivateTeleportationRay

En aquest script, definim 2 GameObject per als Teleportation Ray i 2 InputActionProperty per a l'acció de cada mà. En el mètode Update(), activarem o desactivarem els raigs en funció de si estem prement o no el botó de teletransportació (Trigger Button). Com ja hem mencionat anteriorment, en prémer un botó del controlador, hi ha un valor que determina la força amb la qual el prenem. Per tant, establim una condició amb un valor molt baix perquè el raig s'activi ràpidament en prémer el botó. Només ens queda omplir els camps del component de la següent manera:



Imatge 39: Component Activate Teleportation Ray de l'XR Origin

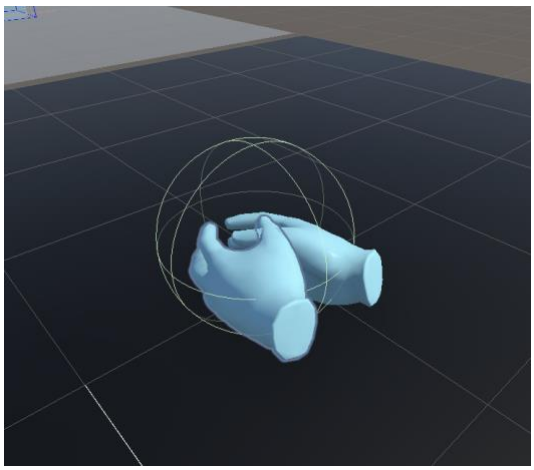
A la següent imatge es pot veure com el raig només apareix a la mà dreta, que és amb la qual s'està prement el botó.



Imatge 40: Execució en la qual es mostra com només surt el raig de teletransportació quan es prem el botó Trigger

Hem de destacar 2 conceptes: Interactor i Interactable. L'Interactor indica el tipus d'interacció que volem i l'Interactable indica com l'objecte respondrà davant de l'interacció. En l'exemple de la teletransportació, el raig seria l'Interactor i l'Interactable seria l'àrea de teletransportació que vam afegir (Teleportation Anchor).

Ara, hem d'afegir el component XR Direct Interactor a ambdues mans (LeftHand Controller i RightHand Controller). Aquest component ens permetrà interactuar amb un objecte que entra en una zona específica molt a prop de les mans. També afegim un Sphere Collider a cadascuna de les mans, que definirà aquesta zona esmentada, amb un radi de 0.1 per a una major precisió. A la següent imatge es pot veure el Sphere Collider:

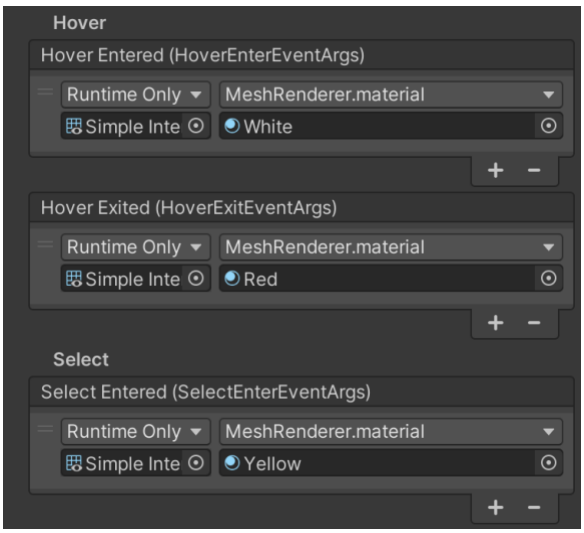


Imatge 41: L'Sphere Collider de les mans per tal d'interactuar amb els elements de l'escena

A continuació, creem una taula a l'escena (Cub) i li afegim un nou material de color blanc. També creem un altre cub més petit (Simple Interactable), amb un nou material de color vermell, amb el qual interactuar, i el col·loquem sobre la taula.

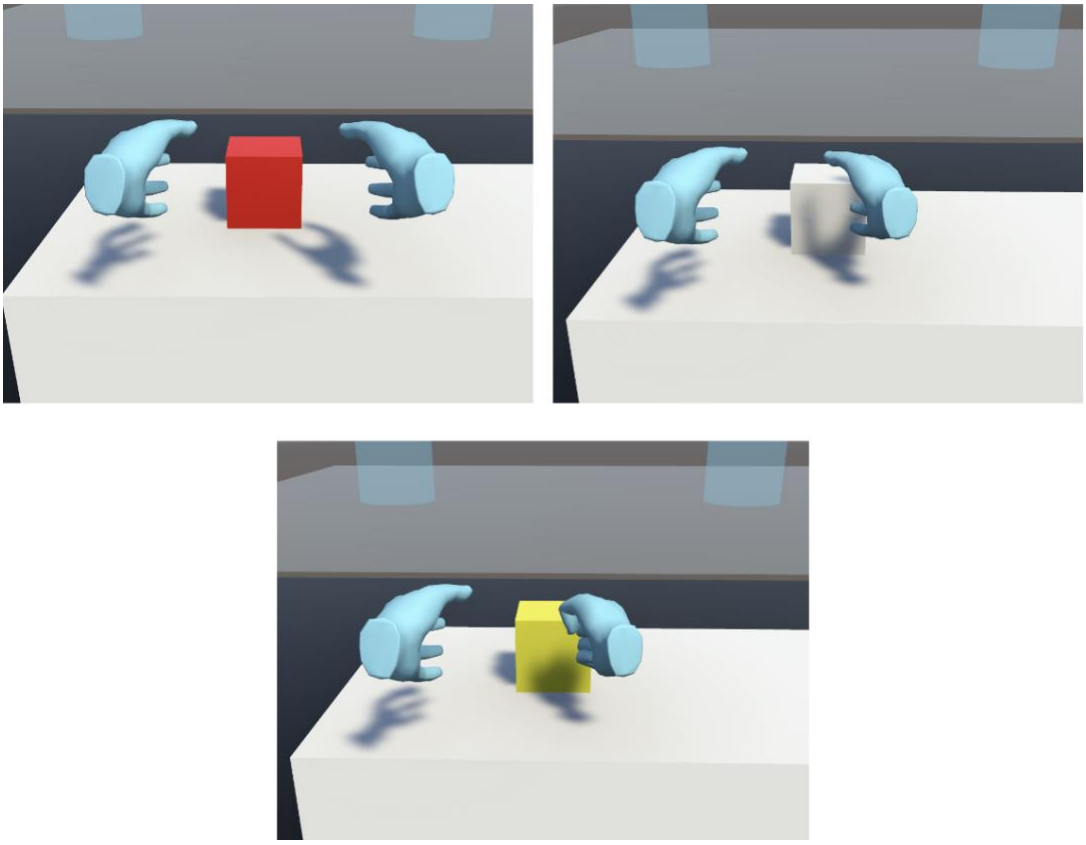
Al Simple Interactable, li afegim un component RigidBody perquè es pugui detectar quan entri en contacte amb el Direct Interactor de les mans. També li afegim el component XR

Simple Interactable. En aquest component, tenim una llista d'esdeveniments per interactuar, com ara el Hover Entered (i el Hover Exited), en el qual afegirem el Simple Interactable i el material (White) que s'assignarà seleccionant l'opció de MeshRenderer.material. Fem el mateix per al Hover Exited amb el material Red i per al Select Entered amb un nou material Yellow.



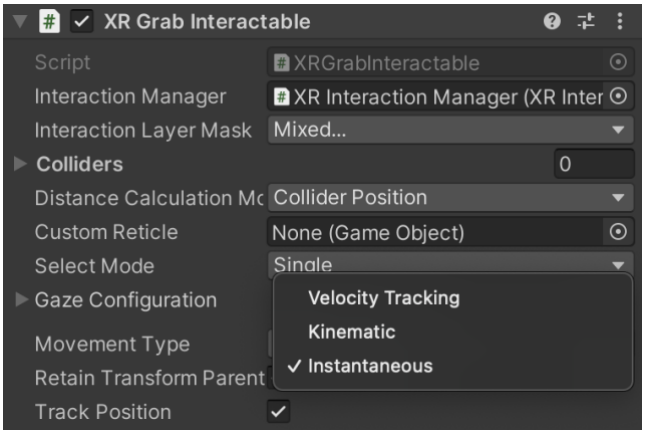
Imatge 42: Esdeveniments configurats del component XR Simple Interactable de l'objecte Simple Interactable

Quan s'executi, el que faran aquestes configuracions és que, en col·lidir qualsevol mà amb el cub, canviarà el material de Red a White, i si es selecciona amb el botó Grip, canviarà a Yellow. En treure la mà del cub, el material tornarà a canviar a Red.



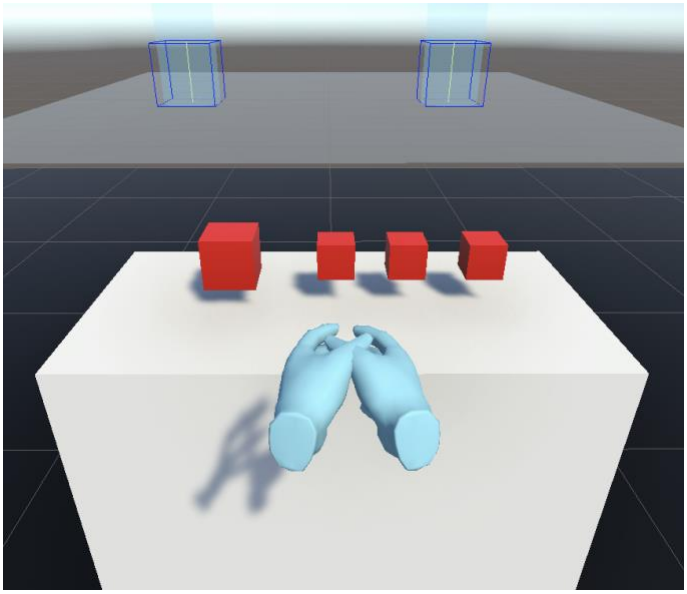
Imatge 43: Execució que mostra les interaccions amb el Simple Interactable

Ara crearem 3 cubs a partir del Simple Interactable. Els anomenarem Grab Interactable Instantaneous, Grab Interactable Kinematic i Grab Interactable Velocity. En lloc de tenir el component XR Simple Interactable, tindran el XR Grab Interactable, de manera que podrem agafar-los acostant la mà i prement el botó corresponent. La diferència entre els 3 radica en la forma en què col·lideixen amb els altres elements de l'escena, i això es configura amb l'atribut Movement Type d'aquest component. Si seleccionem Instantaneous, farà un moviment de manera que podrà travessar els altres elements de l'escena (la taula) o la col·lisió amb ells (els altres cubs, per exemple) serà estranya i no molt precisa (els empenyerà, però semblarà com que els travessa una mica). El Kinematic, en canvi, solucionarà la col·lisió estranya amb els cubs que fa l'Instantaneous, però encara podrà travessar la taula (ja que aquesta no té un component RigidBody). Finalment, el Velocity Tracking col·lidirà de manera correcta tant amb la taula com amb els altres cubs, és a dir, amb tots els altres objectes de l'escena. Aquesta última opció seria la més adequada, tot i que depenent de l'objectiu que busquem, les altres poden ser més útils.



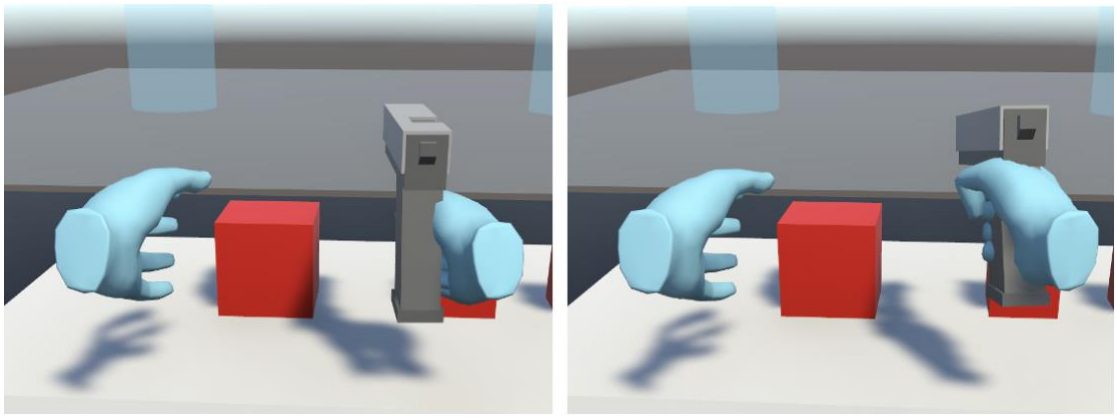
Imatge 44: Opcions de l'atribut Track Position del component XR Grab Interactable

A la següent imatge es pot visualitzar l'escena que hem creat fins ara:



Imatge 45: Escena que s'ha creat fins el moment

Ara, el que farem és aprendre com modificar la posició de l'objecte quan el subjectem (Attach Position). Afegirem un model d'una pistola que hem descarregat del link pistola i la col·locarem a l'escena, damunt de la taula. Li afegirem un Box Collider i l'ajustarem al model de la pistola. També li afegirem un RigidBody i el component XR Grab Interactable. Veurem que, en agafar la pistola, no es subjecta en una posició "real", sense adaptar-se a la mà. Per solucionar-ho, crearem un objecte buit a la pistola anomenat Attach Point i el farem arrossegar a l'atribut Attach Transform del component XR Grab Interactable de la pistola. Mentre executem el joc, haurem de subjectar la pistola i, sense aturar l'execució, moure l'Attach Point a la posició desitjada. Llavors, només haurem de copiar la seva posició i enganxar-la al mateix Attach Point un cop aturem l'execució. Un cop tornem a executar, la pistola es subjectarà a la posició desitjada. A les següents imatges podem veure la diferència en la subjecció:



Imatge 46: Diferència en la subjecció de la pistola una vegada s'ha configurat

A la mà dreta es subjecta en la posició adequada, però fa que a la mà esquerra no es subjecti en una posició adequada, ja que l'hem configurat per a una mà. Això ho solucionarem més endavant.

Per utilitzar un objecte, en aquest cas volem disparar la pistola, el que hem de fer és activar l'esdeveniment Activate (ActivateEventArgs) del component XR Grab Interactable. Aquesta vegada ho farem mitjançant un script. Afegirem un nou component a la pistola, anomenat FireBulletOnActivate. L'script del qual és el següent:

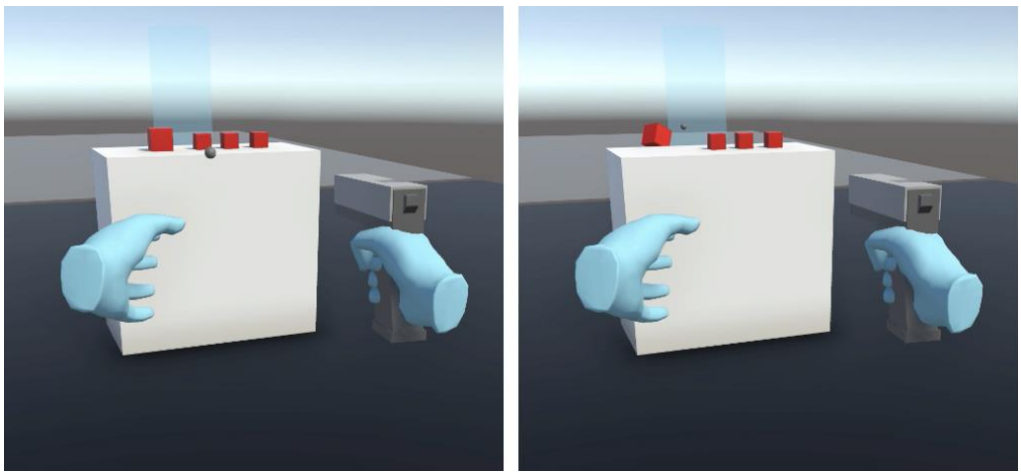
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit; // para acceder al componente XR Grab Interactable

public class FireBulletOnActivate : MonoBehaviour
{

    public GameObject bullet; // the bullet that will be spawned
    public Transform spawnPoint; // the transform position it will spawn
    public float fireSpeed = 20; // the fire speed for the pistol

    // Start is called before the first frame update
    void Start()
    {
        XRGrabInteractable grabbable = GetComponent<XRGrabInteractable>(); // accedemos al componente XR Grab Interactable
        grabbable.activated.AddListener(FireBullet); // añadimos el evento y le indicamos la funcion que se llamara cuando se produzca
    }

    // Update is called once per frame
    void Update()
    {

    }
    // la funcion necesita un parametro de tipo ActivateEventArgs, que nos puede ser util en ocasiones para
    // obtener informacion sobre el interactor y el interactable
    public void FireBullet(ActivateEventArgs arg)
    {   // creamos la bala
        GameObject spawnedBullet = Instantiate(bullet);
        // le asignamos la posicion donde aparecera la bala
        spawnedBullet.transform.position = spawnPoint.position;
        // le asignamos la velocidad al rigid body en la direccion hacia adelante (forward axis)
        spawnedBullet.GetComponent<Rigidbody>().velocity = spawnPoint.forward * fireSpeed;
        // hacemos desaparecer las balas a los 5 segundos
        Destroy(spawnedBullet, 5);
    }
}
```

Imatge 47: Script del component FireBulletOnActivate de la pistola

Definim 3 variables: la bala que es generarà (bullet), la posició on es generarà la bala (spawnPoint) i la velocitat de la bala disparada (fireSpeed). En la funció Start(), accedim al component XR Grab Interactable i indiquem la funció FireBullet(ActivateEventArgs args), que es cridarà quan es produeixi l'esdeveniment Activate, és a dir, quan estiguem agafant la pistola i premem el botó de disparar (Trigger). A la funció FireBullet, necessitem el paràmetre de tipus ActivateEventArgs, que ens pot ser útil en ocasions per obtenir informació sobre l'interactor i l'interactable. Creem la bala, instanciant-la com a spawnedBullet, i li assignem la posició spawnPoint que hem definit perquè aparegui. Després, li assignem la velocitat a la qual es mou la bala en la direcció cap endavant i fem desaparèixer la bala després de 5 segons.

A continuació, creem un nou objecte 3D per a les bales, que en aquest cas serà una esfera molt petita, li afegim un component Rigidbody desmarcant l'ús de la gravetat i indicant la Continous Dynamic Collision Detection, que és útil en objectes que es mouen ràpid perquè Unity faci els càlculs de col·lisió més precisos. El guardarem a la carpeta d'Assets i l'eliminarem de l'escena. Seguidament, l'afegirem al lloc corresponent del component que hem creat (FireBulletOnActivate). Per a la Spawn Position, crearem un objecte buit a la pistola i el col·locarem a la posició on volem que es generin les bales. També l'afegirem

al lloc corresponent del component. Ara, hauríem de poder disparar la pistola. A les següents imatges es pot veure el tret d'una bala impactant amb el Simple Interactable:



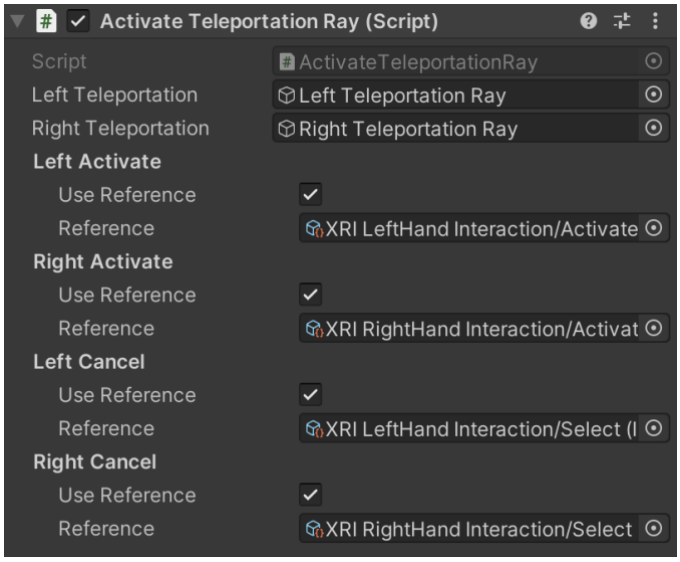
Imatge 48: Execució en la qual es mostra com es pot disparar la pistola i fer que la bala impacti amb els objectes de l'escena

Si provem a executar-ho, veurem que, en agafar un objecte com la pistola, també ens surt el raig de teletransportació (si mantenim el botó Trigger). Per solucionar-ho, hem de modificar l'script ActivateTeleportationRay que vam crear abans, de la següent manera:

```
1  using System.Collections;
2  using System. Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.XR. Interaction. Toolkit;
5  using UnityEngine. InputSystem;
6
7  public class ActivateTeleportationRay : MonoBehaviour
8  {
9      public GameObject leftTeleportation;
10     public GameObject rightTeleportation;
11
12     public InputActionProperty leftActivate;
13     public InputActionProperty rightActivate;
14
15     public InputActionProperty leftCancel;
16     public InputActionProperty rightCancel;
17
18     void Update()
19     {
20         leftTeleportation.SetActive(leftCancel.action.ReadValue<float>() == 0 && leftActivate.action.ReadValue<float>() > 0.1f);
21         rightTeleportation.SetActive(rightCancel.action.ReadValue<float>() == 0 && rightActivate.action.ReadValue<float>() > 0.1f);
22     }
23 }
```

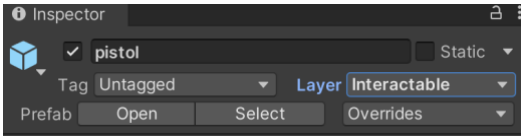
Imatge 49: Script ActivateTeleportationRay modificat del XR Origin

Hem afegit 2 InputActionProperty (leftCancel i rightCancel) per desactivar els raigs de teletransportació quan agafem un objecte. Afegirem la condició per activar-los també quan no estiguem prement el botó Grip (línies 20 i 21).
Només cal emplenar els camps del component de la següent manera:



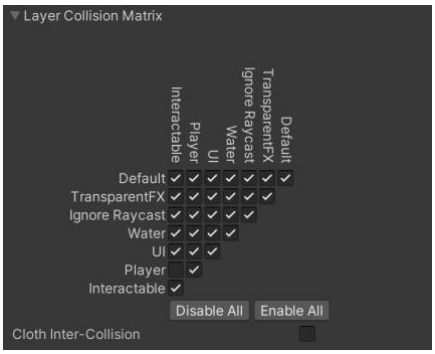
Imatge 50: Component ActivateTeleportationRay actualitzat de l'XR Origin

Per acabar, procedirem a ajustar un possible error que es produeix en agafar un objecte, que aquest col·lideixi amb els controls del personatge (mans). Per fer-ho, hem d'afegir una capa Player i una altra capa Interactable. Assignarem la capa Player a l'XR Origin, però sense assignar-la als seus fills. Assignarem la capa Interactable a tots els objectes interactables (cubs i pistola), aquest cop sí que haurem d'assignar-la també als seus fills. A la següent imatge es mostra on es poden crear i assignar les capes des de l'Inspector (Layer):



Imatge 51: Assignació de la capa Interactable a la pistola des de l'Inspector

Finalment, a "Project Settings > Physics > Layer Collision Matrix", haurem de desmarcar la casella on coincideixen les capes "Player" i "Interactable".



Imatge 52: Layer Collision Matrix de la configuració del projecte

4.2.6 Agafada amb desplaçament i distància

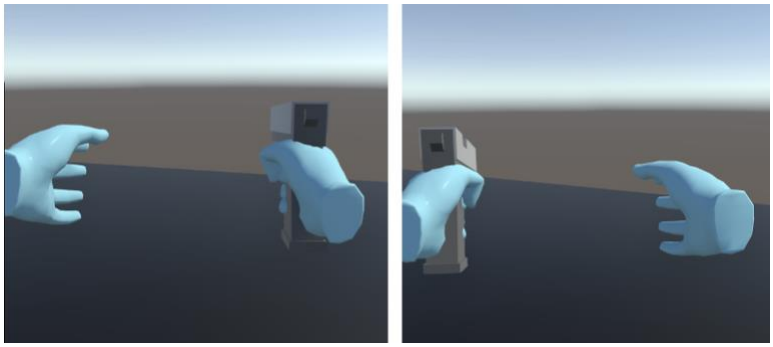
Si ens fixem, en agafar la pistola amb la mà dreta, es col·loca en la posició correcta que havíem configurat anteriorment. En canvi, si l'agafem amb la mà esquerra, aquesta no es col·loca en una posició normal, és a dir, es queda separada de l'agafada de la mà considerablement. Per solucionar això, el que hem de fer és seleccionar la pistola,

eliminar el component XR Grab Interactable i crear-ne un anomenat XR Grab Interactable Two Attach. L'script és el següent:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.XR.Interaction.Toolkit;
5
6   public class XRGrabInteractableTwoAttach : XRGrabInteractable
7   {
8       public Transform leftAttachTransform;
9       public Transform rightAttachTransform;
10
11      protected override void OnSelectEntered(SelectEnterEventArgs args)
12      {
13          if(args.interactorObject.transform.CompareTag("Left Hand"))
14          {
15              attachTransform = leftAttachTransform;
16          }
17          else if(args.interactorObject.transform.CompareTag("Right Hand"))
18          {
19              attachTransform = rightAttachTransform;
20          }
21
22          base.OnSelectEntered(args);
23      }
24  }
```

Imatge 53: Script del component XR Grab Interactable Two Attach

Com es pot veure, la classe hereta de XRGrabInteractable, per la qual cosa tindrà el mateix comportament que el component que hem eliminat, encara que sobreescriurem la funció OnSelectEntered(SelectEnterEventArgs args) per fer l'agafada correctament per a cada mà. Hem declarat 2 Transform, un per a cada posició de les mans. Abans de res, hem de crear 2 etiquetes (Tags), Left Hand i Right Hand, i assignar-les a la seva corresponent mà. Fem això per accedir des de l'script als Hand Controllers a través de l'etiqueta, per si en algun moment modifiquem els seus noms seguiria funcionant. Un cop fet això, podem assignar la posició de l'agafada (attachTransform) en funció de la mà amb la qual agafem l'objecte gràcies al paràmetre args, que ens proporcionarà informació sobre la interacció, del qual mirarem si es tracta del Hand Controller amb l'etiqueta Left Hand o Right Hand (línies 13 i 17). Finalment, el que falta per fer és assignar els camps leftAttachTransform i rightAttachTransform des de l'Inspector. Per això, duplicarem l'Attach Point que teníem creat per a la mà dreta i simplement li invertirem el valor de la component x de la posició. D'aquesta manera, l'objecte que agafem es col·locarà correctament també a la mà esquerra. Assignem els Attach Point de cada mà al component i ja ho tindríem.



Imatge 54: Execució que mostra com l'agafada de la pistola amb les dues mans s'ha corregit

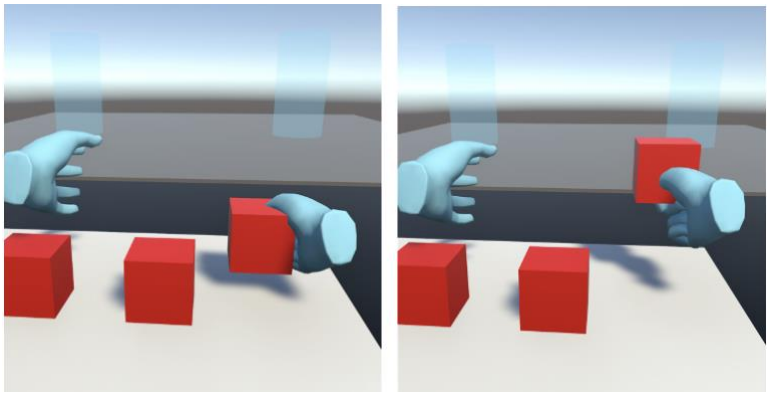
Ara, el que volem fer és mostrar una altra forma d'agafar un objecte de manera que mantingui la posició respecte a la mà. Per fer-ho, tenim 2 opcions. Una és activant l'opció Use Dynamic Attach del component XR Grab Interactable de l'objecte (sempre que tinguem la versió del XR Interaction Toolkit 2.1 o superior). L'altra és mitjançant un script, que explicarem a continuació. Primer, duplicarem un dels cubs i l'anomenarem Grab Interactable Offset. Li eliminarem el component XR Grab Interactable i afegirem un de nou anomenat XR Offset Grab Interactable. L'script és el següent:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.XR.Interaction.Toolkit;
5
6   public class XROffsetGrabInteractable : XRGrabInteractable
7   {
8
9       // Start is called before the first frame update
10      void Start()
11      {
12          if(!attachTransform)
13          {
14              GameObject attachPoint = new GameObject("Offset Grab Pivot");
15              attachPoint.transform.SetParent(transform, false);
16              attachTransform = attachPoint.transform;
17          }
18      }
19
20      protected override void OnSelectEntered(SelectEnterEventArgs args)
21      {
22          attachTransform.position = args.interactorObject.transform.position;
23          attachTransform.rotation = args.interactorObject.transform.rotation;
24
25          base.OnSelectEntered(args);
26      }
27  }
```

Imatge 55: Script del component XROffsetGrabInteractable de l'objecte Grab Interactable Offset

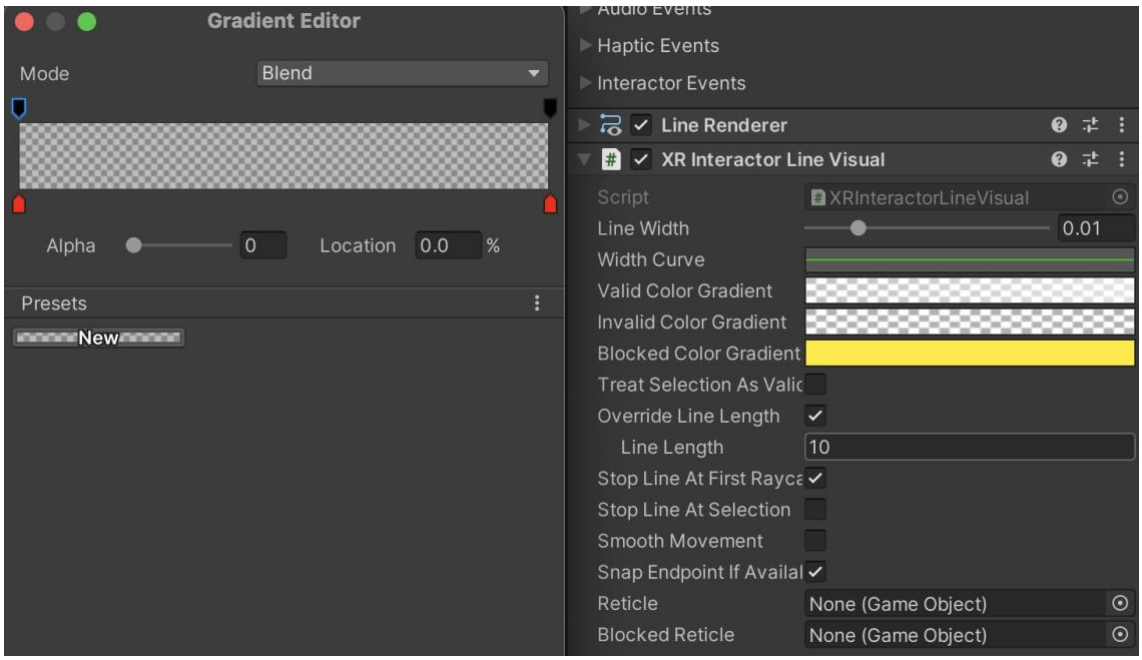
Si ens fixem, també hereta d'XRGrabInteractable com l'explicat anteriorment, per la qual cosa tindrà el mateix comportament que el component XRGrabInteractable, excepte pels mètodes que sobreescrivim. En el mètode Start(), comprovarem si disposa d'un Attach Transform. En cas que no, crearem un anomenat Offset Grab Pivot. A continuació (línia

15), assignem com a pare de l'attachPoint l'objecte que estem agafant, i el paràmetre false indica que tindrà una posició i rotació locals a (0, 0, 0). Ara, sobreescriurem el mètode OnSelectEntered(SelectEnterEventArgs args), com ja havíem fet abans, de manera que assignarem la posició i rotació de l'atribut Attach Transform (attachTransform) perquè siguin les de l'InteractorObject, és a dir, de l'objecte que agafem (línies 22 i 23). A les següents imatges es pot visualitzar el canvi que hem implementat:



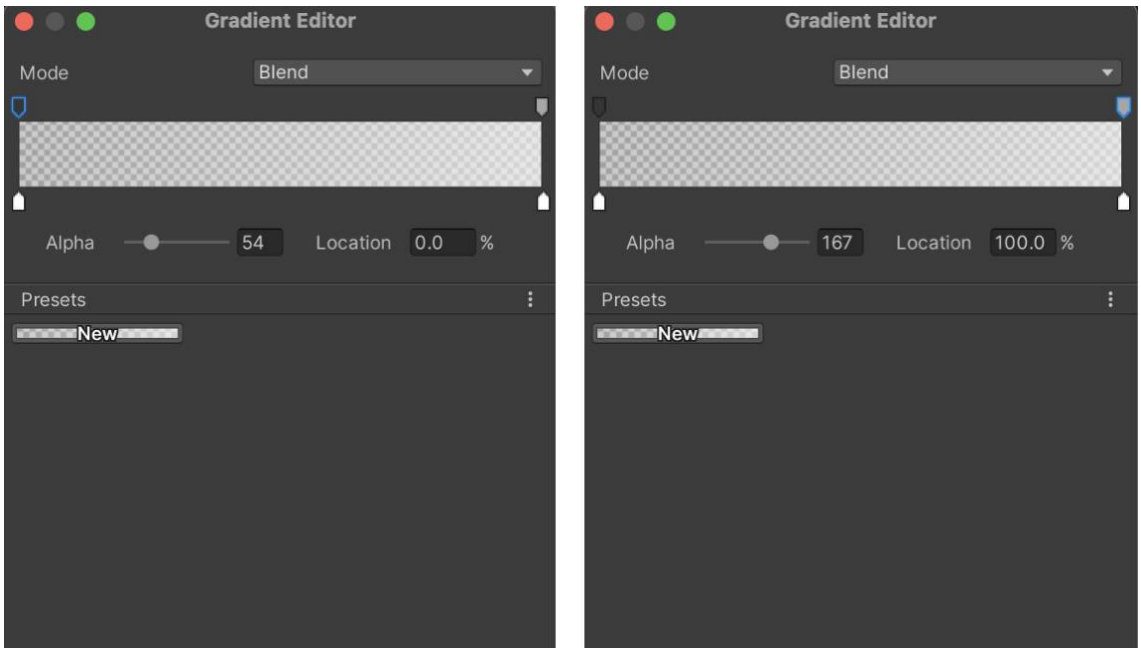
Imatge 56: Execució que mostra el resultat d'haver configurat un objecte perquè quan el jugador l'agafi amb les mans, aquest mantingui la posició respecte la mà

A continuació, el que farem és poder agafar objectes des de la distància mitjançant raigs. Afegirem 2 XR > Ray Interactor, un per a cada mà, en el nostre XR Origin > Camera Offset i assignarem els presets de cada mà com havíem fet anteriorment. Els anomenarem Left Grab Ray i Right Grab Ray. Simplement amb això ja hauríem de poder interactuar amb els objectes des de la distància. Per no visualitzar els raigs quan no apunten a cap objecte amb el qual interactuar (o el que és el mateix, quan no siguin vàlids), modificarem el camp Invalid Color Gradient del component XR Interactor Line Visual posant a 0 els valors Alpha en els dos elements de la part de dalt, com es mostra a la següent imatge:



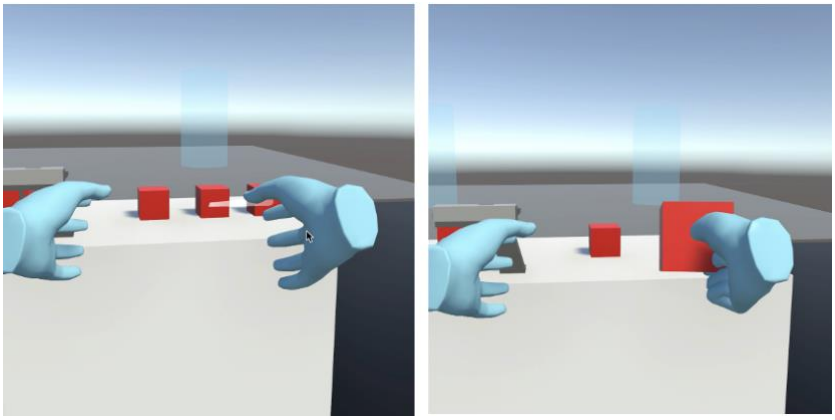
Imatge 57: Configuració del component XR Interactor Line Visual

D'aquesta manera, seran totalment transparents o invisibles quan no siguin vàlids. Hem disminuït l'amplada del raig a 0.01 (camp Line Width). També hem modificat el camp Valid Color Gradient per quan sigui vàlid (apunti a un objecte amb el que pugui interactuar), disminuint els valors Alpha (més al costat esquerre que al dret), com es mostra a les següents imatges:



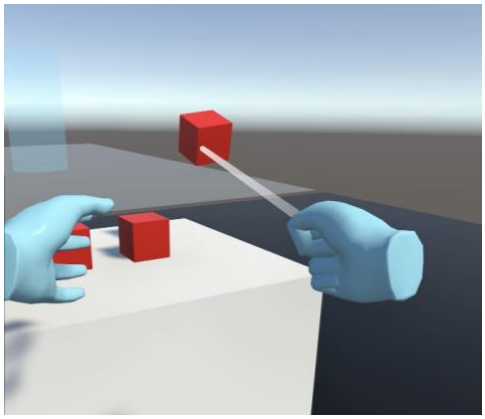
Imatge 58: Configuració del Valid Color Gradient del component XR Interactor Line Visual

Finalment, en l'atribut Max Raycast Distance del component XR Ray Interactor hem posat un valor de 4, per indicar la distància màxima a la qual podrem interactuar amb els objectes. A les següents imatges es pot veure que quan apuntem a un objecte es visualitza el raig i si prenem el Grip Button podem agafar-lo des de la distància:



Imatge 59: Execució en la qual es mostra com podem agafar un objecte desde la distància amb els Grab Ray

Si intentem agafar l'últim cub que hem creat (que hem anomenat Grab Interactable Offset), veurem que manté la distància, tal com es pot veure a la següent imatge:



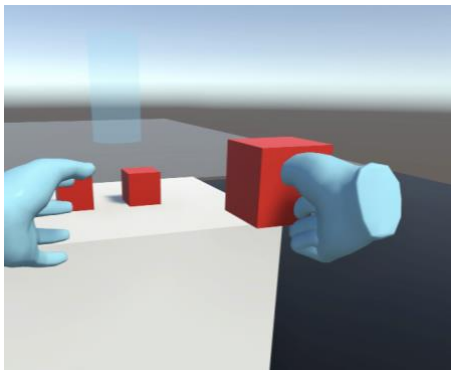
Imatge 60: Execució que mostra com l'objecte Grab Interactable Offset també manté la posició respecte la mà quan s'agafa amb el Grab Ray

Per solucionar això, podem modificar l'script que hem creat anteriorment, XROffsetGrabInteractable, del cub que estem tractant.

```
6   public class XROffsetGrabInteractable : XRGrabInteractable
7   {
8       private Vector3 initialLocalPos;
9       private Quaternion initialLocalRot;
10
11      // Start is called before the first frame update
12      void Start()
13      {
14          if(!attachTransform)
15          {
16              GameObject attachPoint = new GameObject("Offset Grab Pivot");
17              attachPoint.transform.SetParent(transform, false);
18              attachTransform = attachPoint.transform;
19          }
20          else
21          {
22              initialLocalPos = attachTransform.localPosition;
23              initialLocalRot = attachTransform.localRotation;
24          }
25      }
26
27      protected override void OnSelectEntered(SelectEnterEventArgs args)
28      {
29          if(args.interactorObject is XRDirectInteractor)
30          {
31              attachTransform.position = args.interactorObject.transform.position;
32              attachTransform.rotation = args.interactorObject.transform.rotation;
33          }
34          else
35          {
36              attachTransform.localPosition = initialLocalPos;
37              attachTransform.localRotation = initialLocalRot;
38          }
39
40          base.OnSelectEntered(args);
41      }
42  }
```

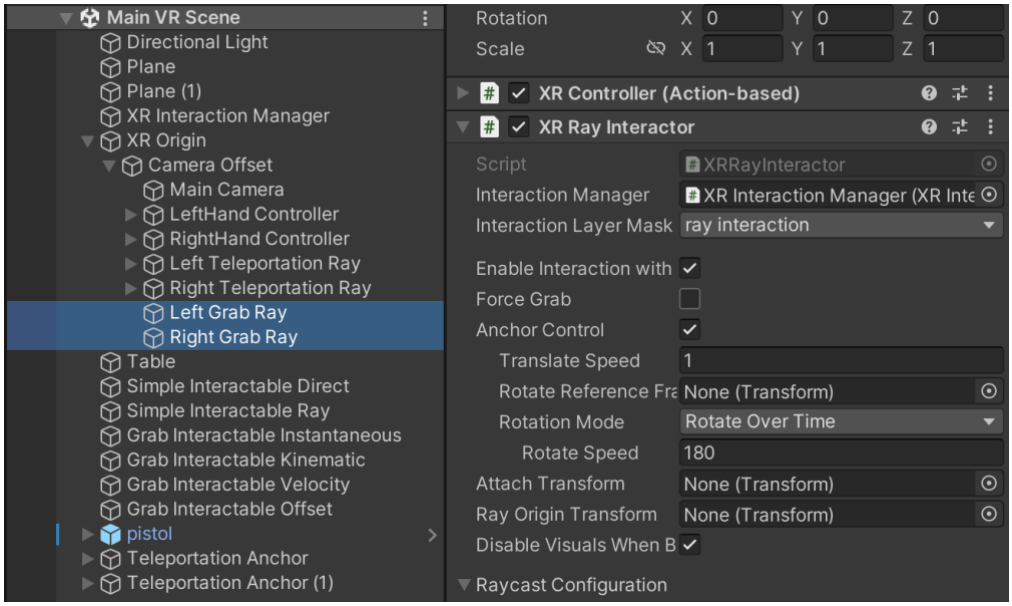
Imatge 61: Script del component XROffsetGrabInteractable modificat

En el mètode OnSelectEntered(), hem afegit una comprovació per determinar si l'interactor és un XR Direct Interactor o no. Si és així, voldrà dir que estem intentant agafar-lo de prop, per la qual cosa hem d'actualitzar la posició i la rotació de l'Attach Transform tal com fèiem abans. D'aquesta manera, si intentem agafar-lo de prop, mantindrà la posició. Si no és el cas, voldrà dir que estem intentant agafar-lo amb el raig, és a dir, des de la distància. Aleshores, hem creat dues variables, initialLocalPos i initialLocalRot, per emmagatzemar-hi una posició i una rotació per defecte. Després, anem a la funció Start(), on afegirem la condició contrària a la que ja tenim (el cas en què no hi ha un Attach Transform assignat). Si ja tenim un Attach Transform, assignarem els valors d'aquest a les variables que hem creat. Tornem a la funció OnSelectEntered(), on si intentem agafar l'objecte amb el raig, assignarem els valors de posició i rotació per defecte. Ara, en intentar agafar aquest últim cub, es col·locarà al costat de la mà.



Imatge 62: Execució en la qual es mostra com l'objecte Grab Interactable Offset es col·loca al costat de la mà al agafar-ho amb el Grab Ray, una vegada modificat l'script XROffsetGrabInteractable

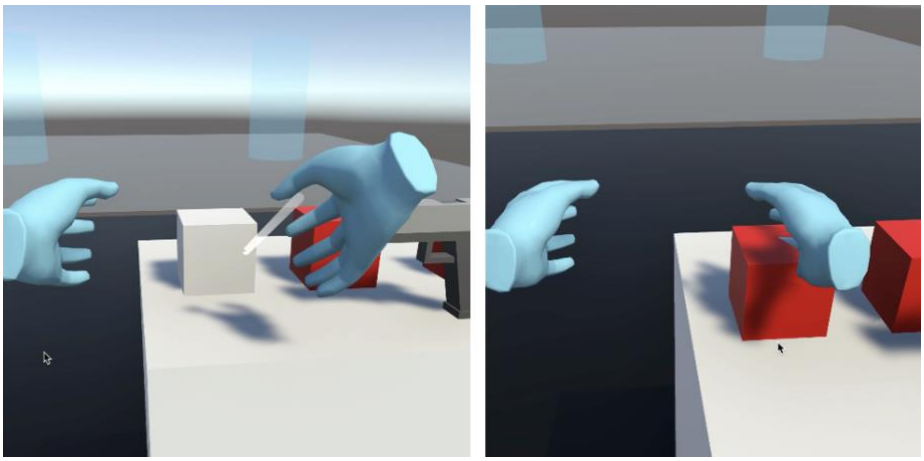
Una altra opció que tenim per solucionar això, tot i que amb tots els objectes, és marcar la casella Force Grab dels 2 Grab Ray:



Imatge 63: Casella Force Grab del component XR Ray Interactor dels Grab Ray

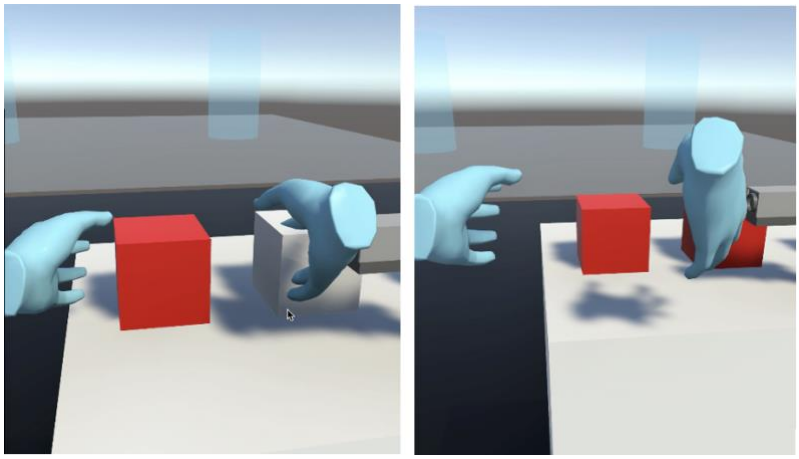
Després de tot el que hem fet, pot ser que les interaccions siguin una mica confuses i desorganitzades, ja que, per exemple, podem teletransportar-nos amb el mateix botó amb el qual podem interactuar amb un objecte des de la distància. Ara, el que farem és

mostrar com fer que un objecte no respongui a un Interactor. Per exemple, imaginem que volem que un objecte només es pugui agafar quan la mà estigui a prop d'ell (direct interactor), però no quan estigui lluny (ray interactor). Separarem la teletransportació, la interacció directa i la interacció per raig. Per fer-ho, seleccionarem qualsevol Interactor (per exemple, un Hand Controller) i, en el camp Interaction Layer Mask, afegirem les 3 capes esmentades: teleportation, direct interaction i ray interaction. Seleccionarem “teleportation" per als Teleportation Ray, "direct interaction" pels Hand Controller i "ray interaction" per als Grab Ray. D'aquesta manera, només podrà interactuar amb els objectes que tinguin com a mínim la mateixa Interaction Layer Mask (poden tenir-ne diverses seleccionades). Ara, provarem això de la següent manera. Seleccionarem la Layer Mask del Pla i dels Teleportation Anchor com a "teleportation". Després, dels cubs petits i de la pistola com a "direct interaction" i "ray interaction" (perquè es puguin agafar tant de prop com de lluny). Finalment, duplicarem el cub gran (Simple Interactable), anomenarem un Simple Interactable Direct i l'altre Simple Interactable Ray. Al primer li assignarem la capa "direct interactor" i al segon la capa "ray interactor". Si ara ho provem, veurem que amb el primer només podrem interactuar quan la mà estigui a prop d'ell, i amb el segon només quan l'apuntem (en aquest apareixerà el raig, en el primer no).



Imatge 64: Execució en la que es mostra com es pot interactuar amb el Simple Interactable Ray només des de la distància, amb el Grab Ray

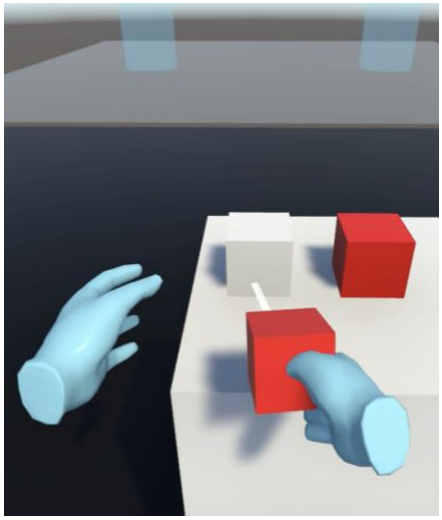
En aquestes imatges, podem apreciar com es pot interactuar amb el Simple Interactable Ray mitjançant una interacció per raig, però no mitjançant una interacció directa.



Imatge 65: Execució en la que es mostra com es pot interactuar amb el Simple Interactable Direct només quan apropem la mà

En canvi, en aquestes imatges podem veure com podem interactuar amb el Simple Interactable Direct mitjançant una interacció directa, però no mitjançant una interacció per raig.

Només ens queda corregir una cosa. Si ens fixem en la següent imatge, veurem que, en agafar un objecte, si apuntem a un altre objecte amb el qual es pugui interactuar mitjançant el raig, aquest apareixerà.



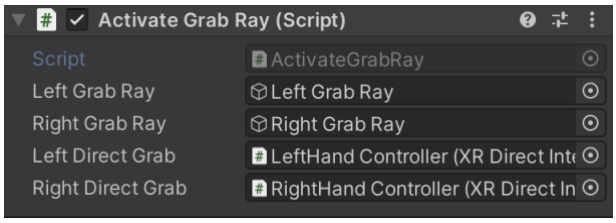
Imatge 66: Execució en la qual es mostra un bug a l'hora d'agafar un objecte i apuntar a un altre

Per solucionar això, hem de crear un nou component a l'XR Origin anomenat ActivateGrabRay, l'script del qual és el següent:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.XR.Interaction.Toolkit;
5
6   public class ActivateGrabRay : MonoBehaviour
7   {
8       public GameObject leftGrabRay;
9       public GameObject rightGrabRay;
10
11      public XRDirectInteractor leftDirectGrab;
12      public XRDirectInteractor rightDirectGrab;
13      // Start is called before the first frame update
14      void Start()
15      {
16
17      }
18
19      // Update is called once per frame
20      void Update()
21      {
22          leftGrabRay.SetActive(leftDirectGrab.interactablesSelected.Count == 0);
23          rightGrabRay.SetActive(rightDirectGrab.interactablesSelected.Count == 0);
24      }
25  }
```

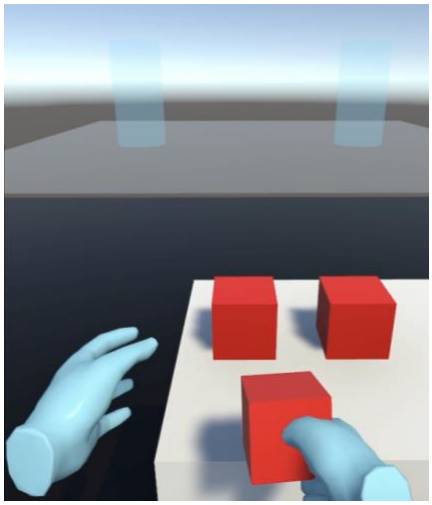
Imatge 67: Script del component ActivateGrabRay de l'XR Origin

Declarem 2 GameObject per als Grab Ray (interacció per raig) i 2 XRDirectInteractor per als Hand Controllers (interacció directa). L'únic que hem de fer és activar els Grab Ray en cas que no estiguem interactuant amb cap objecte (interactable), tal com es pot veure en les línies 22 i 23. Només faltarà omplir els camps del component des de l'inspector:



Imatge 68: Component Activate Grab Ray d'XR Origin

Un cop fet això, si ho provem, veurem que, en agafar un objecte, no apareixerà el raig encara que apuntem a un altre objecte. La següent imatge mostra el mateix escenari que abans, amb la diferència que ja no apareix el raig en apuntar a l'altre cub:

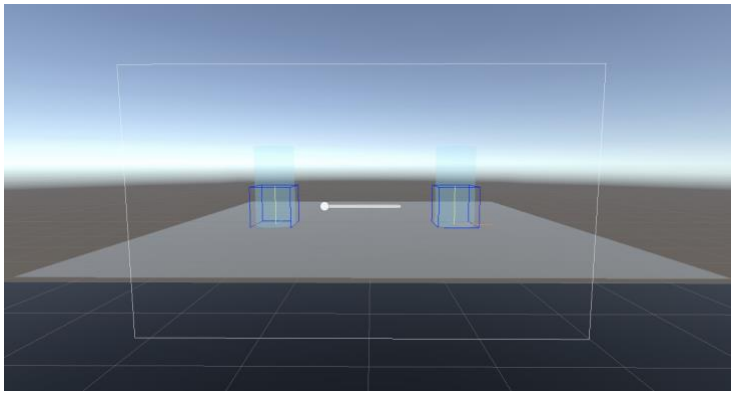


Imatge 69: Execució que mostra com ja no apareix el bug que feia que aparegués el raig a l'hora d'agafar un objecte i apuntar a un altre

4.2.7 Interfície d'usuari

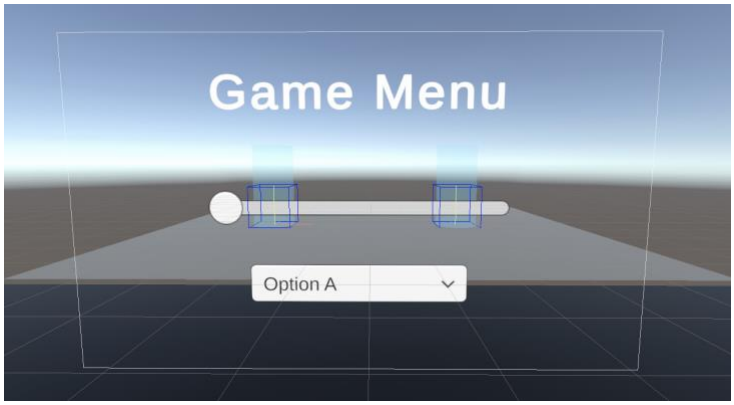
En aquest apartat, el que farem serà configurar un menú pel joc (UI), al qual puguem accedir amb el botó corresponent del comandament esquerre i interactuar amb els Grab Ray, podent canviar alguna configuració.

Per començar, hem d'afegir a l'escena un objecte UI > Canvas, de manera que s'afegirà automàticament un altre objecte anomenat EventSystem. Aquest últim s'encarrega de dirigir la interacció amb la interfície d'usuari (UI). Seleccionarem el Canvas, que el podrem visualitzar com un panell transparent, i afegirem un Slider (UI > Slider), el qual apareixerà al mig. Si provem a executar, veurem el Slider sempre el mig de la pantalla i podrem interactuar amb ell amb el ratolí. Llavors, perquè formi part de l'escena com un objecte 3D, haurem de canviar el Render Mode del Canvas, de Screen Space – Overlay a World Space. Ara, podrem canviar la seva escala i rotació, ja que és massa gran per la nostra escena. El col·loquem i ajustem al nostre gust, en el nostre cas l'hem posat darrere de la taula, enfront de les mans, com es pot veure a la següent imatge:



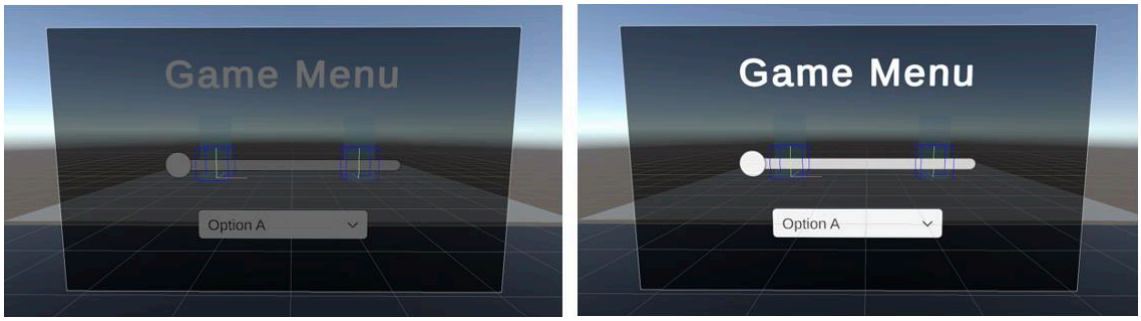
Imatge 70: Canvas afegit a l'escena amb un slider

El que farem ara és afegir-li més elements, com poden ser un títol i un Dropdown. Per tal d'afegir text (UI > Text – TextMeshPro), haurem d'importar els recursos que demanen a la pestanya que ens apareixerà. Canviarem el text per Game Menu i el posicionarem a la part superior central del Canvas. El Dropdown, en canvi, el posicionarem a sota del Slider, tal i com es veu a la següent imatge:



Imatge 71: Canvas actualitzat amb títol i un dropdown

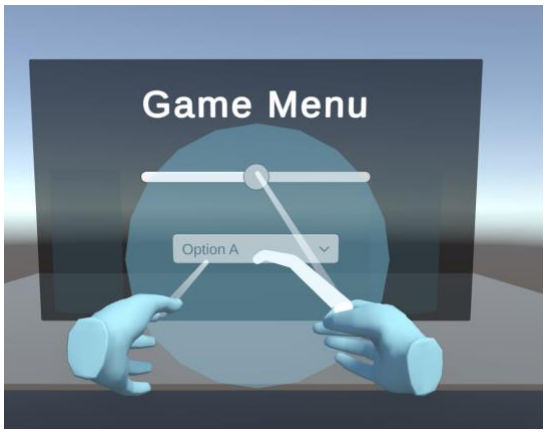
A continuació, afegirem un element Panel al Canvas, al qual canviarem el color i el farem semi-transparent. Perquè no es trobi a sobre dels altres elements, hem de posicionar el Panel com el primer element de la jerarquia d'elements del Canvas, de manera que el Títol, Slider i Dropdown es visualitzin millor. Aquest canvi el podem apreciar a les següents imatges:



Imatge 72: Canvas actualitzat amb un element Panel. Es mostra la diferència de posicionar-lo abans o després de la resta d'elements

Si ens fixem a l'executar, veurem que el podem veure a l'escena, però no podem interactuar amb els raigs encara. Per tal de fer-ho, hem de seleccionar el Canvas, eliminar

el component Graphic Raycaster i afegir el component Tracked Device Graphic Raycaster. Ara, hem d'actualitzar l'Event System, eliminant qualsevol component Input Module que hi hagi per defecte, i afegint el component XR UI Input Module. D'aquesta manera, ja podem interactuar amb el Slider i el Dropdown del Canvas, gràcies als Grab Ray. No obstant això, en fer-ho, també apareixeran els Teleport Ray, que ens faran teletransportar-nos quan deixem anar el botó Trigger. A la següent imatge es pot veure com apareix tant el Right Grab Ray com el Right Teleportation Ray en interactuar amb el Slider:



Imatge 73: Execució que mostra un bug que surt el raig de teletransportació en interactuar amb el menú

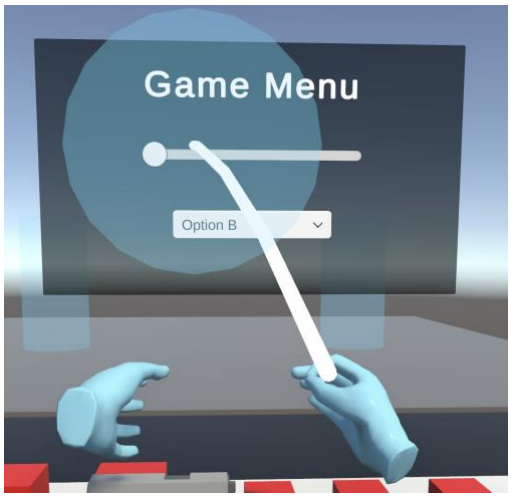
Per arreglar aquest bug, hem de modificar un altre cop el script ActivateTeleportationRay del XR Origin.

```
using System.Collections;
using System. Collections.Generic;
using UnityEngine;
using UnityEngine.XR. Interaction. Toolkit;
using UnityEngine. InputSystem;

public class ActivateTeleportationRay : MonoBehaviour
{
    public GameObject leftTeleportation;
    public GameObject rightTeleportation;

    public InputActionProperty leftActivate;
    public InputActionProperty rightActivate;

    public InputActionProperty leftCancel;
    public InputActionProperty rightCancel;

    public XRRayInteractor leftRay;
    public XRRayInteractor rightRay;

    void Update()
    {
        bool isLeftRayHovering = leftRay.TryGetHitInfo(out Vector3 leftPos, out Vector3 leftNormal,
        out int leftNumber, out bool leftValid);

        leftTeleportation.SetActive(!isLeftRayHovering && leftCancel.action.ReadValue<float>() == 0 &&
        leftActivate.action.ReadValue<float>() > 0.1f);

        bool isRightRayHovering = leftRay.TryGetHitInfo(out Vector3 rightPos, out Vector3 rightNormal,
        out int rightNumber, out bool rightValid);

        rightTeleportation.SetActive(!isRightRayHovering && rightCancel.action.ReadValue<float>() == 0 &&
        rightActivate.action.ReadValue<float>() > 0.1f);
    }
}
```

Imatge 74: Script ActivateTeleportationRay modificat de l'XR Origin

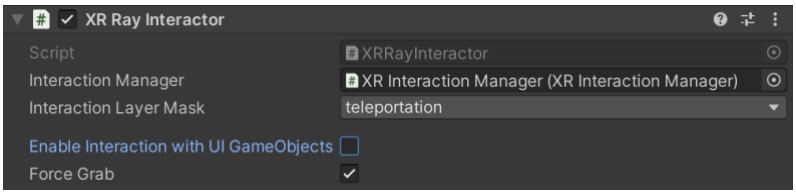
Hem afegit 2 camps pels raigs (línies 18 i 19). A la funció Update(), hem definit 2 booleans, un per cada mà, per tal de saber si els raigs estan col·lidint amb alguna cosa. Llavors, hem afegit la condició de si no estan col·lidint amb cap element de l'escena, a més de les condicions que ja hi havia (no estar pressionant el botó Grip i estar pressionant el botó Trigger).

Ara, se'ns presenta un altre bug a arreglar: si pressionem el botó Trigger per tal de teletransportar-nos i apuntem al menú, veurem que podrem interactuar amb ell, tal i com es veu a la següent imatge:



Imatge 75: Execució en la qual es mostra un bug quan activem el raig de teletransportació i apuntem al menú, el qual segueix apareixent

Solucionar això és tan senzill com desactivar l'opció Enable Interaction with UI GameObjects del component XR Ray Interactor dels Teleportation Ray:



Imatge 76: Configuració del component XR Ray Interactor dels Teleportation Ray

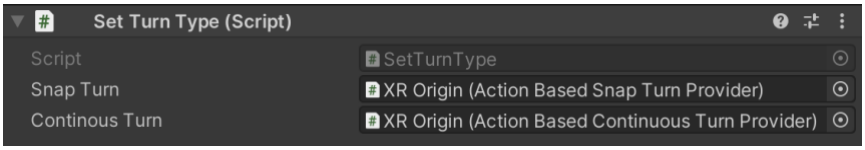
A continuació, utilitzarem el Dropdown per tal de poder escollir entre el Continuous Turn i el Snap Turn. Afegirem un nou component al Canvas anomenat SetTurnType, l'script del qual serà el següent:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class SetTurnType : MonoBehaviour
{
    public ActionBasedSnapTurnProvider snapTurn;
    public ActionBasedContinuousTurnProvider continuousTurn;

    public void SetTypeFromIndex(int index)
    {
        if(index == 0)
        {
            snapTurn.enabled = false;
            continuousTurn.enabled = true;
        }
        else if(index == 1)
        {
            snapTurn.enabled = true;
            continuousTurn.enabled = false;
        }
    }
}
```

Imatge 77: Script del component SetTurnType del Canvas

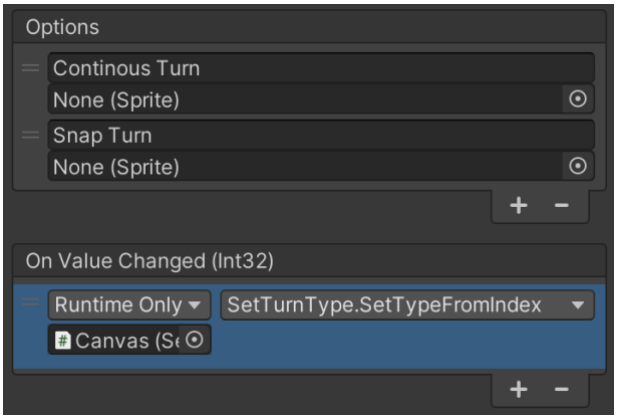
Definim 2 variables anomenades snapTurn i continuousTurn, que els seus tipus són els dels components ContinuousTurnProvider i SnapTurnProvider que vam afegir al XR Origin, per poder rotar el nostre personatge. Després, necessitem una funció, SetTypeFromIndex(int index), per seleccionar el tipus de rotació a partir d'un nombre passat per paràmetre, que representarà l'element escollit del Dropdown. Un 0 voldrà dir que hem d'activar el Snap Turn i desactivar el Continuous Turn, i un 1 voldrà dir el contrari.

Només ens faltarà, doncs, omplir els camps desde l'Inspector, que és bàsicament arrossegar el XR Origin en ambdós, i ja detecta cada component automàticament, tal i com es pot veure a la següent imatge:



Imatge 78: Component SetTurnType del Canvas

I, finalment, haurem d'omplir les opcions del Dropdown del Canvas i l'event per quan canviï l'opció escollida, arrossegant el Canvas i seleccionant la funció corresponent del script que volem que es cridi (SetTypeFromIndex), com es mostra a la següent imatge:



Imatge 79: Configuració de les opcions i l'esdeveniment que es produeix en canviar d'una a altra del Dropdown

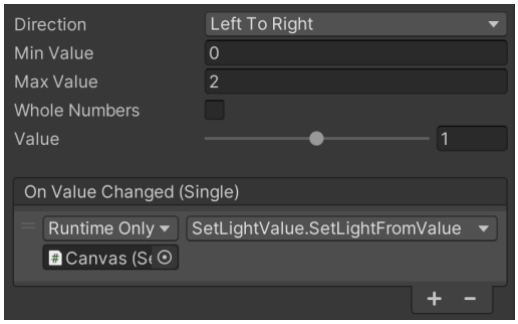
Ara li donarem una funcionalitat al Slider que hem afegit anteriorment. Controlarem la il·luminació, de manera que puguem disminuir i augmentar la intensitat de la llum de l'escena quan vulguem. De manera molt similar al que acabem de fer pel Dropdown, crearem un nou component al Canvas anomenat SetLightValue. El script és el següent:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class SetLightValue : MonoBehaviour
6   {
7       public Light directionalLight;
8
9       public void SetLightFromValue(float lightValue)
10      {
11          directionalLight.intensity = lightValue;
12      }
13  }
```

Imatge 80: Script del component SetLightValue del Canvas

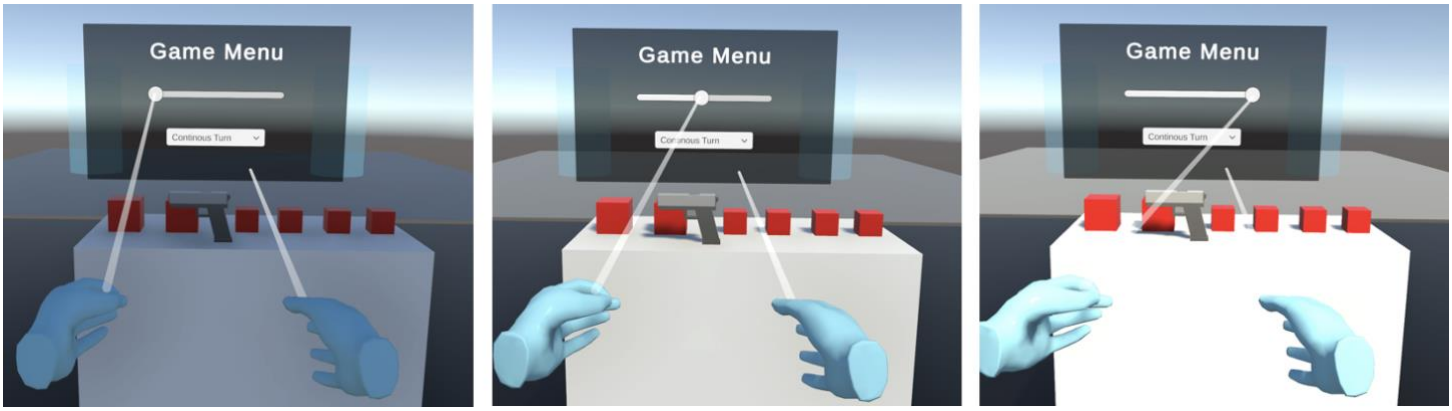
És molt simple. Declarem una variable de tipus Light per emmagatzemar la llum de l'escena. Després, hem declarat la funció SetLightFromValue amb un paràmetre numèric per la intensitat de la llum, en la qual li assignem aquest valor al seu atribut intensity.

A l'Inspector arrossegarem la llum de l'escena al camp corresponent del component i, seguidament, farem la configuració del Slider. Si ens fixem, podem canviar el valor mínim i màxim que el Slider pot tenir. Per defecte, la llum té una intensitat de valor 1. Llavors, posarem el mínim de 0, el màxim de 2 i el valor inicial de 1. Només ens queda configurar l'event On Value Changed del Slider, en el qual seleccionarem el Canvas i la funció SetLightFromValue, que es cridarà cada cop que canviï el seu valor. A la següent imatge es pot veure la configuració del Slider:



Imatge 81: Configuració del Slider

Si ara provem a executar, veurem que podem variar la intensitat de la llum de l'escena, tal i com es pot visualitzar a les següents imatges:



Imatge 82: Execució que mostra com es pot modificar la il·luminació de l'escena des del menú

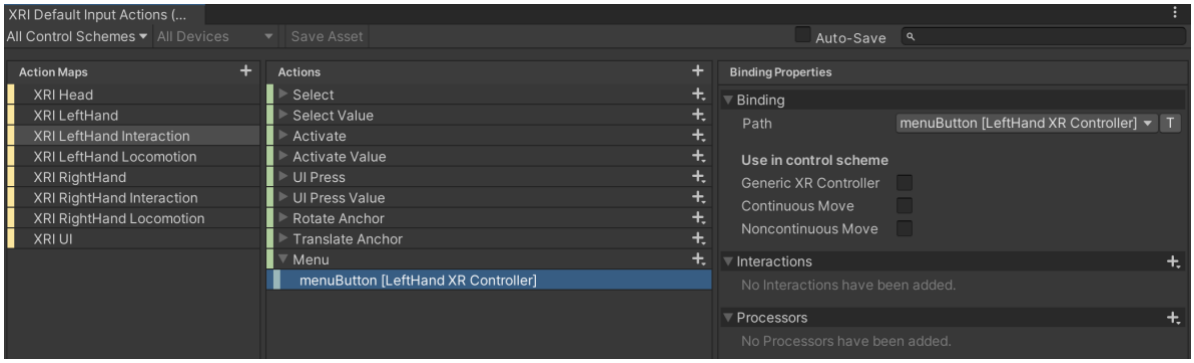
Una vegada ja tenim configurat el menú, volem poder-lo fer aparèixer i desaparèixer davant del jugador polsant un botó en un dels comandaments. Per fer-ho, crearem un objecte buit anomenat Game Menu i posarem el Canvas com el seu fill. Llavors, li crearem (al Game Menu) un nou component anomenat GameMenuManager per poder-ho portar a terme. El script és el següent:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.InputSystem;
5
6   public class GameMenuManager : MonoBehaviour
7   {
8       public Transform head;
9       public float spawnDistance = 2;
10      public GameObject menu;
11      public InputActionProperty showButton;
12      // Start is called before the first frame update
13      void Start()
14      {
15
16      }
17
18      // Update is called once per frame
19      void Update()
20      {
21          menu.transform.LookAt(new Vector3(head.position.x, menu.transform.position.y, head.position.z));
22          menu.transform.forward *= -1;
23
24          if(showButton.action.WasPressedThisFrame())
25          {
26              menu.SetActive(!menu.activeSelf);
27
28              menu.transform.position = head.position +
29              new Vector3(head.forward.x, 0, head.forward.z).normalized * spawnDistance;
30          }
31      }
32  }
```

Imatge 83: Script del component GameMenuManager del Game Menu

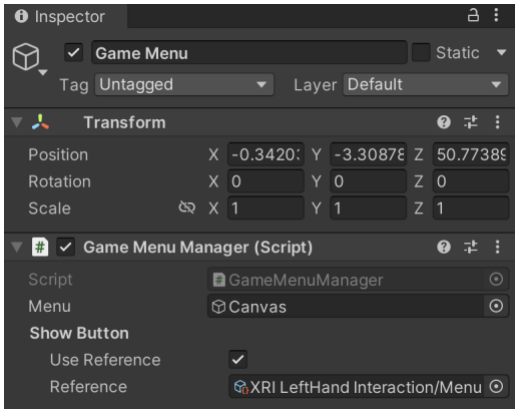
Com volem escoltar una acció, hem d'importar UnityEngine.InputSystem. Hem definit diverses variables: el cap (per saber la posició global del jugador), la distància a la qual apareixerà el menú des del cap del jugador, el menú i el botó per obrir-lo. Llavors, a la funció Update(), anirem actualitzant a cada frame l'orientació del menú, perquè miri cap al jugador (línies 21 i 22), i, si el botó per obrir el menú s'ha polsat en aquest frame, activarem o desactivarem el menú en funció de si ja estava o no activat. A més, també el col·locarem enfront del jugador i a la distància (per defecte 2) que haguem definit (línies 28 i 29).

Ara, anirem a la carpeta Starter Assets importada del XR Interaction Toolkit i obrirem el XRI Default Input Actions. Se'ns obrirà una pestanya on podrem veure i modificar totes les accions que podem realitzar. Si ens fixem en les accions dels comandaments, no hi ha cap predefinida per obrir un menú. Llavors, el que hem de fer és crear una acció, anomenada Menu, al XRI LeftHand Interaction, per tal de poder obrir el menú amb el comandament esquerre. Pel Binding, o vinculació, seleccionarem el menuButton del LeftHand XR Controller al Path, i li donarem a Save Asset. A la següent imatge es pot veure com quedaria l'acció afegida:



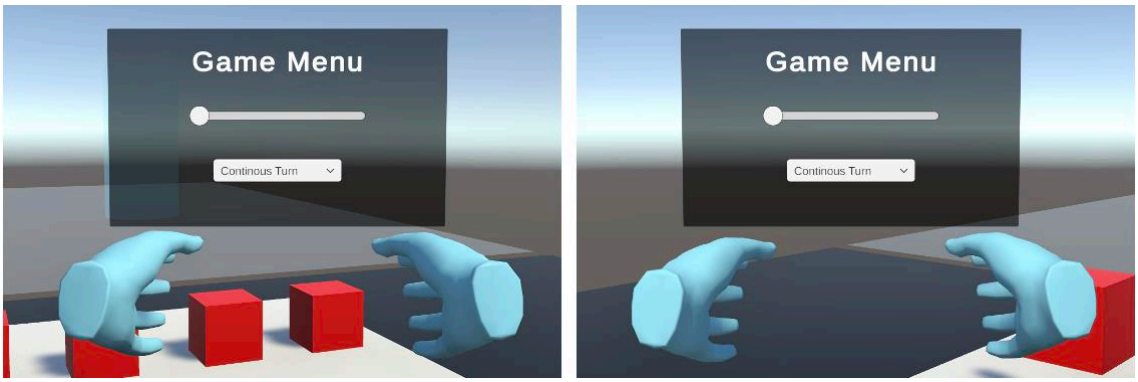
Imatge 84: Pestanya de configuració de l'XRI Default Input Actions on es mostra la nova acció afegida

Ara sí, només hem de seleccionar l'acció que acabem de crear al camp del botó del component que hem creat (GameMenuManager), que seria la XRI LeftHand Interaction/Menu, tal i com es pot veure a la següent imatge:



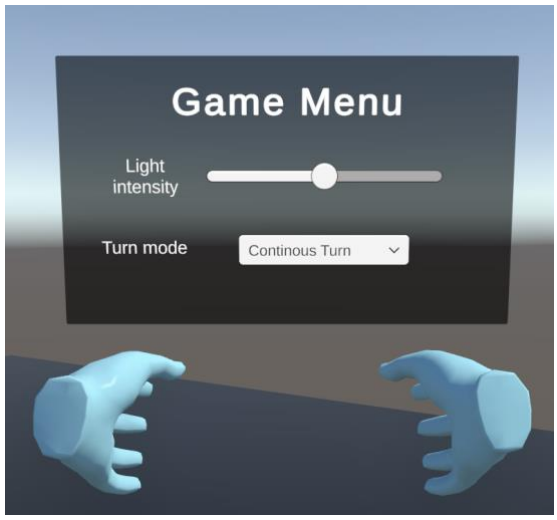
Imatge 85: Configuració del component GameMenuManager

Si provem a executar, veurem que podem fer aparèixer i desaparèixer el menú amb el botó corresponent. A més, encara que rotem el nostre jugador, el menú sempre apareixerà al seu davant i orientat cap a la càmera. A les següents imatges, podem veure com des de diferents angles s'obre el menú, orientat cap al jugador:



Imatge 86: Execució que mostra com, en obrir el menú amb el botó corresponent, sempre es troba orientat cap al jugador

Finalment, afegirem títols als elements del menú, de manera que quedaria així:



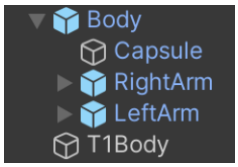
Imatge 87: Execució que mostra el menú completament acabat

4.3 Creació d'un cos amb els braços IK i exportació al projecte de VR

Fins el moment, tenim implementat un braç que es mou seguin un algoritme de Cinemàtica Inversa (IK) en 2D, i un joc de Realitat Virtual en el qual podem interactuar de diverses formes amb els elements de l'escena. Per acabar d'assolir el nostre objectiu final, volíem poder connectar d'alguna manera aquest braç perquè es mogués amb les ulleres i els comandaments de VR. El que hem fet, doncs, és crear una espècie de cos pel jugador amb 2 braços dels que hem implementat. D'aquesta manera, podrem donar la sensació de tenir braços que responguin als nostres moviments quan utilitzem el headset de VR.

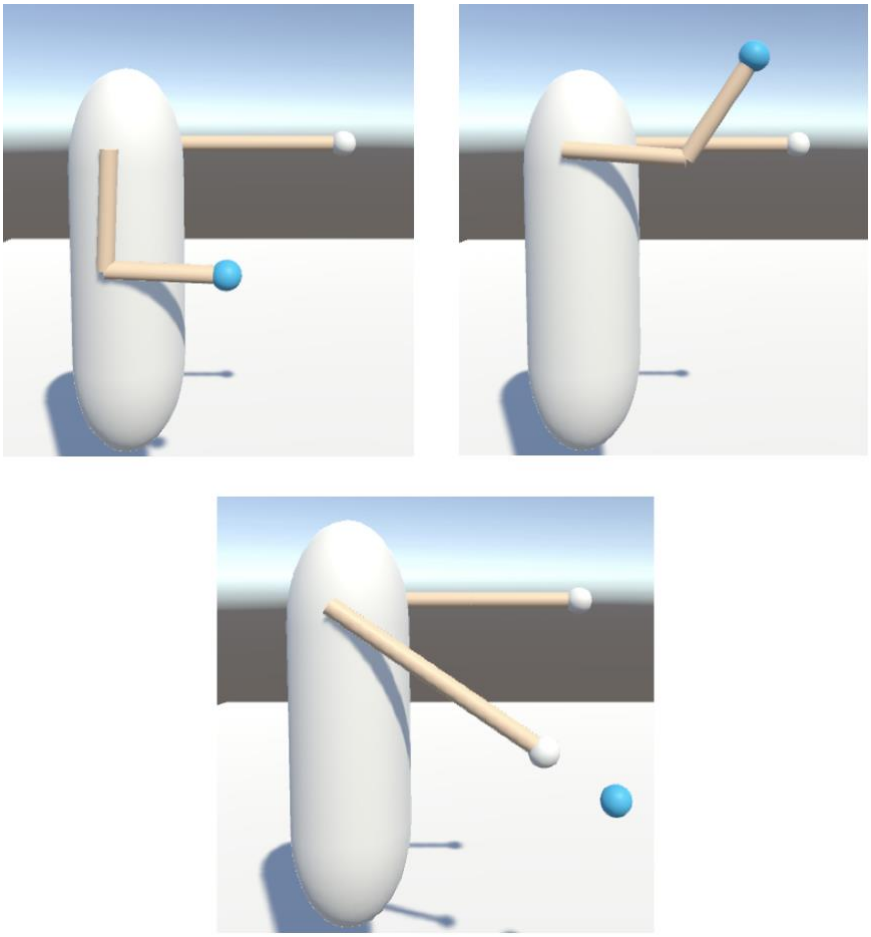
4.3.1 Creació del cos

El primer que hem de fer és crear el cos amb els braços. Afegirem un objecte buit a l'escena anomenat Body, en el qual afegirem una càpsula que simularà el cos del jugador/robot. A la càpsula, afegirem com a fills 2 rèpliques del braç que havíem creat i configurat amb moviment IK en 2D, col·locant-los un a cada costat. L'estructura del cos quedaria de la següent manera:



Imatge 88: Jerarquia del cos

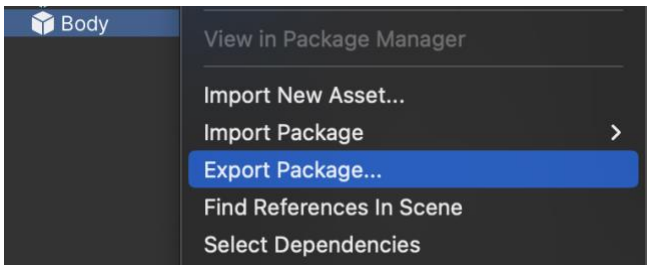
Com es pot veure a la imatge de dalt, també hem replicat l'esfera T1 i l'hem anomenat T1Body, per tal de poder provar el moviment d'un dels braços del cos. A les següents imatges, el que mostrem és com seria el cos sencer i com movent T1Body es mou, seguint-la, un dels braços.



Imatge 89: Visualització del cos i de com el braç dret es mou seguint T1Body, adoptant diferents configuracions

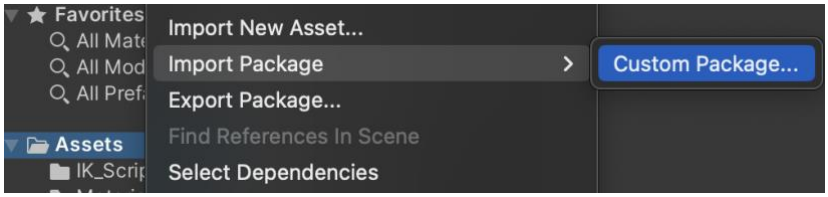
4.3.2 Exportació i adaptació del cos al joc de Realitat Virtual

Una vegada tenim el cos amb els braços funcionals, només ens falta exportar el cos d'un projecte a l'altre. Per fer-ho, el que hem de fer és arrossegar l'objecte Body, que engloba tot el cos, cap a la pestanya de l'explorador de fitxers dins de Unity. A continuació, farem clic dret a sobre seu i premerem Export Package... tal i com es mostra a la següent imatge:



Imatge 90: Exportació del cos del primer projecte

Seleccionarem la opció d'incloure totes les dependències del model. Fent això, no només s'exportarà el model del cos, sinó que també ho faran tots els scripts i models que estiguin relacionats amb ell (per exemple l'script IK_2). El guardarem amb el nom i a la ubicació que vulguem i ja el tindríem exportat. Seguidament, el que hem de fer és importar-lo a l'altre projecte (el del joc de VR). Per importar un paquet, el que hem de fer és seleccionar la carpeta on vulguem importar-lo i seleccionar Import Package > Custom Package, tal i com es mostra a la següent imatge:



Imatge 91: Importació del cos al projecte del joc

Seleccionarem el model del cos que hem exportat de l'altre projecte i importarem totes les dependències que tingui. D'aquesta manera, s'haurà importat el model del cos, a més de totes les seves dependències, inclosos els scripts.

Per últim, ens queda adaptar el model del cos a l'escena i fer les configuracions corresponents perquè segueixi el moviment del jugador i connectar els braços als comandaments perquè segueixi els seus moviments.

Arrossegarem el model a l'escena i el col·locarem sobre el pla. Per configurar-lo de manera que segueixi el moviment i orientació del jugador, hem creat un nou component anomenat FollowMovement, l'script del qual és el següent:

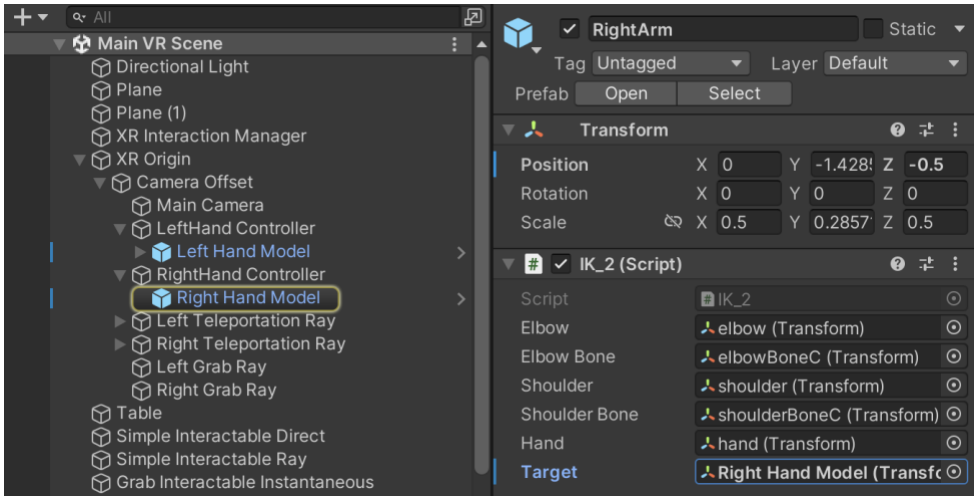
```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class FollowMovement : MonoBehaviour
6   {
7       public Transform headset;
8       public Transform body;
9       // Start is called before the first frame update
10      void Start()
11      {
12          body.position = headset.position;
13          body.rotation = headset.rotation;
14      }
15
16      // Update is called once per frame
17      void Update()
18      {
19          body.position = headset.position;
20          body.rotation = headset.rotation;
21      }
22  }
```

Imatge 92: Script del component FollowMovement del cos

És tan senzill com declarar dos variables de tipus Transform, una pel cap o càmera del jugador i una altra pel model del cos. Tant a la funció Start() com a la Update(), el que fem és assignar la posició i rotació del cap del jugador al model del cos, de manera que, en tot moment, el cos seguirà la posició del jugador.

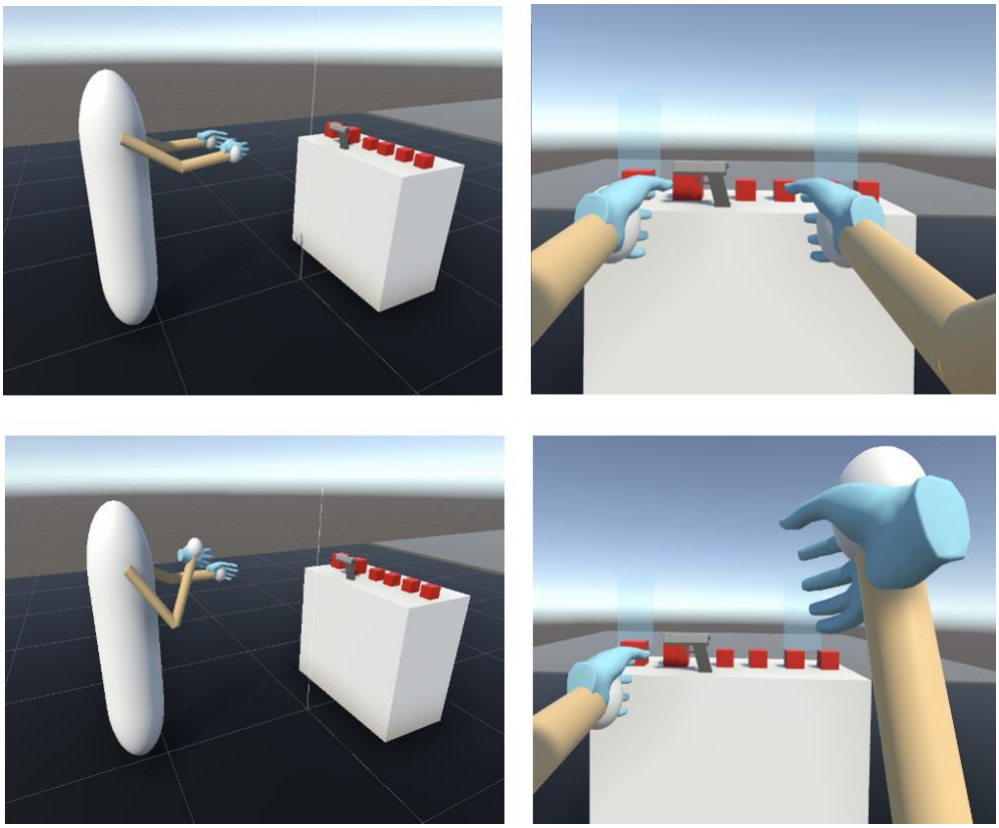
Cal dir que hem hagut de canviar la posició relativa de la càpsula, per tal de que quedi la part superior del cos centrada amb el cap del jugador (dependrà de com estigui configurat el model).

Finalment, per tal de configurar els braços, només hem de seleccionar l'objecte que es correspon amb el target del component IK_2 de cadascun. Es tracta dels models de les mans que utilitzem pels Hand Controller. A la següent imatge es veu la configuració del component IK_2 del braç dret:



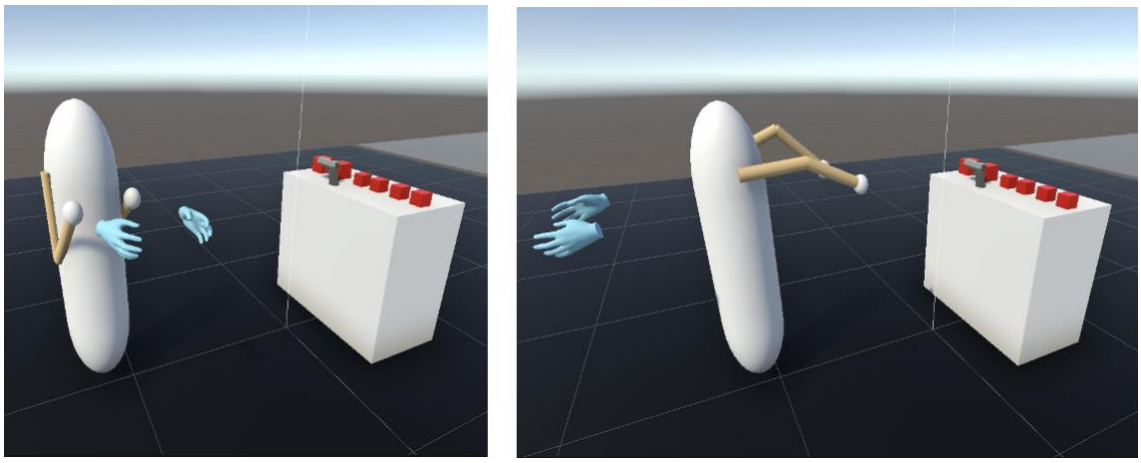
Imatge 93: Configuració del component IK_2 del braç dret

Una vegada configurat tot, si provem a executar el joc, veurem com el cos segueix la càmera del jugador, tant la posició com la rotació, i els braços segueixen les mans.



Imatge 94: Execució en la qual es mostra com el cos i els braços responen als moviments del jugador

Com el moviment dels braços està pensat perquè sigui en 2D, aquests només es mouran de manera correcta quan les mans es moguin en el mateix pla vertical que els braços. Per això, si rotem el jugador (la càmera), també rotarà el cos i, per conseqüència, els braços es mouran de manera que adoptaran configuracions no desitjades. Això es solucionaria una vegada s'implementés el moviment dels braços en 3D. A les següents imatges es pot visualitzar com, a l'hora de rotar el jugador, els braços tenen una configuració que no coincideix amb les mans:



Imatge 95: Execució que mostra com l'algorisme IK no funciona correctament quan rotem el cos

5. Conclusions i treball de futur

En aquest Treball de Fi de Grau, es va plantejar com a objectiu principal el següent: Desenvolupar una experiència de joc interactiva en un entorn de realitat virtual, per mitjà de la creació d'un braç amb el qual es pugui interactuar de diverses formes i amb diferents objectes, donant una experiència immersiva el més natural i realista possible. Els resultats obtinguts mostren de manera significativa que, finalment, l'hem assolit. Hem pogut crear un model d'un braç a Unity que es mou d'una manera força natural i realista, encara que sigui en 2D. També hem creat un petit joc de realitat virtual amb el qual podem interactuar amb els elements i objectes d'una escena, de diverses maneres. Finalment, hem ajuntat els dos projectes de manera que, creant i important al projecte del joc una espècie de cos amb 2 dels braços desenvolupats, podem tenir una experiència de joc interactiva i que ens dona una sensació més immersiva de formar part de l'escena virtual.

No obstant això, també hem de ser crítics tant amb el treball realitzat com amb el que no hem pogut realitzar. Mirant al futur, aquest treball té moltes possibilitats de millora, en molts aspectes. Primer de tot, la implementació del moviment dels braços en 3D és important per acabar de donar una experiència immersiva molt més realista. El cos del jugador també es pot millorar per tal que s'adapti millor a l'altura i rotació de l'usuari, a més de crear un model més "humà". Una vegada fet això, es podria anar millorant cada cop més l'aparença del jugador. Pel que fa al joc, també té hi ha moltes opcions de millora i ampliació. El joc que s'ha desenvolupat presenta les bases de les possibles interaccions que es poden fer. Així doncs, pensem que podria ser la llavor d'una gran varietat de possibilitats interactives, tant amb la finalitat d'entreteniment com per l'aplicació didàctica. Finalment, també es podria donar un altre enfoc relacionat amb els torques. Es podria acabar d'implementar el seu càlcul per tal de realitzar-ne estudis com els

mencionats al corresponent apartat, com per exemple una optimització del control dels braços.

L'altre objectiu secundari era aprendre sobre les bases del desenvolupament en l'entorn de la Realitat Virtual i, concretament, a adquirir uns coneixements d'una de les eines més utilitzades del sector, Unity. Podem dir amb orgull i satisfacció que hem après molt d'aquest àmbit tant ampli, encara que sigui una petita porció de tot el que es pot arribar a fer.

Per acabar, com a conclusió personal puc dir que ha estat un plaer haver pogut ser part d'aquest treball, en el qual he après i aplicat molts coneixements que sense dubte em seran de gran utilitat al llarg de la meva carrera professional.

6. Agraïments

D'una banda, vull expressar el meu sincer agraïment als meus tutors Ignasi Cos i Marc Soler per haver-me brindat l'oportunitat de dur a terme aquest treball. Els vull donar les gràcies per la seva dedicació i guia constant, que han estat essencials per aconseguir els objectius que ens havíem proposat. Els agraeixo de tot cor per la seva confiança en mi i espero haver assolit les seves expectatives.

D'altra banda, agraeixo a totes les persones que em van donar suport i em van aportar una font constant de motivació en els moments difícils, especialment familiars i amics.

Finalment, no puc oblidar expressar el meu agraïment a la Universitat de Barcelona, ja que ha sigut la institució que m'ha proporcionat uns coneixements i experiències d'aprenentatge, que sense ells no hagués pogut realitzar aquest treball.

7. Referències

https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

https://xrbootcamp.com/unity-vr-tutorial-for-beginners/

https://robotacademy.net.au/lesson/inverse-kinematics-for-a-2-joint-robot-arm-using-geometry/

https://www.alanzucconi.com/2018/05/02/ik-2d-1/

https://www.alanzucconi.com/2018/05/02/ik-2d-2/

https://www.alanzucconi.com/2020/09/14/inverse-kinematics-in-3d/

https://dspace.mit.edu/bitstream/handle/1721.1/6358/AIM-635.pdf?sequence=2&isAllowed=y

https://docs.unity3d.com/Manual/index.html

https://www.youtube.com/watch?v=fM0k2n7u8sc&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud

https://www.youtube.com/watch?v=HhtTtvBF5bI&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=2

https://www.youtube.com/watch?v=8PCNNro7Rt0&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=3

https://www.youtube.com/watch?v=0VowAem2aMM&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=4

https://www.youtube.com/watch?v=9pVdiBogmew&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=5

https://www.youtube.com/watch?v=0xt6dACM_1I&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=6

https://www.youtube.com/watch?v=nowlPXuPEG8&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=7

https://www.youtube.com/watch?v=yhB921bDLYA&list=PLpEoiloH-4eP-OKItF8XNJ8y8e1asOJud&index=8

https://docs.unity3d.com/Manual/CustomPackages.html

https://www.youtube.com/watch?v=HCZUwa3blQs

https://www.youtube.com/watch?v=_S2bNLpji2s

https://es.wikipedia.org/wiki/Momento_de_inercia

https://es.wikipedia.org/wiki/Momento_de_fuerza

https://docs.unity3d.com/es/2021.1/Manual/InverseKinematics.html