Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Desenvolupament d'aplicacions interactives de Realitat Virtual amb Leap Motion i Head Mounted displays

MEMÒRIA

Autor: Ruau Asprela, Mariano
Director: Susin Sánchez, Toni

Convocatòria: Septembre 2017

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest Treball de Final de Grau anomenat *Desenvolupament d'aplicacions interactives de Realitat Virtual a partir dels dispositius Leap Motion i Head Mounted displays* pretén investigar el potencial de les noves tecnologies d'interacció amb la realitat virtual, és a dir, entre persona i ordinador, i la possibilitat de combinar-les per generar una millor inversió al món virtual. D'altra banda, es pretén crear una sèrie de *teaser games* o aplicacions que permetin als usuaris interactuar amb entorns en 3D utilitzant una virtualització de les seves mans i amb un punt de visió en primera persona, tot això utilitzant el motor de videojocs Unity 3D.

En els últims anys, el hardware vinculats a la interacció amb la realitat virtual han augmentat les seves possibilitats d'actuació en els diferents camps de la simulació científica, a més d'augmentar també la qualitat de dita interacció. En aquest cas, el desenvolupament de les aplicacions del treball ha fet servir un dispositiu Leap Motion, encarregat de materialitzar les mans a l'escena i permetent una interacció amb l'entorn, i el casc de realitat virtual HTC Vive que posiciona a l'usuari en l'interior de l'escena amb una visió de 360 graus.

Per tal d'aprofitar al màxim les opcions que ofereixen aquest dos dispositius, les aplicacions creades al treball, s'han creat tres aplicacions controlades amb les mans del Leap Motion de caire molt diferent: Una aplicació d'escriptori on les mans interaccionen amb objectes de l'escena, una altra aplicació d'escriptori on l'usuari controla al personatge utilitzant gestos amb les mans, i una aplicació amb casc de realitat virtual a la qual l'usuari, utilitzant gestos, controlarà un automòbil que circularà per un circuit.

Sumari

1. Prefaci

La finalitat d'aquest treball és la creació d'aplicacions, creades gràcies al motor gràfic Unity3D, que permetin a l'usuari tenir una visió de l'escena en primera persona, gràcies al Head Mount Display, i interactuar amb ella utilitzant les mans, amb l'ús del Leap Motion.

Un dels objectius principals és analitzar les possibilitats que proporcionen cadascun dels dispositius per separat i la sinergia a l'hora d'utilitzar-los simultàniament.

Altre objectiu és descriure pas a pas com utilitzar el hardware escollit amb Unity 3D i explicar el procediment realitzat per la creació de les escenes del treball.

En el cas del Leap Motion, s'analitzarà una nova versió, la més recent, de la part de software del dispositius que ha provocat un canvi a la manera d'implementar els scripts i la manera d'utilitzar-lo amb el motor Unity 3D.

1.1. Motivació

El treball té com motivació principal estudiar les possibilitats que es poden obtenir de la combinació del dispositiu Leap Motion y els dispositius Head Mounted Display, analitzant diferents formes d'ús possibles generades per les diferents maneres de combinar-los.

A més d'una motivació personal que és les ganes de progressar i ampliar el coneixement de programació obtingut als estudis de Grau en Enginyeria en Tecnologies Industrials, havent d'aprendre un llenguatge de programació totalment desconegut i mai estudiat a la carrera com és C#, també influeix la passió pel món dels videojocs i les ganes d'introduir-se als mètodes de creació. El fet de ser dispositius tant versàtils, fan que el projecte sigui atractiu per les possibilitats que proporciona a l'hora de combinar-los, que són pràcticament infinites.

1.2. Problemes trobats

A l'inici del projecte, on es va estudiar i treballar purament amb el dispositiu Leap Motion, no es va tenir cap problema donat que és un aparell portàtil que permet treballar amb ell fora del departament que facilitava el material (Centre de Realitat Virtual, FME, UPC) i amb una disponibilitat major. El problema principal va ser amb l'ús del dispositiu en la nova versió *Orion Beta*, una versió de software encara en proves que generava moltes dificultats de compatibilitat, errors de compilació dels scripts dels desenvolupadors, i el més problemàtic,

problemes de qualitat entre el software y Unity3D que requerien canviar les propietats del projecte per tal de obtenir-ne un funcionament òptim sense cap coneixement sobre rendiments dels programes i el hardware de Leap Motion ni compatibilitats entre tots dos. A una etapa avançada del treball, el software va actualitzar la seva API i va arreglar molts d'aquests problemes, però també va provocar que, a causa d'aquest fet, em fos impossible crear una interfície de joc, donat que no serien escenes de la mateixa versió.

En fases més avançades del projecte, la unió de Leap Motion i Head Mounted Display no va ser tan complicada per que els assets dels cascs estan testejats i en una versió molt més avançada que les del Leap Motion, el problema principal, que va obligar en part a prorrogar l'entrega del treball, va ser la disponibilitat del dispositiu al departament, que estava ocupant projectes vinculats al dispositiu i no desponien per desgracia de cap que no estigués en ús. No obstant, i gràcies als coneixements assolits a la primera fase de la realització del treball, la creació de l'aplicació que combinava el casc VR amb el Leap Motion no va resultar tant difícil de crear.

2. Introducció

2.1. Objectius del projecte

La finalitat del projecte és obtenir informació sobre dos hardwares de realitat virtual que actualment es troben en fase de desenvolupament que són els Head Mounted Display, o cascs de realitat virtual, i el Leap Motion, capaç de virtualitzar les mans de l'usuari o jugador en un escena tridimensional. A més, per tal d'aplicar els coneixements obtinguts i demostrar que realment han estat assolits, és necessari una demostració pràctica amb la creació d'una aplicació que combini l'ús de tots dos simultàniament i de la manera més òptima, cercant la màxima inversió possible al món de la realitat virtual. Per aquest motiu, una altre finalitat del projecte és la creació de diverses aplicacions/escenes de visualització que combinin tots dos aparells. Tot i que el treball es basa en la combinació, al ser Leap Motion un hardware més recent amb més dificultat a l'hora de la seva programació, algunes de les aplicacions estan centrades únicament en aquest aparell.

D'aquesta manera, el treball també suposarà crear un informe tècnic que permetrà entendre de manera detallada el funcionament mecànic dels aparells, la lògica de programació i una breu introducció a la seva història.

2.2. Abast del projecte

Pel que fa a l'abast de les aplicacions del treball, no es pretén la creació d'aplicacions complexes ni un *serious game*, es pretén crear una sèrie d'escenes que mostrin el potencial dels dispositius i el programa utilitzats. El temps i la manca de coneixements i formació sobre l'ús i la programació en C# per Unity3D, generen una complexitat major a la creació dels jocs, a més del temps per familiaritzar-se amb la interfície.

Aquesta familiarització inclou adaptació als mètodes d'implementació, acostumament al llenguatge de programació, estudiar les possibilitats que proporciona l'ús de la API de Unity3D i Leap Motion, i finalment, i de forma més general, aprendre a optimitzar scripts i aprendre a fer *clean code.* Per aquest motius no es pretén crear un joc complet.

3. Introducció històrica a la Realitat Virtual

La realitat virtual es defineix com un entorn generat mitjançant la tecnologia capaç de crear sobre l'usuari la sensació d'immersió en aquest. Tot i que ens trobem en un moment on el desenvolupament i la inversió en realitat virtual es troba al seu auge, el primer sistema de creat específicament per simular una realitat virtual es va crear l'any 1935 per Stanley G. Weinbaum, i es tractava d'un sistema anomenat les 'gafas de Pigmalion', les quals contenien gravacions hologràfiques i incloïen sensacions d'olor i tacte. A partir dels seus inicis, la tecnologia basada en al realitat virtual es va desviar cap a la simulació i inversió audiovisual, donat que per una raó morfològica, el cos rep la major part de la informació de la natura per aquest dos sentits. Això centra la realitat virtual en la cerca de la perfecció en la creació d'un ambient que faci a l'usuari sentir-se igual que en un ambient real, l'inconvenient és que a més de ser els més importants per la immersió, els ambients capaços de recrear aquest dos sentits també són els més difícils de recrear.

A la dècada dels 50, una època on el món occidental es trobava en prosperitat econòmica, es va produir un fenomen conegut com la revolució digital, a la qual es va començar a invertir en investigació i desenvolupament d'aparells digitals i computadors. En aquesta època, les tecnologies de realitat virtual no van progressar significativament, donat que la tecnologia de la època encara no permetia realitzar simulacions de qualitat, i que no hi havia un objectiu viable per la seva implementació. Tot i així, es van crear diversos dispositius de simulació per immersió utilitzats per al món teatral i el cinema. El més destacable va ser el Sensorama, utilitzat a l'obra teatral de Morton Heilig, que s'encarregava de reproduir estímuls visuals, olfactius, auditius i tàctils durant una sessió de cinema.

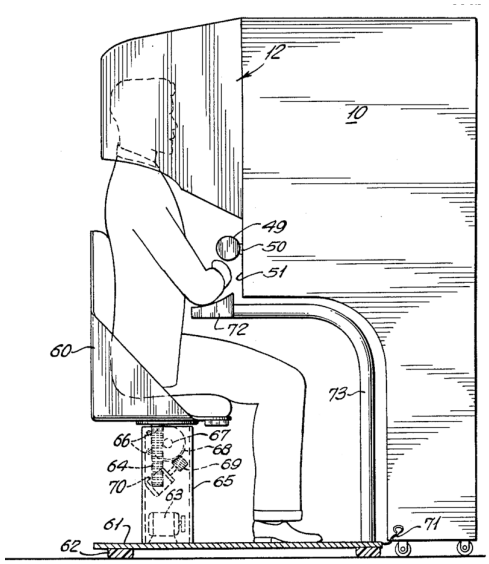


Figura 1. Imatge esquemàtica del sensorama

Entre la època del 1970 i la del 1990, amb l'avanç de la tecnologia en computadors, la realitat virtual es va convertir en l'objectiu de moltes empreses del sector de la investigació informàtica, i va ampliar molt el seu abast, arribant a la creació de dispositius per usos clínics, simuladors de vol, dissenys industrials, automobilístics i militars.

A dia d'avui, el mercat de la realitat virtual ha anat creixent i ampliant els seus camps d'aplicació, tant és així que en aquest últim any (2016-17) el mercat de la realitat virtual ha arribar a duplicar el seu valor, i s'estima un creixement d'un 600% fins el 2020.

3.1. Història dels cascs de realitat virtual

Un Head Mounted Display és un dispositiu de visualització, similar a un casc o unes ulleres, que permet reproduir imatges creades amb un ordinador amb unes pantalles properes als ulls de l'usuari. El primer aparell Head Mounted Display que es coneix, va ser creat l'any 1968 per Ivan Sutherland, un casc primitiu, de poca qualitat i d'un pes que el feia poc pràctic, però va inspirar a la creació de tots els cascs que coneixem avui dia.

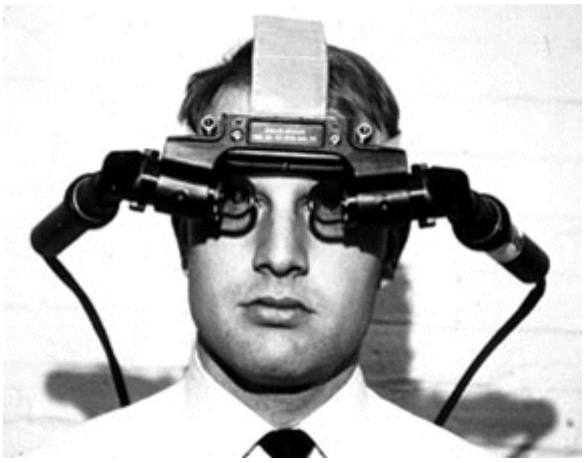


Figura 2 Fotografia del casc de Sutherland

Aquest tipus de dispositiu ha estat impulsat al llarg del final del passat segle fins a dia d'avui per l'industria del videojoc. Les primeres ulleres comercialitzades de l'historia van ser les ulleres Sega VR a l'any 1991, per a videojocs de la consola Mega Drive, però la verdadera revolució amb impacte als mercats va ser l'aparició de l'Oculus Rift, un model de casc que va impulsar la inversió en aquest tipus de tecnologia per part d'un gran nombre d'empreses. Tant va ser la pretensió de crear el millor model, que el preu de venta del model Oculus Rift, encara en fase de proves, va ser de 2 bilions de dòlars, i va ser adquirit per Facebook.

Figura 3 Imatge del casc Oculus Rift

Avui dia, l'expansió del mercat ha provocat l'aparició de models convencionals amb preus assequibles, lògicament molts d'ells amb qualitats molt baixes, i es considerat un producte de consum familiar.

3.2. Aparició del dispositiu Leap Motion

El dispositiu Leap Motion es va començar a desenvolupar a l'any 2008 per David Holz, co-fundador de l'empresa que dona nom a l'aparell Leap Motion Inc. El primer llançament al mercat del dispositiu es va produir l'any 2012, però no va ser fins l'any 2015 que es va crear el primer kit de creació d'aplicacions pels usuaris. Actualment, encara s'actualitza el software del dispositiu a la versió Orion creada específicament per la combinació amb aparells VR, inaugurada al Febrer de 2016.

Figura 4 Dispositiu Leap Motion

3.3. Tècniques de realitat virtual

Per provocar la sensació de realitat sobre l'usuari, les tècniques més comunes i més utilitzades són:

- **Seguiment del cap:** L'aplicació ha de ser capaç de reconèixer els moviments de l'usuari i provocar un desplaçament de la imatge en funció de la direcció de dit moviment. Pel seguiment dels moviments, s'utilitzen aparells de seguiment com els acceleròmetres, giroscopis i magnetòmetres. Aquest tipus de tècnica, és la utilitzada als casc de realitat virtual com el HTC Vive.

- **Rastreig del moviment:** És una extensió del seguiment del cap, en aquest cas, permet reconèixer moviments més complicats, com poden ser els moviments de les extremitats. Aquest tipus de tecnologia no es troba tan desenvolupat com l'anterior. Un dels sistemes de detecció més prestigiosos en aquest camp és el Leap Motion.

- **Seguiment ocular:** Tecnologia capaç de detectar els moviments dels ulls de l'usuari, permetent-lo enfocar únicament el que l'usuari desitgi. Actualment no

existeix cap dispositiu que sigui capaç de dur a terme aquesta tasca, és una tecnologia que encara es troba en una fase molt inicial del seu desenvolupament.

3.3.1. Principals problemes de la realitat virtual

Avui dia, el principal problema d'aquest tipus de tecnologia és la manera en la que afecta a la salut del jugador, experimentant sensacions de mareig i nàusees, la causa d'aquest problemes és el desajust entre el sistema vestibular, líquids i fluids a les cavitats de l'oïda, i el sistema visual. Algunes de las causes principals de estos efectes solen ser:

- Latència: L'augment en el retard entre les l'enfocament de l'usuari i la representació d'aquestes a la pantalla, provoca un major desajust entre el sistema vestibular i el visual.

- Duplicació de les imatges i la persistència: La combinació de l'esborronament i l'estroboscòpia. Aquest problema, es podria resoldre amb l'augment del nombre de fotogrames per segon a un nombre semblant al que és capaç de captar l'ull humà, el principal problema és que la tecnologia no està preparada per resoldre aquest problema. Seria necessària una simulació amb 1000FPS, quan, a dia d'avui, difícilment amb ordinadors preparats es por arribar als 200FPS.

4. Tecnologia dels dispositius

4.1. Dispositius HTC Vive

El casc de realitat virtual HTC Vive va ser creat per les empreses HTC, coneguda empresa tecnològica i una de les més conegudes al mercat de creació de mòbils, i l'empresa Valve, famosa empresa de videojocs i actualment posseïdora de la major plataforma de venta videojocs online del món (Steam). Va aparèixer per primer cop al març del 2015 a l'event del Mobile World Congress de Barcelona, i després d'un any, el 5 d'abril de 2016 es va comercialitzar al públic. Actualment, és considerat el millor casc de realitat virtual en quant a potència i qualitat.

Els aspectes generals de les especificacions tècniques del producte indiquen:

- la seva freqüència d'actualització és de 90Hz.
- consta de dues pantalles AMOLED, cada una posicionada per ser captada per cada ull de l'usuari, amb una resolució de 1080x1200, i un *framerate* limitat als 90fps.
- Utilitza més de 70 sensors, entre els que s'inclouen acceleròmetres i sensors làser. Aquest sistema de seguiment s'anomena Light House i funciona a través de fotosensors que s'utilitzen pel seguiment d'objectes. Amb Vive, utilitzem dos Light House que enfoquen tot l'espai de joc.
- Incorpora una càmera frontal encarregada de la detecció d'objectes en l'àrea, és una mesura de seguretat per evitar possibles col·lisions causades per la desorientació del jugador.

Figura 5 Dispositiu HTC Vive

- Pel seu ús és necessari utilitzar software oficial gratuït (HTC Vive) i, a través d'aquest, completar el calibrat dels aparells i la comprovació del requisits físics. Es requereix una sala que permeti un àrea mínima de joc de 2m x 2m.

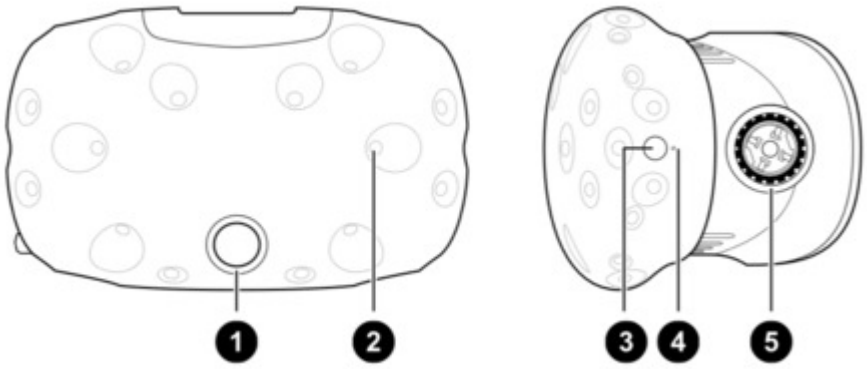
4.1.1. Components de hardware

El conjunt complert d'HTC Vive, està format per tres positius principal, un *headset* (HMD) de visualització panoràmica i àudio envolvent, dos *controllers* que permeten interaccionar amb l'entorn de realitat virtual, i dues *base stations* encarregades de la detecció constant de l'usuari i ubicar-lo a l'entorn.

4.1.1.1. Headset

El *Headset* s'encarrega de la part visual, representa l'escena a l'usuari un cop incorporades. L'usuari les col·loca al seu rostre a mode d'ulleres on, en comptes de cristalls trobem dues pantalles. Té un pes aproximat de 0.6kg, incloent ajustadors i tires de sustentació. Les parts i les seves respectives propietats en detall d'aquest dispositiu són:

- Dues pantalles AMOLED, resolució 1080p amb un *framerate* limitat als 90fps.
- Dos ports USB3.0, un HDMI, un port àudio jack 3.5mm i un port d'alimentació DC.
- Un micròfon, situat sota la part frontal del casc, i una càmera frontal
- Sensors de proximitat, acceleròmetres, giroscopis i posicionament làser.



1	2	3	4	5
Càmera	Fotodíodes	Botó d'encès	Status Light	Ajustador

Figura 6 Esquema del casc HTC Vive (frontal/lateral)

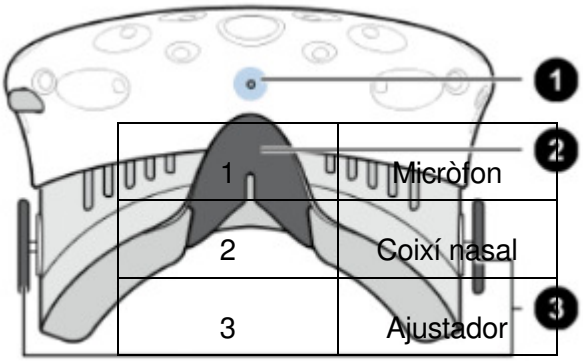
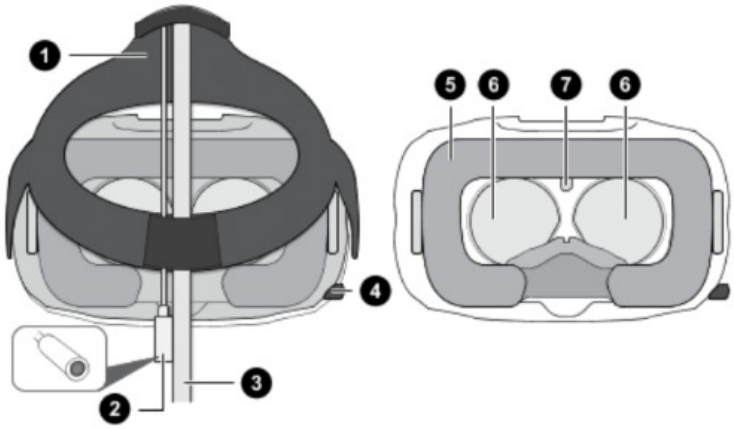


Figura 7 Esquema de la part inferior del casc



1	Tira de sustentació
2	Port d'àudio
3	Ports USB3.0 i HDMI
4	Ajustador de distancia usuari-lents
5	Coixí facial
6	Pantalles/Lents
7	Sensors de moviment i proximitat

Figura 8 Esquema de la part inversa del casc

4.1.1.2. Base stations

Les anomenades base stations s'encarreguen del rastreig de l'usuari a l'espai de joc. Aquestes generen aquest espai a través de detecció làser que s'encarreguen del rastreig dels fotodíodes del casc a l'interior de la seva zona d'abast.

Figura 9 Imatges de la base station

Aquest dispositius són electrodinàmics, generant un camp de làsers dinàmic gràcies al moviment d'aquests que genera l'aparell. Aquest moviment el provoquen dos motors (per base) de rotació. Això provoca que es rebi una informació més precisa i actualitzada de les dades del rastreig, ja que el dispositiu re-calcula en cada moment la posició i redueix qualsevol *delay* a la detecció que podria ocasionar-se.

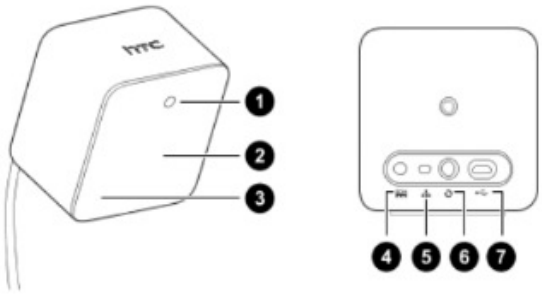


Figura 10 Esquema de parts de la base station

1	Status Light
2	Panell frontal
3	Indicador de canal
4	Port d'alimentació
5	Botó de canal
6	Port de sincronització
7	Port Micro-USB

4.1.1.3. Controllers

Els *controllers* o controladors s'encarreguen de que l'usuari pugui interactuar amb la realitat virtual amb accions en el món real. El disseny consta de diversos botons amb diferents funcionalitats que permeten a l'usuari interactuar amb l'API i creen un efecte sobre l'ambient virtual, cadascun diferent a l'altre. L'únic que no posseeix una funció dins del joc i és purament d'ús necessari pel *setup* de l'aparell, és el Trackpad, que s'encarrega d'iniciar la localització dels dispositius a la zona de joc.

Figura 11 Fotografia d'un controller

La detecció dels controllers es basa en un sistema de fotodíodes que envolta tota l'anella del controlador, així com passa amb el casc, els fotodíodes interactuen amb les bases i el seu camp làser i es genera un càlcul de la seva posició relativa a aquestes.

1	Botó del menú
2	Trackpad
3	Botó del sistema
4	Status Light
5	Port Micro-USB
6	Fotodíodes
7	Trigger (gatell)
8	Botó de Grip

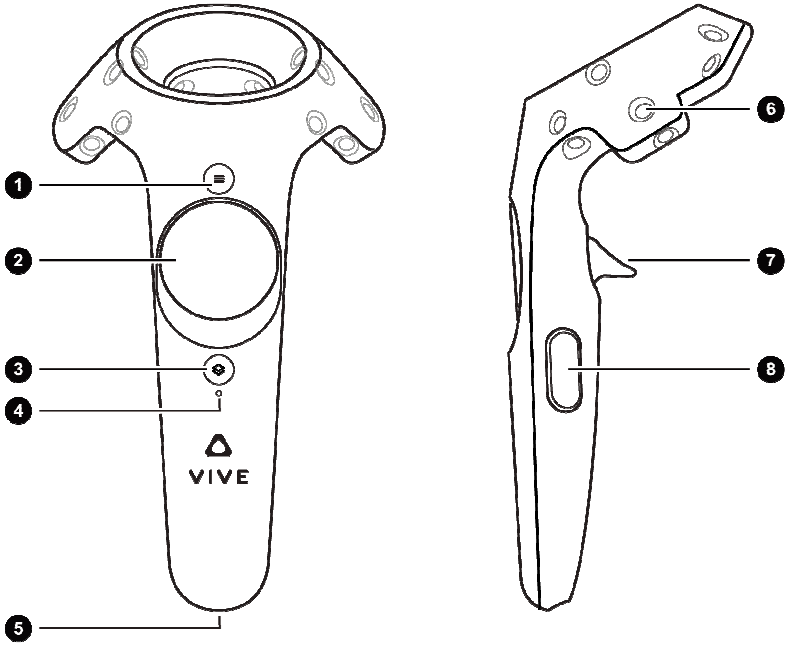


Figura 12 Esquema de parts d'un controller

4.1.2. Tecnologia de funcionament

El dispositiu HTC Vive principalment es basa en la combinació d'un sistema de visió estereoscòpica i una sistema de rastreig, o tècnica tracking, amb un subsistema que s'encarrega dels aspectes de seguretat a la zona de joc.

4.1.2.1. Ulleres de visió estereoscòpica

Les ulleres del dispositiu són les encarregades de proporcionar a l'usuari la visió estereoscòpica, que és la integració de dues imatges que a través del cervell humà acaben generant una única que avarca les dues. Les ulleres simulen aquest fenomen creant dues imatges a dues pantalles convexes, una per a cada ull, que permeten generar la sensació de profunditat i volum. Les ulleres proporcionen a l'usuari un rang de visió sobre l'escena d'aproximadament 110 graus. Gràcies als ajustadors dels casc, a més, podem calibrar la distància entre pupil·les per a major precisió.

Figura 13 Fotografia de les pantalles del casc

4.1.2.2. Sistema Lighthouse

El sistema Lighthouse és el sistema de tracking, tant de les ulleres com dels controladors, que utilitzen les ulleres de realitat virtual de HTC Vive. El sistema es basa en la generació de làsers per part de les *base stations* que es distribueixen per tota la zona de joc, delimitant un àrea de joc. Els dispositius amb fotodíodes, són capaços de captar l'energia d'aquest làsers, i transmetre la informació sobre la recepció al PC, que els posiciona a l'espai.

Aquests làsers s'emeten amb una longitud d'ona imperceptible per l'ull humà. Com s'ha dit anteriorment, els dispositius base no són estàtics, emeten els làsers situant-los al voltant d'un motor rotor. La velocitat de gir d'aquests rotors es aproximadament de 3600rpm.

Es podria dir que les *base stations* il·luminen la zona de joc on es trobaran els jugador, i dita llum, quan impacta amb algun cos, genera la informació necessària perquè sigui possible calcular la posició relativa a l'espai de joc del jugador. El càlcul de localització de l'objecte utilitza el temps transcorregut entre recepcions de llum per part dels fotodíodes, un cop s'obté aquesta informació es calcula la distància a la que es troba el fotodíode, o distancia a la que s'ha donat el contacte amb la llum, i, finalment, la posició relativa a les bases del fotodíode. Per aquests motius, és important que s'utilitzi el dispositiu utilitzant les bases sense objectes bloquejant la trajectòria de la llum que impedeixin que impacti als fotodíodes.

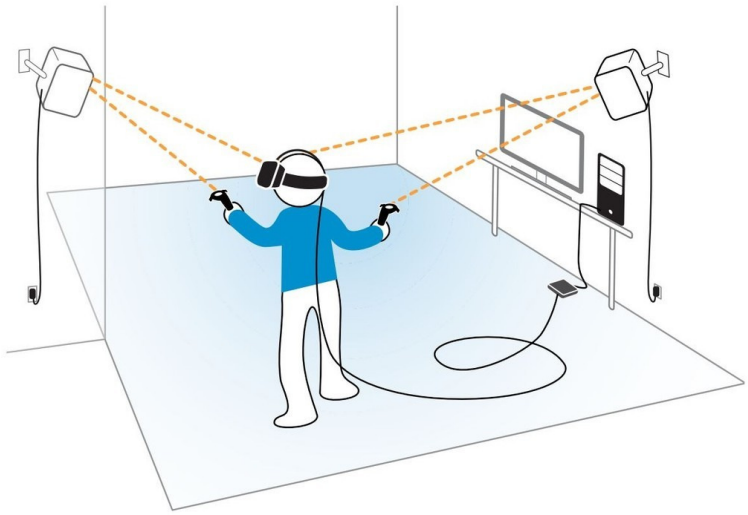
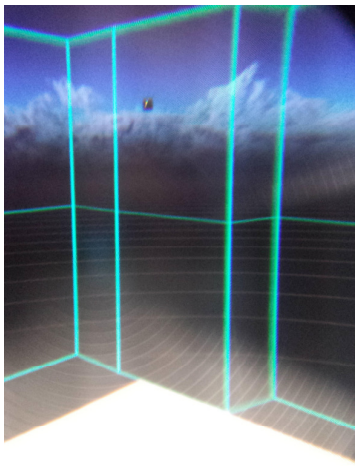


Figura 14 Esquema d'una habitació ben configurada

4.1.2.3. Mallat de Chaperone



El mallat de Chaperone suposa el sistema de seguretat del dispositiu, i pretén informar a l'usuari d'un possible impacte o sortida de l'àrea de joc, imperceptible pel jugador immers a la realitat virtual. Consisteix en la generació d'un recinte delimitat per una cel·la blava generada posteriorment a la col·locació i la libració de les bases. D'aquesta manera, el dispositiu és capaç d'avisar a la persona de la seva posició al món real. Aquesta malla es pot configurar i fins i tot desactivar si l'usuari ho desitja.

Figura 15 Imatge d'un mallat de Chaperone

4.1.3. Especificacions tècniques de funcionament

Tot el sistema HTC Vive i els seus components necessiten un ordinador encarregat de processar la informació de les aplicacions i de la generació gràfica de l'ambient. Donat que es tracta d'un dispositiu de tecnologia avançada, les recomanacions dels creadors pel que fa als requeriment del PC encarregat d'executar les aplicacions són les d'un ordinador per sobre de la mitja en quant a targeta gràfica, processador i memòria RAM, ja que el gràfics han de renderitzar-se a una resolució de 1800p i 90 fps. Les característiques mínimes són les següents:

Targeta gràfica	Nvidia GeForce GTX 970 / AMD Radeon R9 290 o superior.
CPU	Intel Core i5 4590 / AMD FX 8350 o superior.
RAM	4 GB o més.
Sortida de vídeo	HDMI 1.4 / DisplayPort 1.2, o una versió més recent
Ports USB	1 USB 2.0 o superior
OS	Windows 7 SP1 o més recent

Taula 1 Requisits per l'ús de HTC Vive

HTC Vive, juntament amb Steam, van crear el software necessari per la instal·lació del dispositiu i per l'execució de les aplicacions. El software s'anomena Steam VR, descarregable des de l'aplicació de Steam per PC. A través d'aquest software també es realitzen els passos de calibratge i referència de la posició de l'ordinador a l'àrea de joc.

Els requisits d'espai, diuen que les dues estacions han de col·locar-se a una distància màxima de 5 metres entre elles, i la superfície de joc ha de tenir unes dimensions de 2x2m.

Figura 16 Esquema de requeriments d'espai

4.1.3.1. Software necessari

Per l'ús de l'HTC Vive és necessari la descàrrega a l'ordinador al que estarà connectat des d'un dels dos instal·ladors disponibles a la web oficial d'HTC. Un d'ells és específic per la configuració i posada a punt del dispositiu, amb el nom de Vive Setup, inclou la instal·lació de tot el necessari per l'us HTC Vive. L'altre en canvi serveix per la descarrega d'un programa per personalitzar l'ús del dispositiu i inclou l'escriptori d'aplicacions del dispositiu HTC Vive, on es pot cercar i descarregar un gran número d'aplicacions enfocades al joc amb el dispositiu. Aquesta segona rep el nom de Viveport, i la seva instal·lació es troba inclosa a la del Vive Setup.

Amb el Vive Setup, es realitza primerament una comprovació de la capacitat de l'ordinador amb el que s'utilitza i es dóna una recomanació sobre si es convenient o no l'ús del dispositiu al dit ordinador. Seguidament, inicia la descàrrega dels softwares necessaris per la utilització del dispositiu, que són el client de Steam juntament amb l'aplicació SteamVR i el Viveport o Vive.

Pel que fa al dispositiu SteamVR, és un programa utilitzat per la configuració, inicialització i control de requisits mínims del dispositiu HTC Vive. És l'encarregat de detectar si tots els requisits de hardware s'estan assolint i de permetre, o no, l'ús del dispositiu. També detecta errors de software o de capacitat del dispositiu.

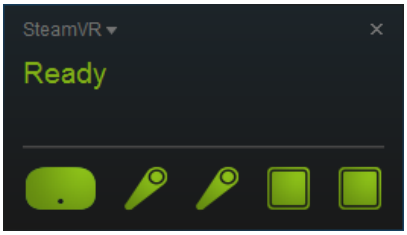


Figura 17 Imatge del programa Steam VR

El dispositiu s'encarrega de la detecció dels components del sistema, que són a la figura 17 de esquerra a dreta, les ulleres, els controladores i les bases de detecció. Si no es compleix un mínim de dispositius actius, el programa no permet l'ús d'aquest ni la connexió a l'ordinador.

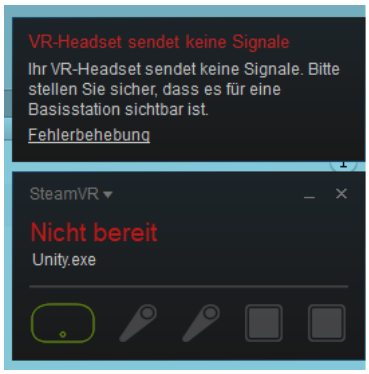


Figura 18 Detecció d'un error al programa Steam VR

També inclou un menú de configuració tant tècnica com gràfica, i un resum de la capacitat de l'ordinador de funcionar, tenint en compte les especificacions mínimes. Un dels requisits necessaris per poder començar a utilitzar el dispositiu és que és necessari que l'usuari superi el Room Setup, on configurarà totes les parts del dispositiu de manera òptima.

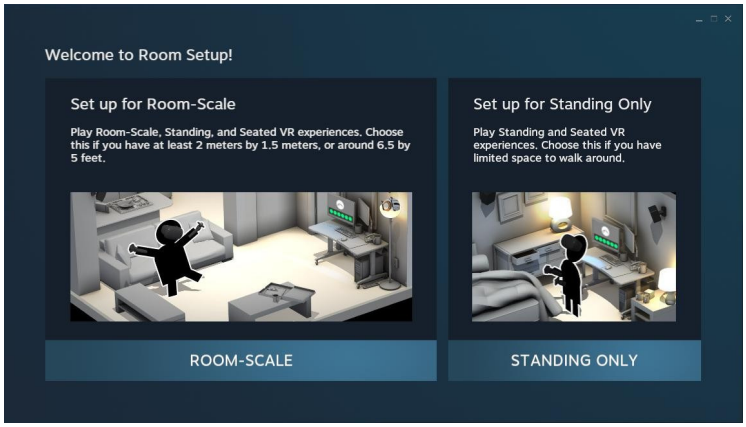


Figura 19 Imatge del Room SetUp

El programa Vive o Viveport, en canvi, és la botiga oficial per la compra d'aplicacions pel dispositiu. Una de les opcions que ofereix aquesta aplicació és que es por utilitzar directament des de la realitat virtual i seleccionar les aplicacions a utilitzar i fins i tot realitzar compres immersos en el dispositiu.

Figura 20 Imatge del Viveport en la seva versió VR

4.2. Dispositiu Leap Motion

El dispositiu Leap Motion és un dispositiu de reconeixement gestual de les mans humanes. Aquest fet, fa que sigui un dispositiu que pot utilitzar-se en aplicacions de camps molt variats, des de models pilots de pràctica en treballs delicats, com per exemple practiques d'enginyeria, fins a jocs d'entreteniment que facin ús de les mans humanes a la escena virtual.

4.2.1. Hardware

Es tracta d'un dispositiu d'unes dimensions de 75 mm de llarg, 25 mm d'ample i 11 mm d'alçada. Per la seva connexió a ordinadors, el dispositiu incorpora una entrada de cable específica que permet connectar-lo a ports USB3.0.

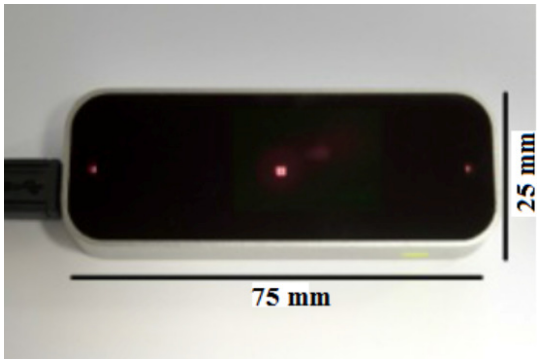
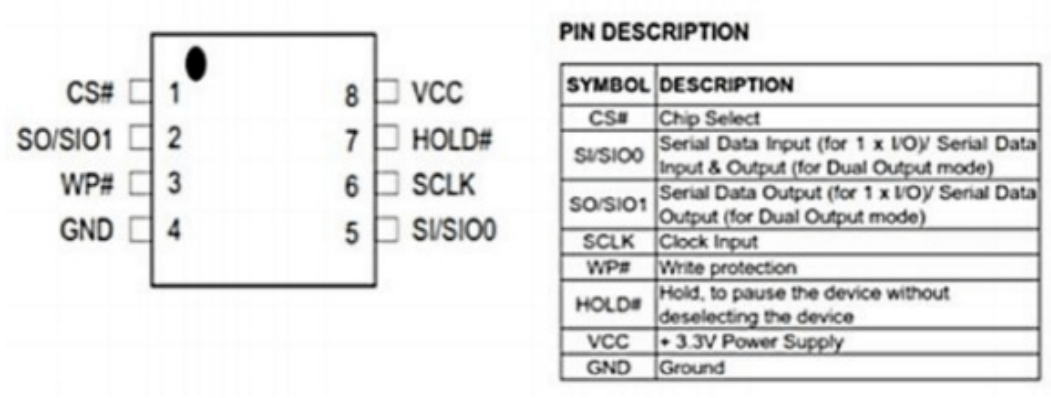


Figura 21 Dimensions del Leap Motion

Al seu interior, incorpora dues càmeres amb sensors mónocromàtics, per llum ultraroja de longitud d'ona 850nm, que treballen fins a 200 frames per segon, i tres emissors de llum ultraroja a 850nm de longitud d'ona.

Figura 22 Imatge de la placa del Leap Motion

També, està format per un microcontrolador, com els que es solen encarregar del funcionament de la BIOS, i que en aquest cas s'encarrega del control total de les funcions i parts del dispositiu, així com d'administrar la memòria. Aquest microcontrolador rep el nom de MXIC MX25L3206E-32M bits CMOS SERIAL FLASH.



PIN DESCRIPTION

SYMBOL	DESCRIPTION
CS#	Chip Select
SI/SIO0	Serial Data Input (for 1 x I/O)/ Serial Data Input & Output (for Dual Output mode)
SO/SIO1	Serial Data Output (for 1 x I/O)/ Serial Data Output (for Dual Output mode)
SCLK	Clock Input
WP#	Write protection
HOLD#	Hold, to pause the device without deselecting the device
VCC	+ 3.3V Power Supply
GND	Ground

Figura 23 Imatge del component MXIC MX25L3206E-32M bits CMOS SERIAL FLASH

4.2.2. Tecnologia de funcionament

El dispositiu és capaç cobrir un camp de visó molt ampli gràcies a les lents que cobreixen les seves càmeres. A la seva versió més recent, Orion beta, la utilitzada al treball, detecta la posició de les mans fins a una profunditat de 85cm, amb una il·luminació apropiada.

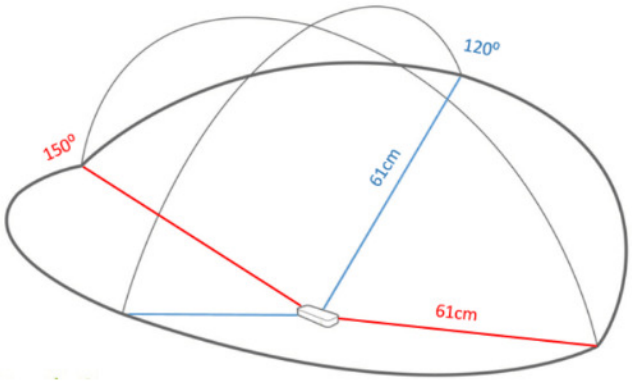


Figura 24 Esquema de l'angle de detecció

El funcionament de l'aparell, consisteix en la il·luminació del camp de visió gràcies a la llum dels tres LEDs. La forma final i les dimensions del camp de visió del dispositiu depèn de la intensitat màxima capaç de lliurar la connexió, cosa que fa important el tipus de connexió USB utilitzada, i l'angle de visió de les lents. D'aquesta manera, l'angle de visió del dispositiu segueix la formula:

$$\alpha = 2 \cdot \arctan\left(\frac{d}{2 \cdot f}\right)$$

Figura 25 Equació de l'angle de visió del Leap Motion

(d: diagonal del sensor ; f: distància focal)

El dispositiu, detecta si es produeixen reflexions de llum causades per un cos opac, donat que aquest rajos tornen a les càmeres. El funcionament és semblant al dels interruptors de reflexió de cos opac amb electrodiodes. Les lents biconvexes s'encarreguen de concentrar aquests rajos de llum, a través dels quals, recopilant informació de la llum rebuda, digitalitza la imatge de les mans.

Figura 26 Imatge de les càmeres del Leap Motion

La informació rebuda es mesura segons el valor de lluminositat que es rep del cos per cada píxel de la imatge de les càmeres. La finalitat de la informació és generar-ne dues imatges de 640 x 120 píxels, que posteriorment es barrejaran per crear la profunditat de les mans. Existeixen fins a un total de 256 valors diferents d'intensitat. El valor d'intensitat es mesura amb 8 bytes, dos dígits hexadecimals, i gràcies a aquest es genera una imatge en escala de grisos. En primera instància, es produeix una distorsió notòria sobre l'objecte a la imatge, per aquest motiu, es necessària una correcció d'aquesta distorsió abans de procedir a la identificació de les mans.

Figura 27 Distorsions representades sobre mallats

Els tipus de distorsió coneguts a causa dels tipus de lents que s'utilitzen al dispositiu, són la distorsió barril i la distorsió coixí. Per evitar dita distorsió, el dispositiu realitza un mapa mallat de punts de calibrat superposar a la imatge per a cadascun dels seus sensors.

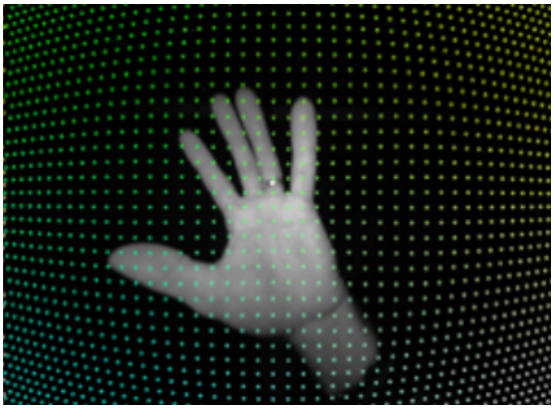


Figura 28 Imatge del mallat de punts de detecció

Cada buffer de dades de la imatge que és enviada directament al driver del dispositiu té associat un altre buffer que conté les seves dades de distorsió. Aquestes dades són una reixeta de 64 x 64 punts amb dos valor de 32 bits cadascun. El valor d'un punt del mallat defineix la lluminositat d'un píxel a la imatge distorsionada i es poden obtenir les dades de lluminositat de cada píxel de la imatge. Els valor de la quadricula que es troben fora del rang [0...1], no corresponen a un valor de dades de la imatge i no s'utilitzen a la imatge, és a dir, són ignorats.

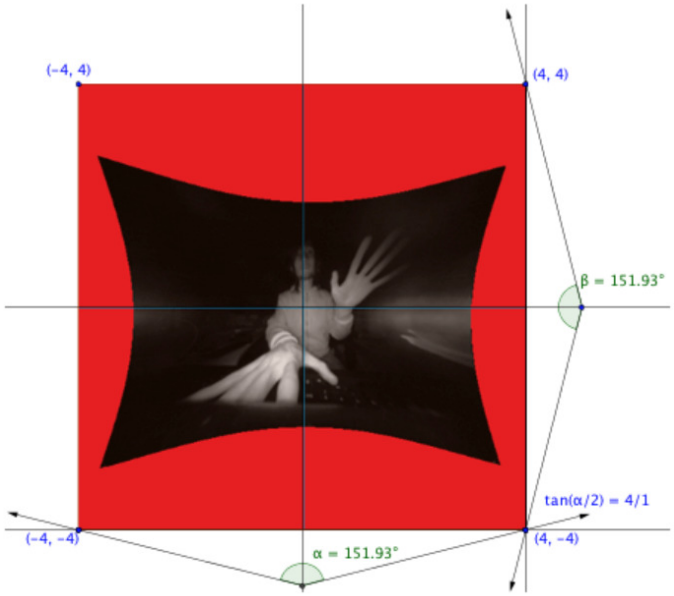


Figura 29 Exemple de distorsió corregida

Per exemple, aquesta imatge, mostra una reconstrucció de les dades d'una imatge amb distorsió corregida. La construcció final de la imatge es duu a terme mitjançant càlculs de les pistes horitzontals i verticals representades per cada píxel i es por trobar la lluminositat

real de les dades de la imatge, utilitzant el mapa de calibratge. Les parts vermelles de la imatge són les àrees a l'interior de la escena per la que no es presenta cap valor de lluminositat. Aquestes imatges, un cop generades, són enviades al *driver*, que és l'encarregat de detectar la presencia de mans a l'escena i materialitzar els efectes de tridimensionalitat o profunditat.

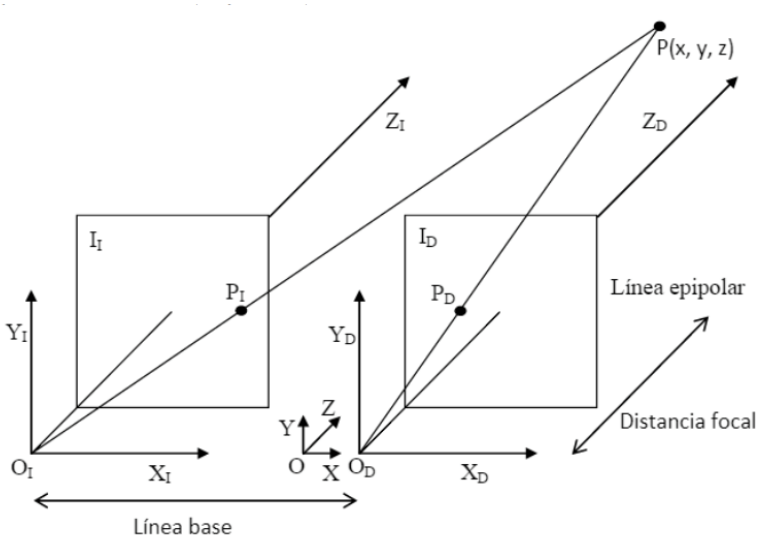


Figura 30 Esquema de detecció d'un punt per part de les dues càmeres

Posteriorment, un cop identificades les mans, es determina la posició al sistema de referència. Aquest procediment es realitza gràcies a la visió binocular del dispositiu Leap Motion i a través de tècniques de visió estereoscòpica, igual que la tecnologia de les ulleres HTC Vive vistes anteriorment. La visió de les dues càmeres, que estan situades de manera física sobre el mateix pla, tenen la mateixa distància focal, gràcies a aquesta propietat, és possible el càlcul de la disparitat per a cada parella de punts Pi i Pd, utilitzant la formula general d = XI-XD.

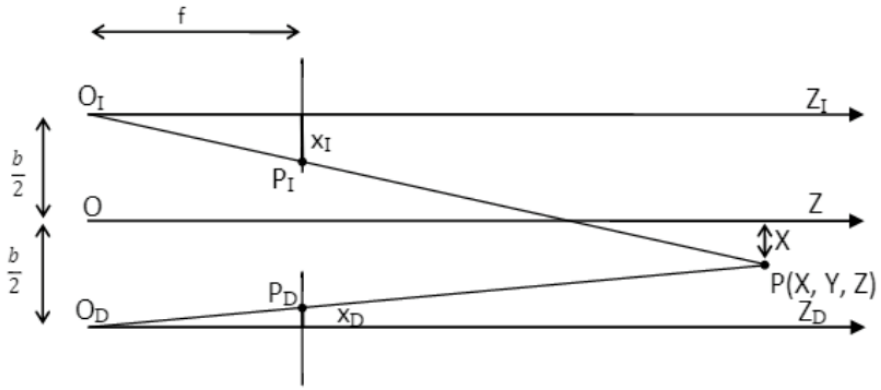


Figura 31 Vista lateral de la detecció d'un punt

Considerant idealment que la distància focal *f* de totes dues càmeres és equivalent i coneguda la distància *b* de separació entre totes dues, podem desenvolupar un sistema d'equacions a través del qual és possible l'obtenció de la posició física de l'objecte respecte al dispositiu. Aquest sistema és el següent:

$$\left.\begin{aligned} O_I &: \frac{\frac{b}{2}+X}{Z} = \frac{x_I}{f} \\ O_D &: -\frac{\frac{b}{2}-X}{Z} = \frac{x_D}{f} \end{aligned}\right\} \Rightarrow \left.\begin{aligned} x_I &= \frac{f}{Z}\left(X+\frac{b}{2}\right) \\ x_D &= \frac{f}{Z}\left(X-\frac{b}{2}\right) \end{aligned}\right\} \Rightarrow d = x_I - x_D = \frac{fb}{Z} \Rightarrow Z = \frac{fb}{d}$$

Figura 32 Equacions del càlcul de la posició física

4.2.3. Especificacions tècniques de l'aparell

Les característiques necessàries per un ús òptim del dispositiu Leap Motion són:

Connexió	USB 2.0
Hardware	Dues càmeres mónocromàtiques 300fps amb lents infrarojos.
Distància aproximada d'actuació	Semiesfera d'un metre de radi
Precisió màxima	0.01 mm
Requisits d'entorn	Llum (diürna halògena o incandescents).

Taula 2 Requisits per l'ús de Leap Motion

4.2.3.1. Software necessari

Pel que fa als requisits de sofware, és necessària d'instal·lació de la versió oficial del programa Leap Motion, per a la detecció i calibrat del dispositiu a l'ordinador. Actualment es treballa amb la versió del programa Leap Motion Orion, especialitzada en el funcionament del Leap Motion combinat amb ulleres VR. Per a la configuració del dispositiu a l'ordenador, s'utilitza un panell de control que gestiona totes les opcions que posseeix dit dispositiu.

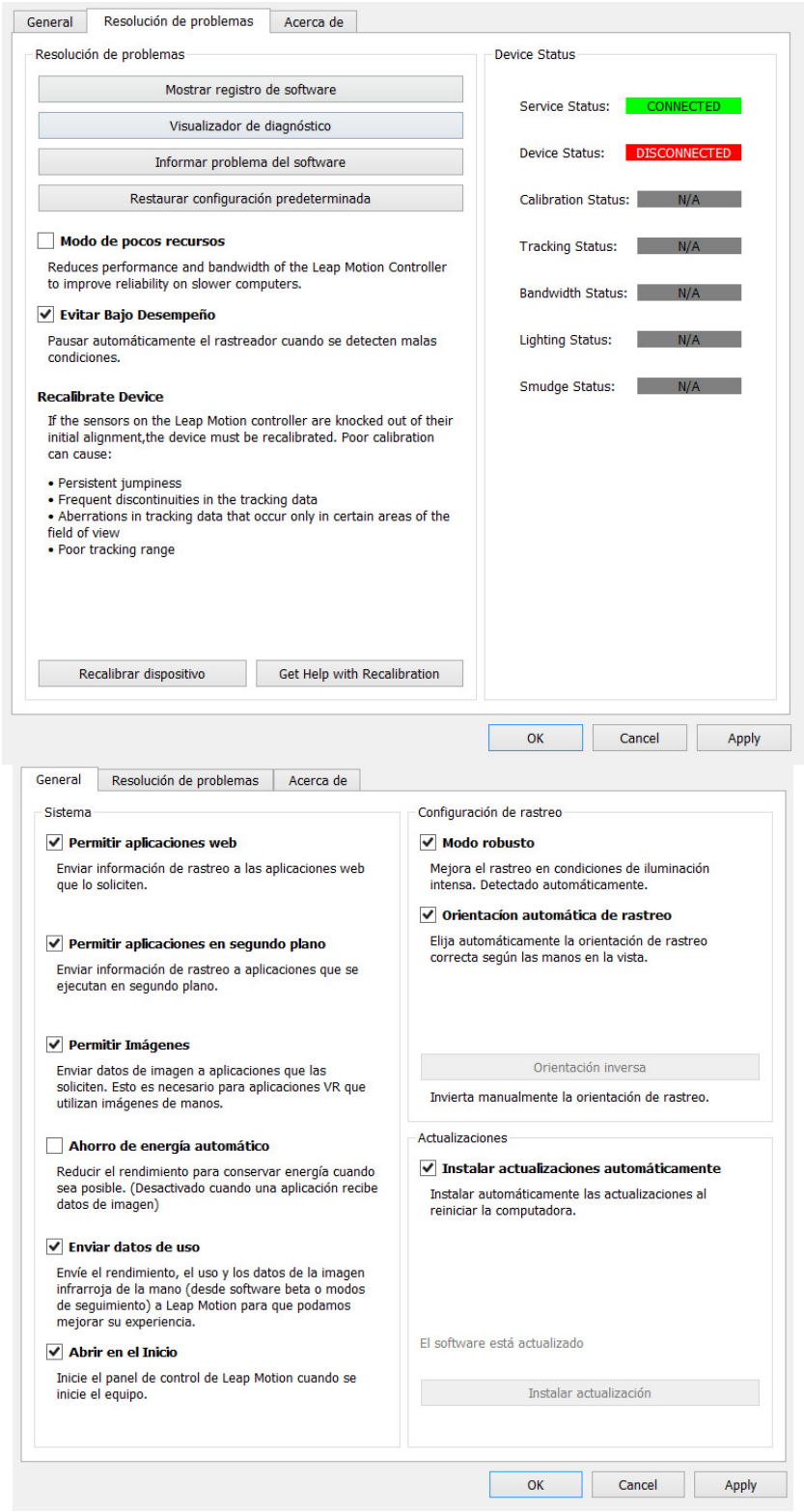


Figura 33 Menu del software de Leap Motion

5. Motor de videojocs

Un motor de videojocs es refereix a una sèrie de rutines de programació que permeten el disseny, creació i la representació de videojocs, existents tant en consoles com en sistemes operatius.

La funció principal d'un motor de videojocs és fer de motor de renderitzat, per a gràfic 2D o 3D, motor físic, administrador de memòria i escenari gràfic. El procés de creació d'un videojoc pot variar de forma notable per la reutilització d'un motor gràfic o la seva adaptació. És una pràctica molt comú a dia d'avui, on es solen reutilitzar gran nombre de motors, inclòs, existeixen empreses com Epic, Valve i Crytek que han posat els seus motors a disposició pública per a qualsevol usuari que els vulgui utilitzar.

5.1. Aspectes bàsics d'un motor

Els aspectes bàsic d'un motor de videojocs es poden explicar com a una sèrie de conceptes necessaris per entendre el seu funcionament:

1. *Els assets:* El concepte d'asset s'entén com element que serà introduït a la escena del videojoc. Poden ser models 3D, textures, scripts, animacions, materials, sons entre d'altres. Cada motor té una gamma específica d'acceptació d'assets als seus videojocs.

2. *API:* Una API, o *Interfície de Programació d'Aplicacions* és un sistema de rutines, protocols i eines per al desenvolupament de programes per a videojocs. Aquesta interfície es sol materialitzar en una llibreria d'*scripts* que facilita la creació de programes més complexos i presenta totes les funcionalitats bàsiques del dispositiu. Com idea general, és la connexió entre la part de hardware i la part de software.

3. *Renderització:* És el procés en el que l'ordinador mostra l'aspecte visual del videojoc. És l'encarregada de mostrar a l'usuari l'aspecte gràfic que és capaç d'assolir el joc, visible en textures i materials.

Aquest són els aspectes més enfocats a la creació dels models del treball, en el qual ens centrem en la interacció entre els programes creats i el dispositiu utilitzat. És necessari entendre com s'interactua amb el model d'assets i saber jugar amb l'API (interfície de programació d'aplicacions), com es veurà als pròxims punts.

5.2. Motor Unity 3D

Unity és el motor de videojocs utilitzat per la creació d'aplicacions de la part aplicada del treball. Va ser creat per l'empresa *Unity Technologies* i és una aplicació d'ordinador oberta al públic, que pretén facilitat la creació de videojocs amb funcionalitats i mètodes de creació senzills i intuïtius. Entre les comoditats que aporta a l'usuari, trobem un gran nombre de tutorials oficials que mostren el més bàsic que es por realitzar amb el motor i el funcionament jeràrquic que segueix, i un gran nombre de packs d'assets que faciliten la tasca de creació, alguns d'ells gratuïts, a una plataforma anomenada Asset Store que funciona com cercador.

El motor no és el més potent que es pot trobar avui dia ni el més competitiu, però tot i així, moltes empreses s'estan centrant en la creació de videojocs en aquesta plataforma per les comoditats que aporta i l'estalvi de temps i inversió. També té l'avantatja de permetre la creació de videojocs per a qualsevol plataforma, consola o ordinador segons la configuració que aporti el creador.

5.2.1. Interfície del motor

El disseny de l'editor de Unity permet realitzar totes les accions arrossegant fitxers i objectes i modificant jerarquies. La interfície del programa és personalitzable, però sempre consta d'unes parts bàsiques i necessàries per a la edició en la creació de videojocs. L'aspecte predeterminat de Unity consta de les següents parts:

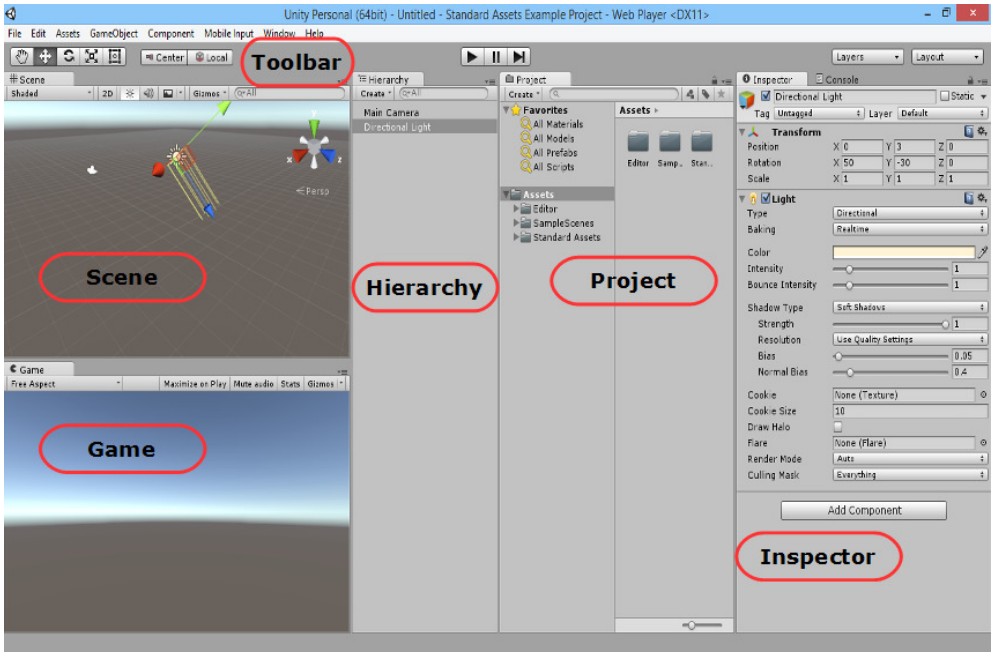


Figura 34 Imatge de la interfície de Unity

- *Project Window*: ventana que permet l'accés als packs d'assets, arxius importats, objectes, materials, textures, àudios, scripts, d'entre altres recursos que poden ser utilitzats a l'aplicació. Es poden guardar, importar arxius des de la memòria local de l'ordinador o importar-ne objectes des de l'Asset Store.

- *Hierarchy Window:* Finestra on es mostra la jerarquia dels objectes a l'escena. Cada element ha de tenir una posició a l'arbre jeràrquic de l'aplicació, fet que després permet jugar amb dita jerarquia a la creació de scripts pel joc i amb les interaccions dins del joc.

- *Inspector Window:* Permet visualitzar totes les propietats d'un objecte seleccionat a l'escena o la *Project Window* i permet la seva edició.

- *Scene Window:* Permet interaccionar amb l'escena de manera gràfica, tant si l'aplicació es troba iniciada com si no, i modificar aspectes de localització de qualsevol objecte a l'escena.

- *Game Window:* Permet visualitzar l'escena des del punt de vista del jugador un cop iniciat el joc. No permet editar cap aspecte de l'escena, només permet testejar el que ja ha estat creat.

- *Toolbar:* Permet l'accés a les modificacions bàsiques del projecte. D'esquerra a dreta, trobem les eines bàsiques per la interacció amb la *scene view,* els botons d'inici, pausa i stop del videojocs, els botons d'accés directe a Unity Cloud, un menú d'accionament sobre les capes del projecte i, finalment, un menú del layout de l'editor.

Hi ha una altra accessibilitat destacable de la interfície que no apareix descrita a la figura i és d'utilització comú en la creació d'aplicacions independents, es tracta de:

- *Asset Store Window:* Finestra que dóna accés a l'usuari al cercador virtual d'assets. En aquesta finestra es solen trobar, quan es vol començar un projecte, els recursos bàsics necessaris per qualsevol tipus de projecte i que agilitzen la seva creació. En el nostre cas, podem trobar packs d'assets per a cascs de realitat virtual.

5.2.2. Objectes i components

Al creador de Unity, la paraula objecte o *GameObject* no fa referència únicament a objectes visuals en 3D o 2D, fa referència a qualsevol element, visual, sonor o element de referència, que és capaç d'interactuar amb l'escena i té un paper fonamental al desenvolupament de l'acció de l'aplicació. Tot objecte es defineix a l'API de Unity com a una classe diferent a l'hora de ser programat, però sempre con un GameObject, fet que facilita la creació dels Scripts fent més senzill la definició i la crida dels objectes del projecte. Els objectes bàsic a la creació d'aplicacions en Unity 3D, es podrien llistar de la següent manera:

- Components de rendering:

- Càmera: Dispositiu encarregat de mostrar la visió de l'aplicació des d'un punt concret de l'espai i poder materialitzar-la a la finestra de joc. Normalment, es sol crear una càmera de jugador que s'encarrega de seguir els moviments d'aquest o una càmera que enfoqui el punt on es centra l'acció del joc.

- Skybox: Són el conjunt d'imatges que s'encarreguen d'envoltar l'escena i presenten una visió d'horitzó i immensitat superior al que realment posseeix el joc, que té unes dimensions limitades. Es composa de 6 imatges que generen un cub, el qual envolta completament l'escena del joc.

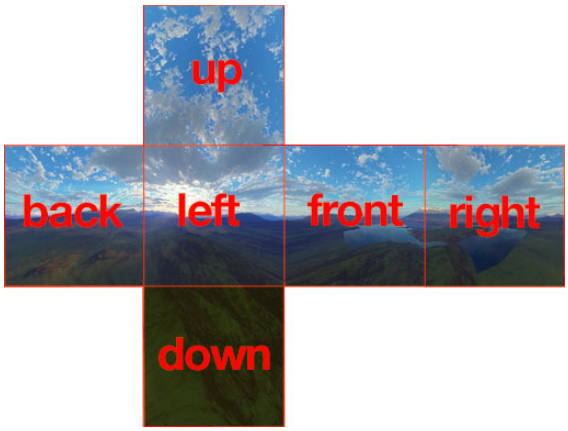


Figura 35 Exemple de parts d'un SkyBox

- Textures: La textura d'un material és l'encarregada de crear realisme en un objecte a través de generar en el seu disseny superficial sensacions de profunditat, realisme en la reflexió de la llum en aquest. És un tipus *d'script* capaç de calcular matemàticament el color que ha de tenir cada píxel en funció de l'angle de visió i la il·luminació que rep.

- Components de posicionament:

- Transform: Encarregat de donar valor a la posició en la espai, grandària, i rotació d'un objecte.

Les propietats de posicionament d'un objecte poden modificar-se ja sigui des de l'*Scene* o des de l'*Inspector* depenent de la precisió en les dimensions que es vulgui assolir. Modificant-los directament des de l'*Scene* suposa modificar-los amb el moviment del cursor i a ull nu, en canvi, amb l'*Inspector* podem donar un valor numèric a les dimensions i la orientació dels objectes a l'espai, a més de poder afegir-hi textures, colors, materials, scripts, entre d'altres.

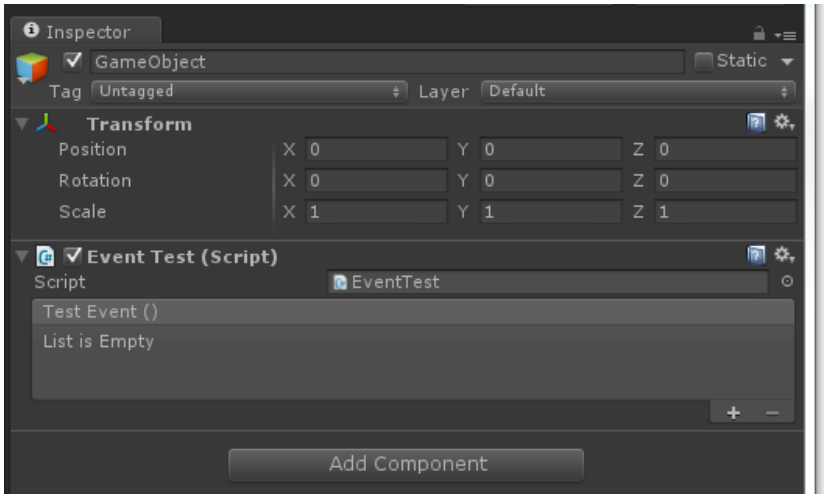


Figura 36 Imatge de la finestra Transform

Amb les propietats *Position*, *Rotation* i *Scale* de la finesta *Inspector* podem ser capaços de modificar la posició, rotació, i dimensions físiques d'un objecte respectivament. El valor *d'Scale* va referenciat a la grandària original del model de l'objecte, i el numero és la proporcionalitat de les dimensions amb la referència de l'original en la direcció de cada eix. La posició depenent de la jerarquia de l'objecte, por fer referència a l'origen de l'objecte al qual es troba jerarquitzat o bé a l'origen de coordenades absolut del sistema. La rotació es mesura en graus de rotació de cada eix de coordenades relatiu a l'objecte en qüestió.

- Components físiques:

- Rigidbody: Permet definir les normes físiques a les que es troba sotmès un objecte dins de l'escena. Aquestes accions en poden definir a *scripts*, o de manera predeterminada, com l'acció de la gravetat sobre el cos, la massa d'aquest, entre d'altres.

- Colliders: És l'encarregat de definir de quina manera l'objecte interactuarà físicament amb l'acció d'altres objectes de l'escena. Es defineix com un perímetre d'actuació que defineix la zona a la qual l'objecte interactuarà d'alguna manera amb la resta. Les possibilitats a la definició d'aquest perímetre geomètric, es troba limitat a formes de cubs, esferes i capsules.

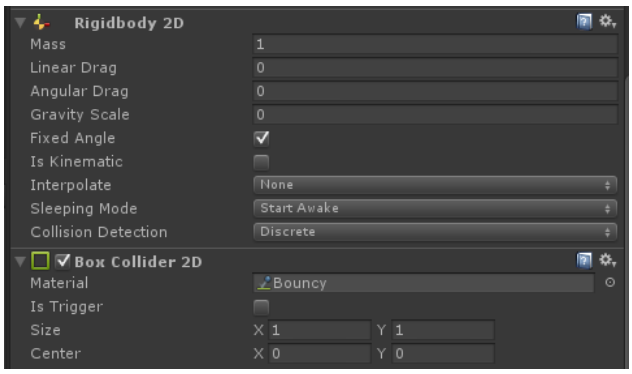


Figura 37 Finestra de l'opció collider 2D

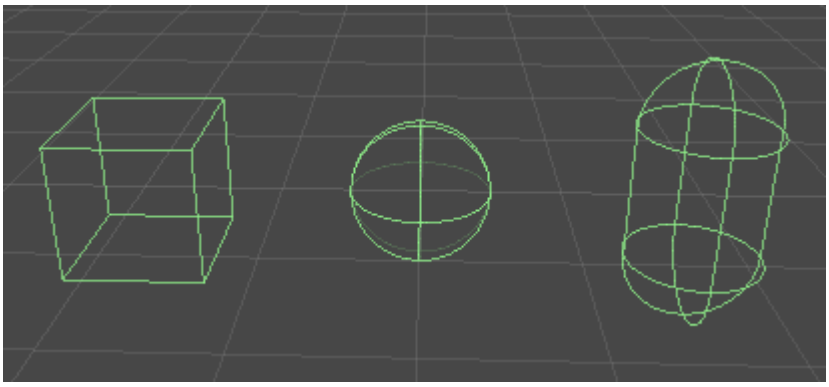


Figura 38 Formes estàndard de colliders

■ Components d'interfície d'usuari:

- Canvas: objecte utilitzar per definir un element que afectarà directament a una porció d'espai de la interfície de l'usuari. És a dir, es defineixen dits elements en un determinada posició a la referència de la interfície del jugador ocupant un determinat espai. Aquest elements poden ser:

    - Text: Un text ocupa espai de la interfície.
    - Imatge: Una Imatge ocupa l'espai de la interfície.
    - Botó: Un botó ocupa l'espai a la interfície.

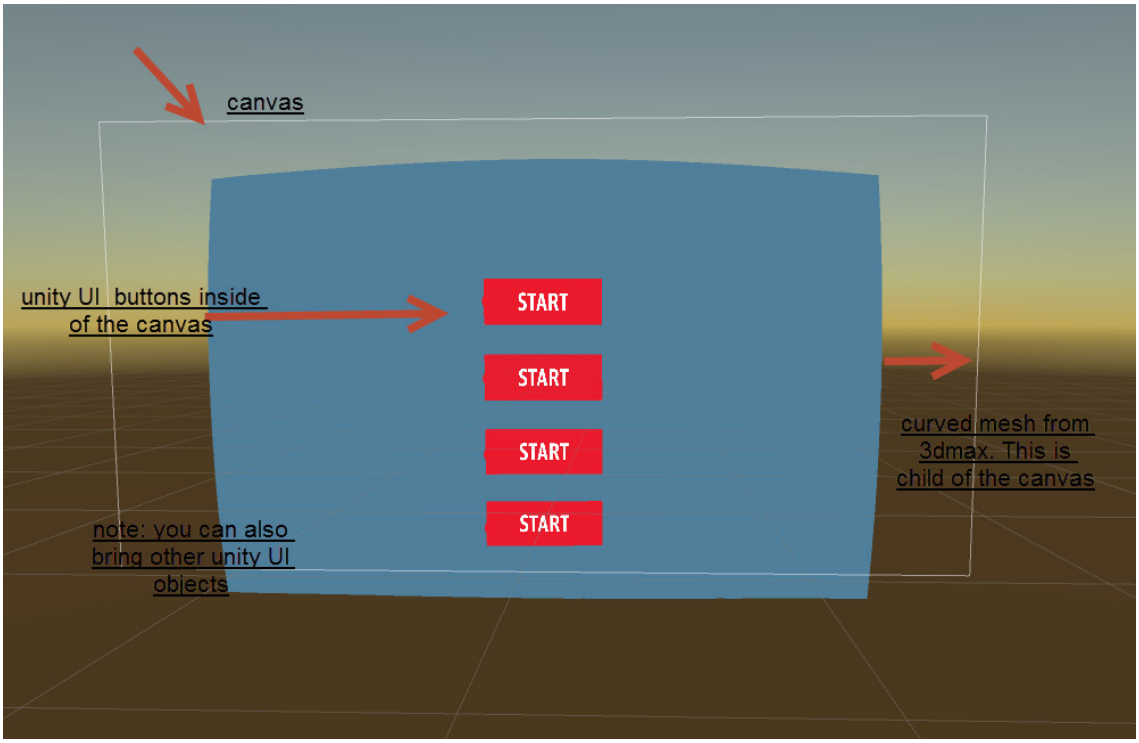


Figura 39 Exemple i parts d'un Canvas

- Components de programació:

  Scripts: Els scripts permeten al creador anar més enllà de les opcions que facilita el motor a la interfície, i poder crear-ne característiques específiques i pròpies per a cada videojoc. Representen un medi per al jugador d'activar i desactivar uns events determinats definint-ne unes especificacions per a cadascun d'aquests.

  Aquest, es creen amb l'ajuda de les diferents API's, tant de la específica de Unity, com de la dels diferents hardwares (en cas de ser necessari per la creació del joc). Els llenguatges de programació més comuns per la creació d'aquest programes són *C#* i *Javascript*.

  Aquests fitxers per defecte, s'obren amb MónoDevelop, tot i que depenent del sistema i les preferències del sistema poden obrir-se amb l'editor de text Visual Studio. Inicialment, un *script* té una forma semblant a la següent:

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

Figura 40 Estructura bàsica d'un script

Els *scripts* es connecten internament amb el motor incorporant la classe MónoBehaviour, que forma part de l'API. Les dues funcions principals dels programes són la funció *Start()* i la *Update()*. La funció *Start()* s'executa abans de que el videojoc s'executi i s'encarrega de realitzar la inicialització de l'objecte, definint els events amb els quals inicia la seva activitat i la seva configuració en un inici. En canvi, la funció *Update()* s'encarrega de l'actualització i comprovació de l'estat de l'objecte a cada frame del desenvolupament del joc, i s'encarrega de decidir com actuarà l'objecte al següent.

- *Prefabs:* Els *prefabs*, com s'indica al seu nom, són objectes prefabricats. La seva utilitat és la de permetre una recursivitat senzilla sempre que sigui necessària la creació de diversos objectes idèntics, guardant un model de referència que podrà ser utilitzat posteriorment a qualsevol projecte on s'importi el *prefab*. El procediment necessari per la creació d'aquest objectes és senzill, només cal que s'arrossegui l'objecte des de la Jerarquia del projecte fins a la finestra de Project. Normalment, els *prefabs* son objectes fonamentals pel funcionament de dispositius amb llibertat pel desenvolupament d'aplicacions, solen oferir prefabs que faciliten la feina de creació de scripts basats en l'ús de les respectives API's, com és el cas del casc HTC Vive i el dispositiu Leap Motion.

6. Programació dels dispositius i adjunció a un projecte de Unity 3D

Per a cadascun dels dispositius del treball, existeix un procediment per l'adjunció al motor de Unity3D, però, gracies als avanços de la nova versió de Leap Motion Orion, el procediment és un procediment comú per tots dos dispositius gràcies a la modificació recent l'API de Leap Motion. D'aquesta manera, els requisits de descarrega de paquets d'assets es limita als *Developer Packs* de la web oficial de Leap Motion.

6.1. Programació del Dispositiu Leap Motion

La programació del dispositiu Leap Motion es basa en l'ús de les funcionalitats de l'API adaptades per la creació *de scripts* al dispositiu Leap Motion. Aquesta API, utilitza com a orientació principal i referència un eix de coordinades situat al centre de la superfície del dispositiu.

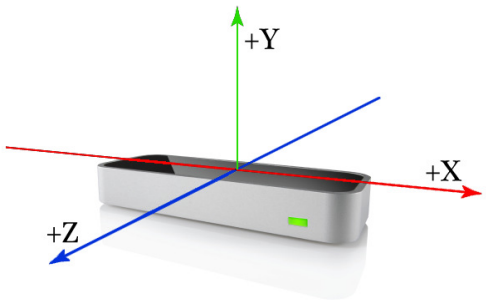


Figura 41 Sistema de referència del Leap Motion

Aquest, utilitza com a unitats físiques de quantificació els mil·límetres (distància), microsegons (temps), els mil·límetres/segon (velocitat) i els radians (angles). Pel que fa a la concepció de referència relativa i posició de les mans, podem obtenir aquesta informació i més dades gràcies a la classe *Hand*, la qual té associada una llista dels dits.

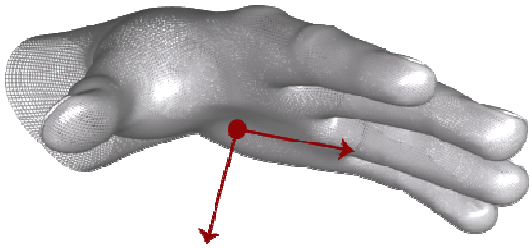


Figura 42 Sistema de referència de les mans

Els vectors de la classe *Hand* que observem a la figura s'anomenen *PalmNormal* i *Direction*. Sabent l'orientació i la direcció d'aquests vectors, podem programar en funció de la posició de la palma de la mà. En el cas dels dits, que és una classe que deriva de la classe mà, anomenada *Finger*, també podem utilitzar com a referència vectors de posició a l'hora de programar. Aquest vectors reben el nom de *TipPosition* i *Direction*, que proporcionen la posició de cada extrem de cada dit, i la direcció a la que apunten respectivament.

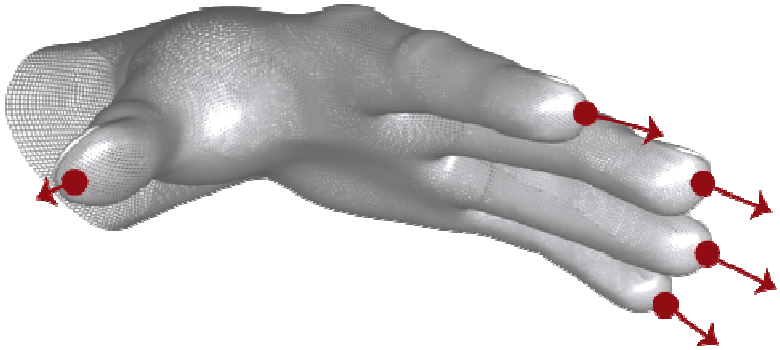


Figura 43 Vectors de les puntes dels dits

Cada objecte *Finger*, proporciona un *Bone* object descrivint la posició i orientació de cada os de la mà. Cada *Finger* conté 4 ossos.

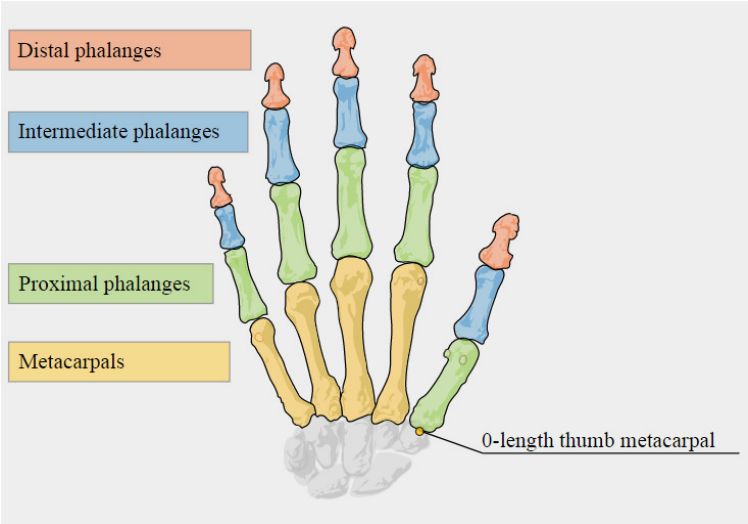


Figura 44 Parts de cada dit

A la figura s'observen els noms que les classes subordinades per cada part del dit, d'aquestes es pot obtenir informació de posició i direcció a cada frame. També és possible combinant la idea de classes de l'API de Leap Motion, i l'API de Unity3D, programar les mans en funció dels moments de col·lisió de cada objecte amb un altre, específic o no.

Depenent de la part, classe, de Hand(), podem obtenir-ne una informació o una altre. La informació per a cada part és:

Objecte	Informació disponible
Pointable Tool	- Posició/Velocitat de la punta. - Direcció de l'eix. - Base ortonormal. - Amplada/Longitud.
Hand	- Posició/Velocitat de la palma. - Direcció/orientació dels eixos. - Base ortonormal.
Finger	- Posició/Velocitat de la punta. - Direcció/Orientació dels eixos - Base ortonormal. - Amplada i longitud.
Bone	- Posició articulació - Base ortonormal - Amplada/Longitud del model
Arm	- Posició del canell i el colze. - Direcció de l'eix. - Base ortonormal - Amplada/Longitud del model

Taula 3 Informació que es por obtenir de cada part de la mà

6.1.1. Preparar un projecte en Unity 3D

El primer pas per la creació de l'aplicació, és incloure la virtualització de les mans que ens ofereix el dispositiu Leap Motion a la nostra escena, així com permetre a les ulleres VR fer el paper de càmera del nostre joc. Inicialment és necessària la creació d'un projecte de Unity3D, de la forma més usual:

1. Obrir l'editor de Unity.
2. Seleccionar File > New Project.
3. Escollir un nom i un *path* on guardar el projecte
4. Fer click a Create Project.

Després, importem el Leap Motion asset package al nostre projecte:

1- Descarreguem de la web de Leap Motion, el Packet d'assets..

2- Seleccionem a Unity Assets > Import Package > Custom Package.

3- Seleccionem el packet d'assets i fem click a obrir.

Després d'haver importat totes les eines necessàries, es pot procedir a la adició de les mans a l'escena. Primerament, es situa el prefab *HandController* sota el punt on volem que apareguin les mans a l'escena. Les mans a la escena estan posicionades a la mateixa posició relativa que les mans reals per sobre del dispositiu físic. Aquest prefab, inclou tres components:

1- **LeapServiceProvider**: Classe encarregada de comunicar el servei de Leap a la plataforma i proveir de la virtualització de les mans a Unity. És una implementació que defineix la interfase bàsica esperada pels mòduls utilitzats.

2- **LeapHandController**: Pren les dades proporcionades des del *LeapServiceProvider* i crea o actualitza els models manuals basats en les mans. Requereix un HandPool al mateix objecte.

3- **HandPool**: Implementa un grup d'objectes de dades de mà. El *LeapHandController* l'utilitza per reutilitzar els objectes de dades a mesura que entren i surten les mans del marc de detecció. Per tal que el *HandPool* funcioni, és necessària l'addició de grups a la gamma Pool, un per cada parell de models gràfics de *IHandModels* o *HandModels* que es vulguin utilitzar. Només cal augmentar la mida de la finestra del model, i arrogar els models esquerre i dret.

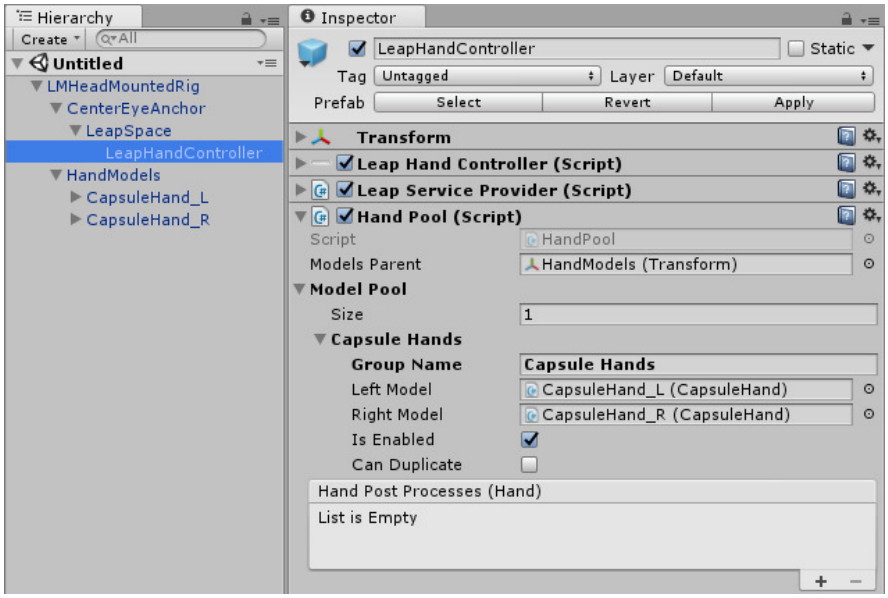


Figura 45 Esquema de la jerarquia de LMHeadMountedRig

En el cas de projectes de realitat virtual, combinació de casc i Leap Motion, s'utilitza un prefab diferent, es tracta del *LMHeadMountedRig* prefab. *LMHeadMountedRig* representa l'objecte arrel. Aquest objecte conté *l'script VRHeightOffSet* opcional de manera predeterminada, que ofereix una manera senzilla d'ajustar l'alçada inicial de la vostra aplicació depenent de si s'està executant en un dispositiu Oculus Rift, Vive o d'altres. El *CenterEyeAnchor* és la càmera principal, la seva posició i rotacions locals estan controlades directament per la integració de VR de Unity; no es podrà moure manualment. Aquest *GameObject* també conté l'objecte *EnableDepthBuffer* i el component *LeapVRCameraControl*. *childed* al *CenterEyeAnchor* és l'objecte LeapSpace, que conté el component *LeapVRTemporalWarping*. Aquest component manipula la posició i rotació efectiva del *LeapHandController* en funció del moviment dels cascs VR per tenir en compte les discrepàncies temporals entre les actualitzacions de seguiment del casc i les actualitzacions de seguiment del controlador. El *LeapHandController* es troba sota l'objecte *LeapSpace*, internament les dades de la mà de Leap s'entenen per referir-se a l'espai local del controlador, de manera que la posició d'aquest objecte determina en última instància el lloc on es generen i es troben les mans. *HandModels* és germà de *CenterEyeAnchor*, i és l'objecte pare de tots els models de mans i físiques de mans. La visibilitat de les mans és activada i desactivada per la programació del *LeapHandController* en detectar la presencia de mans sobre el dispositiu. Qualsevol altre objecte centrar en el reproductor, com l'objecte Hands Attachment, pot residir com a fills de l'objecte càmera.

Els prototips de mans creats per evitar als creadors la necessitat de la creació dels models gràfics, són els següents:

:

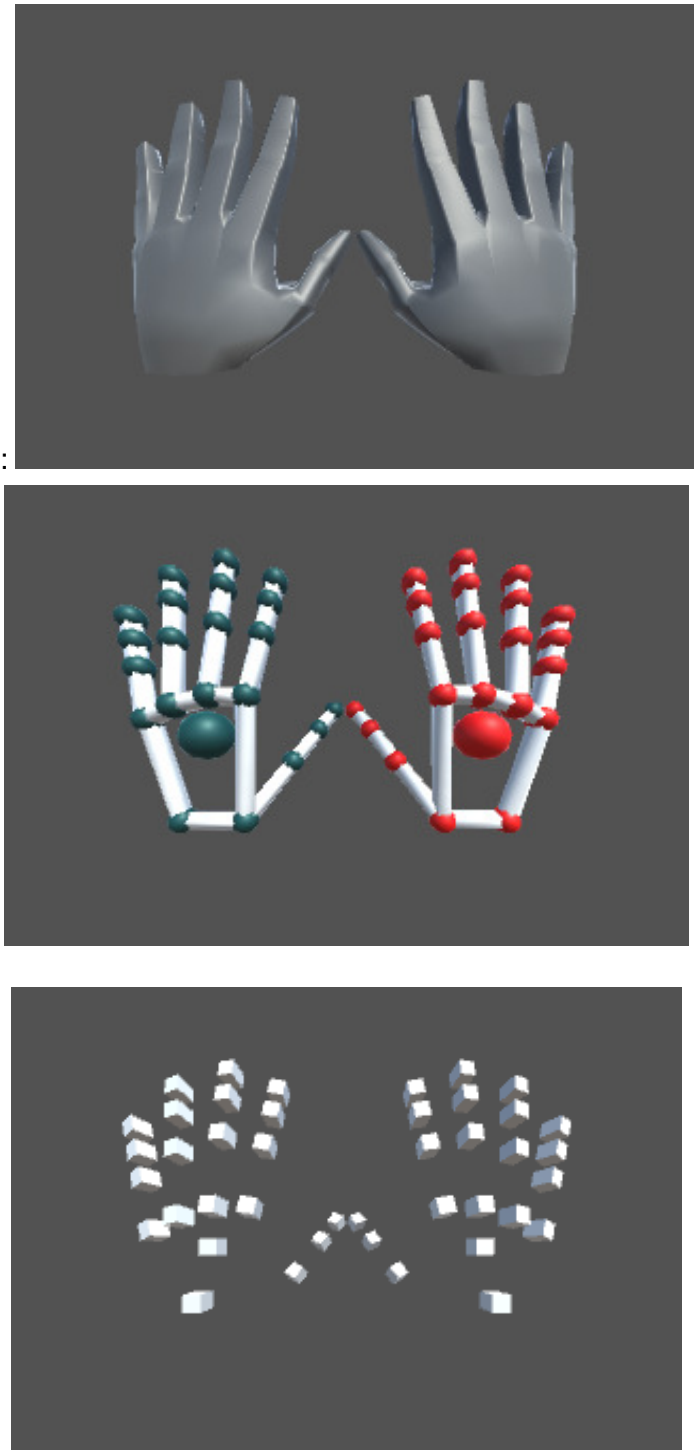


Figura 46 Models de mans estàndard: Rigged, Capsule, i Non-handmodels

6.1.2. Interaction Engine

El Interaction Engine és un dels mòduls públics que ofereixen els creadors del dispositiu Leap Motion per la creació autònoma d'aplicacions amb el dispositiu. Permet al creador treballar amb aplicacions en VR o escriptori creant interaccions físiques o psudo-físiques amb objectes de l'escena. Les condicions necessàries per la implementació d'aquest mòdul en un videojoc són:

1- Els objectes amb els quals es por interactuar són objectes que contenen *l'script* Interaction Behavior del paquet de developers, han de ser *Rigidbody's* i al menys tenir un *Collider*.

2- El Interaction Manager rep la informació del contacte directament del FixedUpdate de Unity i controla tota lògica que fa possible la interacció, inclosa l'actualització de dades de les mans i el controlador. És necessari incloure'l a l'escena perquè les interaccions funcionin. Es recomanable situar-lo com fill de l'objecte que conté la càmera.

3- La física de les mans, ha de ser de caràcter *Interaction Hand* per tal que la interacció funcioni. El model gràfic no afecta a la interacció.

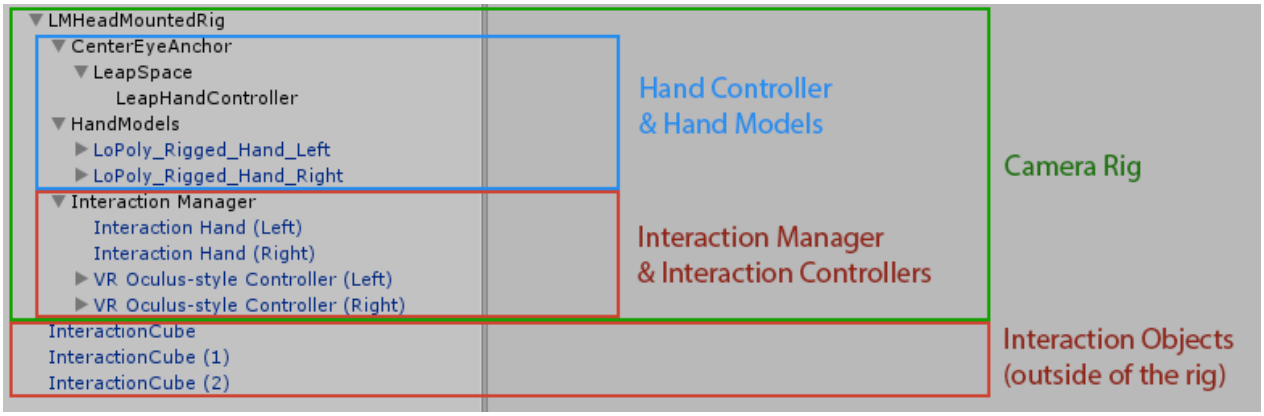


Figura 47 Esquema de la jerarquia apropiada pel funcionament de InteractionEngine

En afegir el component *InteractionBehaviour* a un objecte, un parell de coses passen automàticament:

- Si prèviament no tenia un, guanyarà un nou component de *Rigidbody* amb gravetat habilitada, transformant-lo en un objecte físic que es regeix pel motor de Unity.

- Assumint que l'objecte està associat a un o més controladors d'interacció, serà possible amb les mans interactuar i agafar l'objecte.

6.1.3. Graphic Render

El mòdul *Graphic Render* es centra en ser un sistema que permeti tractar una col·lecció d'objectes gràfics com a grup, en lloc de com objectes individuals. Això permet optimitzar el projecte i facilitar l'accessibilitat als objectes.

Resumint la base de la seva implementació, el mètode de representació és un codi que porta una llista de gràfics i decideix la millor manera d'obtenir els objectes representats com a píxels a la pantalla. Un mètode de representació pot prendre enfocament diversos, donat que existeixen diferents mètodes de representació és que hi ha diferents estratègies d'optimització que es poden utilitzar per representar una mateixa idea. Cadascuna com és lògic posseeix diferents avantatges i inconvenients en la seva implementació. El renderitzador de Leap ofereix tres mètodes d'implementació diferents:

- The Baked Render: Aquest mètode només és capaç de dibuixar gràfics de malla. L'estratègia consisteix en combinar tots els gràfics en una única malla per tal de poder trucar-los amb una única ordre.

- The Dynamic Render: Aquest mètode, com l'anterior, només pot dibuixar gràfics de malla. L'estratègia és fer una manipulació de malla personalitzada que garanteix que els gràfics es puguin crear amb el mateix material i es pugui realitzar un *mesh* automàtic utilitzant la unitat dinàmica de Unity.

- The Dynamic Text Render: Aquest mètode només pot renderitzar gràfics de text. Utilitzant la unitat dinàmica de Unity, crea *meshes* que creen l'aspecte gràfic del text a mostrar.

S'ha de tenir en compte que la potència del mètode és poder mirar tots els gràfics alhora i optimitzar de manera senzilla la informació.

6.1.4. Hands Module

Aquest mòdul permet a qualsevol persona que utilitzi el Leap Motion com a creador incloure models propis de mans virtuals a l'aplicació. Pretén automatitzar i simplificar els models 3D dels seus Core Assets, per agilitzar el procés de creació de models.

Aquest mòdul separa l'estructura de la mà per ossos, de manera que l'usuari por definir el model que vulgui a cada os i a cada articulació, d'aquesta manera es torna un procés més intuïtiu i senzill, on només cal arrossegar el model que volem fins a la casella indicada. El prefab més àgil, és el *LeapHandsAutorig,* on per defecte ja trobem una configuració estàndard del models de les mans i totes les caselles creades, l'altra alternativa es l'ús de *RiggedHand*, on les transformacions de la mà es defineixen afegint elements manualment, però ofereix major versatilitat si es vol programar utilitzant el tipus de model de la mà.

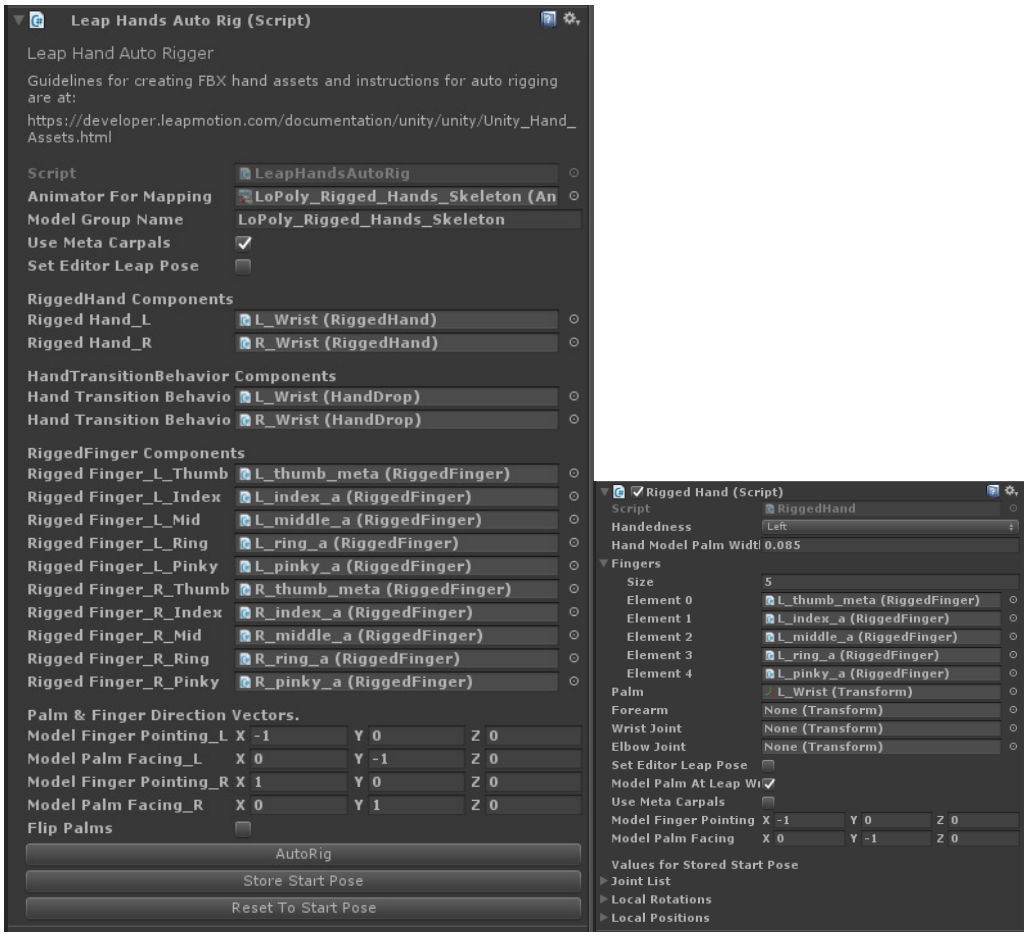


Figura 48 Configuració d'un mòdul de mans a Unity

7. Creació de les aplicacions

Per aplicar el conjunt de coneixements assolits al llarg de l'estudi de la realitat virtual i dels dispositius Leap Motion i *HTC Vive*, el treball pretén crear aplicacions que mostrin el potencial de la combinació dels dispositius, les maneres més òptimes d'ús i exemples aplicats dels mètodes de programació amb l'ús de les funcions que facilita l'API i els diversos packs d'assets d'ús públic.

En totes les aplicacions, s'utilitza la versió més recent de Leap Motion, la versió Orion, una versió encara en probes. S'han creat tres aplicacions diferents, amb jugabilitats i programacions molt diferents per tal de cobrir al màxim les possibilitats de les que es disposa amb l'API del Leap Motion. La primera aplicació utilitza el *grasping* amb les mans, novetat de l'última versió de Leap Motion, la segona es centra en el control d'un personatge en un joc bàsic utilitzant gestos, i l'última aplicació, combina la visualització a través del casc VR amb el control total del joc a través de gestos.

7.1. Grasping and Throwing Game

El primer joc és un joc de *Grasping and Throwing*, o el que es el mateix, un joc que consisteix en utilitzar les mans virtuals per agafar un objecte i llençar-lo contra un objectiu. En aquest cas, el joc ha estat creat amb dos dissenys gràfics diferents per mateixa jugabilitat i opcions. La primera ambientada en el cel, rep el nom de *ThrowSky,* la segona ambientada a una zona coberta amb lava, *Vulcano*. Per l'aspecte final més polit de la versió Vulcano, es va decidir escollir-la com versió gràfica final

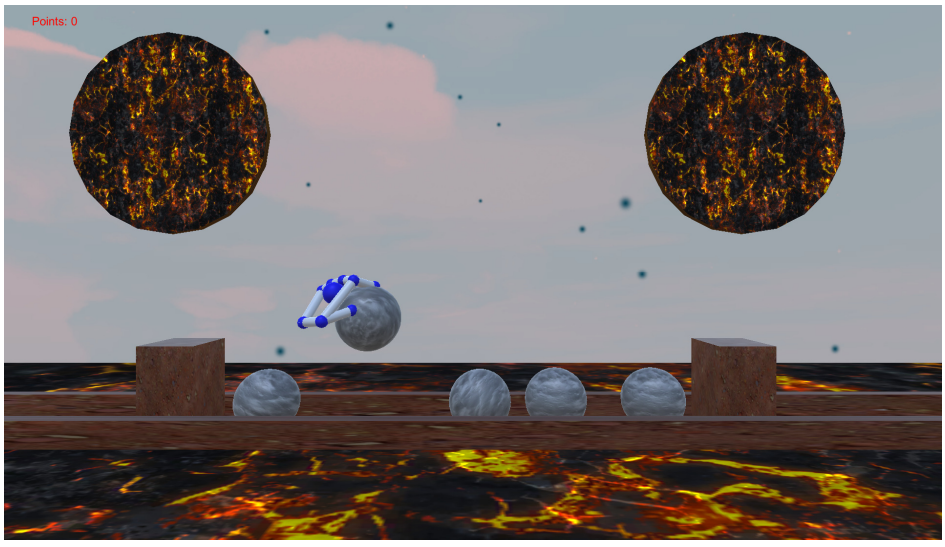


Figura 49 Pantalla del joc Vulcano

Es tracta d'un joc d'escriptori, no afegeix el casc de realitat virtual per jugar-lo com a condició indispensable. La idea bàsica del joc, és utilitzar les mans per llençar unes pilotes a unes dianes que es troben en moviment. Si la pilota impacta amb la diana, es sumarà un punt al marcador del jugador, si s'impacten dos d'una tacada, donat que tenen un punt d'intersecció, s'obtindran dos punts. Si s'arriba a la puntuació màxima que, donat que el jugador disposa de 5 pilotes són 10 punts, apareix un cartell que felicita al jugador amb el text "Perfect", en cas contrari, apareix un cartell de final del joc "Game Over". El joc disposa d'opcions de reinici i cancel·lació de l'escena.

Inicialment es va afegir el *LeapHandController*, que, com ja s'ha explicat anteriorment, s'encarrega de situar les mans a l'escena per sobre d'aquest element amb l'abast amb que es configuri la seva grandària amb l'opció *Transform* dels objectes de Unity. Es pot jugar a més de dues mans habilitant la seva opció "Can Duplicate" al Hand Pool. Seguidament, es va crear l'escenari, que consta d'una superfície inferior (lava), un carril on es troben situades les pilotes i les dianes. El primer que es va decidir programar pel funcionament del joc va ser el moviment de les dianes i la seva velocitat. *L'script* utilitzat és el següent:

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

public class Movement : MonoBehaviour
{
    public Vector3 positioni = new Vector3(0.0f, 0.0f, 0.0f);
    public Vector3 positionf = new Vector3(2.0f, 2.0f, 2.0f);

    Vector3 targetDestination;

    public float toleranceDist = 0.5f; //tolerance of positioning

    void Start()
    {
        transform.position = positioni; //initial position
        targetDestination = positionf; //final position
    }

    void Update()
    {
        transform.position = Vector3.MoveTowards(transform.position, targetDestination, 2 * Time.deltaTime);

        if (Vector3.Distance(transform.position, targetDestination) <= toleranceDist)
        {
            if (targetDestination == positioni)  //if target is initial pos
            {
                targetDestination = positionf; //final is the new target
            }
            else
            {
                targetDestination = positioni;
            }
        }
    }
}
```

Figura 50 Script del moviment de les dianes

Per les possibilitats i la versatilitat limitada de les mans a l'escena, era preferible que les dianes segueixin un recorregut senzill i que no requereixi girs en las mans i sigui més necessari un tir recte, per aquest motiu, es van programar amb cicles de moviment d'una banda a l'altra de l'escena, de forma horitzontal, amb intersecció entre les dues a la part central. Per entendre's bé, la posició final d'una és l'inicial de l'altre, i segueixen una trajectòria recta fins aquest punt. Es va programar de manera que a la interfície de Unity es pogués programar el punt inicial i final.

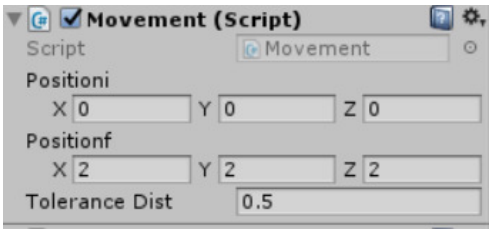


Figura 51 Configuració a Unity del script Movement

Com s'observa l'script, es crea la classe *Movement*, inicialitzada amb la definició de les variables:

- *positioni:* Vector que defineix la posició inicial de la diana.
- *positionf*: Vector que defineix la posició final de la diana.
- *targetDestination*: Vector que defineix quin serà el següent objectiu del moviment, la posició inicial o la final.

El programa utilitza la funció bàsica *Update*() per detectar la posició de l'objecte en tot moment. Mentre l'objecte no arribi al punt on dirigia, *targetDestination,* continuarà el seu moviment, en el moment en que es detecta que l'objecte es troba a la tolerància de dit punt, canvia el seu objectiu i es dirigeix al punt contrari. Els punt als que es pot dirigir són *positioni* i *positionf*.

Un cop creat i comprovat que el moviment de les dianes era l'adequat es va procedir a la configuració de les pilotes per tal que siguin objectes amb grasping habilitat per les mans del Leap Motion. Al ser una versió més antiga a la presentada al treball del mòdul *InteractionEngine*, versió, 0.3.1, funciona de manera diferent. El procediment a seguir és:

1- Afegir a l'escena un *InteractionManager* (prefab), que pel seu funcionament necessita associar-li a la seva configuració un el *LeapHandController* on trobem el *LeapProvider* i el *HandPool* amb el que volen que estigui especificat.

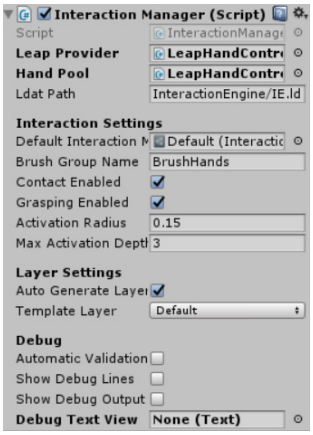


Figura 52 Menú del Interaction Manager

2- S'afegeix l'script *InteractionBehaviour* als objectes que es vol que s'habiliti el *grasping.* Aquest script requereix associar-li un *InteracionManager*. La seva funció es basa en, bàsicament, informar al *InteractionManager* sobre l'estat de l'objecte i informar-li que l'acció de les mans està activada sobre ell.

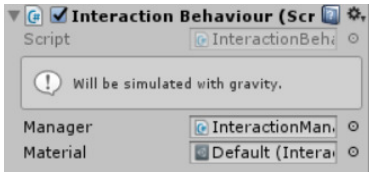


Figura 53 Menú del Interaction Behaviour

3- Al *HandPool* de l'InteractionManager és necessari que s'utilitzi com a model físic de mans, les mans *BrushHands*, incloses en el mòdul. Per tant, cal afegir un segon *size* en la configuració del *HandPool*.

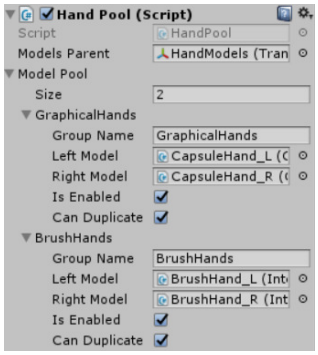


Figura 54 Opcions del menú HandPool

El següent pas en el disseny, va ser crear una series *d'scripts* que aconsegueixen fer que, un cop afegit un text en pantalla, sumin un punt cada cop que s'aconseguia fer que una pilota impacti amb l'objectiu. A més, es va voler afegir un so quan es produeixi l'impacte, per tal que el jugador detecti a través d'aquest so que havia aconseguir l'objectiu. Per aquest motiu, es va afegir a les dianes *l'script impactPointSound,* que consistia en el següent codi:

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

public class impactPointSound : MonoBehaviour
{
    public static AudioSource pam;
    void Start ()
    {
        pam= GameObject.Find("Punto").GetComponent<AudioSource>();
    }
    void OnTriggerEnter(Collider other)
    {
        Point.count++;
        pam.Play ();
    }
}
```

Figura 55 Imatge del script impactPointSound

ETSEIB

És senzill entendre el funcionament de *l'script,* que es basa en activar la variable *pam*, que està vinculada al clip de àudio que s'executarà en impactar la pilota amb la diana, i afegir a la variable *count* de la classe *Point* un nou punt (sumant-li 1). Pel que fa a *Point* es tracta d'un script associat a un *empty* amb el nom *GameObjects.* El funcionament de l'script es basa en que cada execució actualitza els textos de points, en cas de que es puntuï, de detectar quan totes les pilotes han desaparegut, i d'enviar el missatge de finalització de partida. L'script utilitzat el següent:

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

public class Point : MonoBehaviour {
    public static  Text countText;
    public static Text finalText;
    public static int ballsLeft;
    public static int count;

    void Start ()
    {
        count = 0;
        ballsLeft = 5;
        countText = GameObject.Find("Canvas/Text").GetComponent<Text>();
        finalText = GameObject.Find("Canvas/finalText").GetComponent<Text>();
    }

    // Update is called once per frame
    void Update () {
        countText.text = "Points: " + count.ToString();
        if (ballsLeft == 0)
        {
            if (count == 10)
            {
                finalText.text = "Perfect!";
            }
            else
            {
                finalText.text = "Game Over";
            }
        }
    }
}
```

Figura 56 Script Point

A l'script, quan s'inicialitza, es defineixen 4 variables:

- *countText*: Vvariable associada al text del Canvas que s'encarrega de mostrar la puntuació del jugador.
- *finalText:* Variable associada al text que apareixerà un cop el jugador s'hagi quedat sense pilotes i el joc acabi.
- *ballsLeft:* Variable que expressa el nombre de pilotes que resten al jugador.
- *count:* Variable que expressa el nombre de punts que el jugador a aconseguit.

A la funció *Update* de *l'script* s'actualitza el text *countText* en funció del valor que té *count*, per tant, cada cop que el jugador faci un punt, des de *l'script impactPointSound,* s'actualitzarà el valor de *count* sumant-li un punt, i en executar-se la funció *Update* el text canviarà sumant dit punt. Un cop el nombre de pilotes restant *ballsLeft* es redueixi a 0, un condicional farà que el joc decideixi com actualitzar el text de final de joc *finalText,* si el jugador a aconseguit fer els 10 punt, retornarà el text "Perfect!" en cas contrari retornarà "Game Over".

Aquest *script* necessita d'un mètode de detecció de quantes pilotes resten al jugador a la partida, per aquest motiu, s'inclouen uns plans que en impactar les pilotes contra ells es considerarà que les pilotes estan el suficientment allunyades de la plataforma de joc i que el jugador disposa d'una pilota menys. Es van crear 5 plans invisibles i es va delimitar l'àrea de joc, 4 parets i un terra. Aquestes parets incorporaven els següents scripts:

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

public class BallsLeft : MonoBehaviour {
    void OnTriggerExit(Collider other)
    {
        other.gameObject.SetActive(false);
        Point.ballsLeft--;
    }
}
```

Figura 57 Script BallsLeft

Aquest script detecta amb la funció *OnTriggerExit* si s'ha produït algun impacte, si és el cas, el GameObject definit com col·lisionador, amb el nom de *other,* es desactiva, per tant la pilota desapareix de l'escena, i la variable *ballsLeft* redueix en un el seu nombre.

7.2. Gesture Game

Els *Gesture Games* són jocs on aspectes fonamentals de jugabilitat es controlen a traves de la gesticulació amb les mans, és a dir, en funció dels moviments que es realitzin, el joc actuarà d'una manera o d'una altre. Per la creació d'aquest joc s'ha fet ús d'un mòdul de creació de Leap Motion que no està actualitzat per l'ultima versió de Unity i pels nous canvis, anomenat *DetectionExamples*. El nom del joc creat amb aquestes funcions es *The Rich Tunnel* a causa de la seva decoració amb materials valuosos, com or, plata i zèfirs.

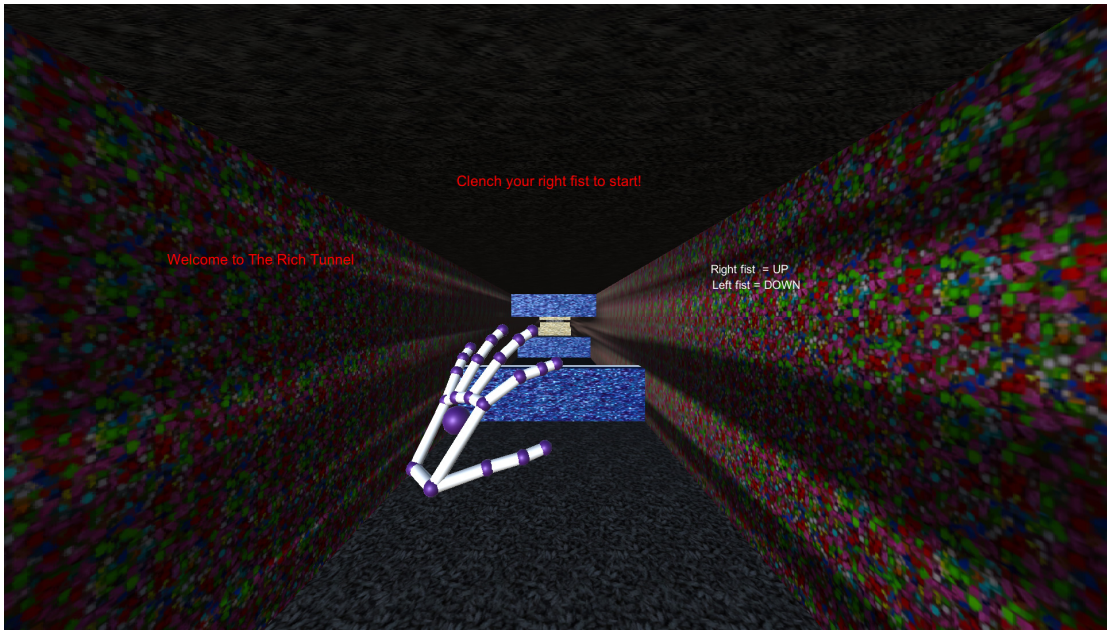


Figura 58 Imatge del joc The Rich Tunnel

El funcionament del joc consisteix en esquivar els diferents obstacles que trobem al túnel mentre el personatge avança de forma constant per l'interior del túnel. Utilitzant les nostres mans, sempre que tanquem el puny dret, el personatge s'elevarà a la part superior del túnel, en cas de tancar la mà esquerra, es baixarà fins el terra. L'objectiu és arribar al final del túnel, en cas de xoqui amb algun obstacle, es perd el joc.

Al igual que amb l'aplicació *grasping*, es tracta d'un joc d'escriptori. El primer pas per la creació de l'aplicació, com a tots els jocs que es basen en l'ús del Leap Motion, és la introducció a l'escena del prefab *LeapHandController.* Un cop posicionat i configurat adientment es va procedir a la creació del Tunnel, que consta de 4 parets, que en aquest cas no són plans sinó tetraedres sòlids, i els diferents obstacles a esquivar.

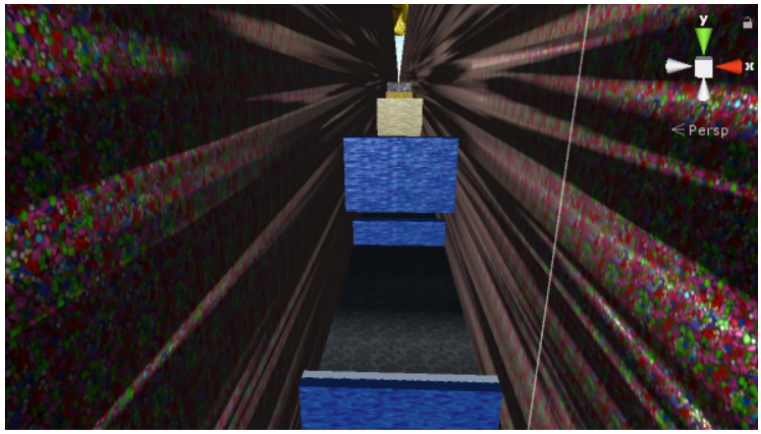


Figura 59 Col·locació dels obstacles

La primera posició de les parets no es tractava de la forma final, donat que encara no estava clar amb quin mecanisme i criteris es detectaria que el jugador ha xocat amb un obstacle, i per tant perdut. La decisió va ser que l'aspecte del jugador des del punt de vista del joc sigui un cub, donat que només es té en compte com a cos del jugador l'angle de visió. Es podria haver utilitzat qualsevol altre cos que avarques una superfície perpendicular al avanç semblant la projecció de l'angle de visió.

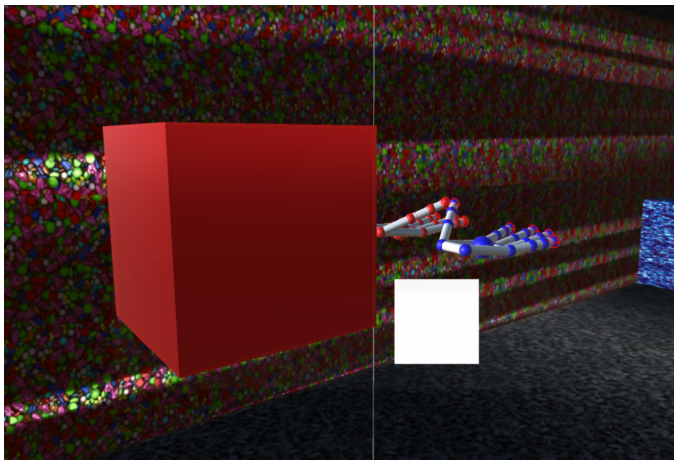


Figura 60 Player del joc The Rich Tunnel

Un cop creat, el jugador, es va procedir a la programació del jugador i els obstacles per fer que es mogui i el jugador i que sigui eliminat en cas de xocar amb un obstacle. Els scripts de la llibreria de detecció utilitzats per les mans, són un factor que determina com s'han de programar els elements que depenen de la gesticulació de les mans. El mètode programació s'ha basat en les possibilitats que oferien aquests *scripts.*

Els scripts que ofereix la llibreria per a desenvolupador d'aplicacions són:

- *ExtendedFingerDetector:* aquest script detecta si els dits de la mà es troben estesos o no, i permet actuar sobre altres elements de l'escena en funció de si es compleixen les condicions que defineix l'usuari sobre els dits. Es pot seleccionar a la interfase de Unity quina posició dels dits es vol detectar, quins elements es veuran afectats, quin efecte tindran sobre aquest elements i el nivell de doblegament dels dits. Aquest script a de situar-se al model de mà que es vol que sigui afectat i a més incloure'l a la seva configuració a la interface a l'apartat Hand Model.

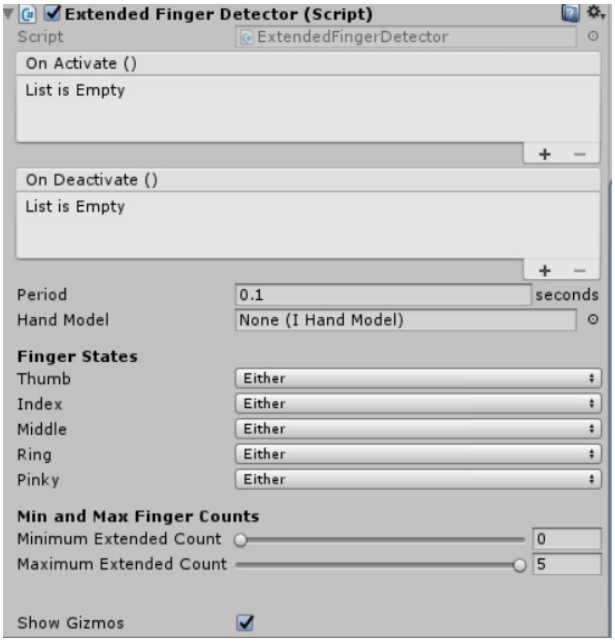


Figura 61 Menú del script ExtendedFingerDetector

- *FingerDirectionDetector*: Aquest *script* té un funcionament similar a l'anterior en quant a la manera d'actuar sobre l'escena, la diferencia és que es centra en la posició del vector direcció dels dits. Es programa per realitzar una funció o una altre en funció de la direcció del dit que es configuri amb *l'script*. Requereix la definició del període en segons en que es comprovarà la posició del dit, el HandModel al que pertany la mà, el dit que s'utilitzarà, la referència de posició (relativa a la referència absoluta, a l'horitzó o referència relativa al dit), la direcció a la que apuntarà en aquesta referència i l'angle de mesura des de la punta.

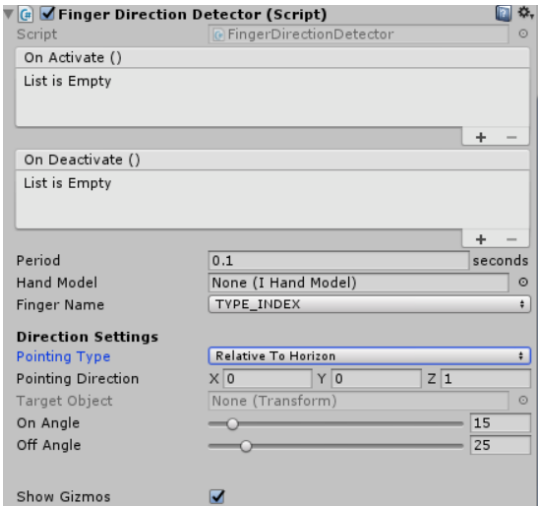


Figura 62 Menú del script DirectionFingerDetector

- *PalmDirectionDetector*: Aquest *script* s'encarrega de detectar la posició del vector perpendicular positiu a la palma de la mà, i actuar sobre l'escena d'igual manera que els anteriors. La seva configuració és exactament igual que la del FingerDirectionDetector, amb la diferencia que no cal definir-ne el dit.

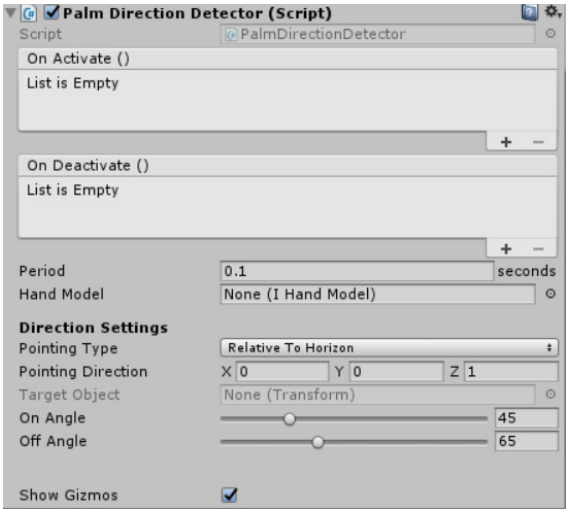


Figura 63 Menú del PalmDirectionDetector

La nova versió Orion va eliminar les funcions de gesture, fet que fa que la programació per gestos no es bassi en l'API de Leap Motion sinó en la programació d'objectes amb les funcions de la llibreria de Unity. Per aquest motiu, i per l'objectiu del treball, aquest joc pretén adaptar-se als mètodes de Leap Motion i els seus scripts es crearan tenint en compte la capacitat d'aquests.

La manera de combinar els *scripts* propis amb els de la llibreria de detecció es crear-ne petits scripts que s'activaran o desactivaran en funció del gest de la mà o bé crear-ne un gran *script* i activar únicament les seves funcions. En aquest joc, s'ha utilitzat el primer mètode, posteriorment al joc *VR Gesture Game,* es realitzarà la segona opció.

El script utilitzat per aquest joc ha estat el *ExtendedFingerDetector*, per detectar si la mà es trobava tancant el puny. El primer que es va programar és l'arrencada del personatge, és a dir, l'inici del seu moviment pel túnel. No és molt còmode que el joc comenci amb el personatge amb moviment, per tant es pretén que el moviment no tingui lloc fins que el jugador tanqui el puny dret per primer cop. En aquesta arrencada s'inclou un text en pantalla amb instruccions sobre com jugar i el nom del joc, que desapareixerà en iniciar el moviment. Per aquest motiu, es va crear l'script *Run:*

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

public class Forward : MonoBehaviour
{
    public float start = .1f;
    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        Vector3 run = new Vector3(0, 0, start);
        transform.Translate(run * Time.deltaTime);
    }
}
```

Figura 64 Script Forward

Un cop tancat el puny per primer cop, *l'script* romandrà actiu. La inicialització es basa en la creació d'una variable amb el nom *start* que s'utilitzarà per definir la velocitat en que es mourà el personatge mentre travessa el passadís. A la funció *Update* es definirà que el cos es mogui de forma constant en tot moment. Els moviment que del personatge que depenen del gestos, moviment de pujada i baixada, s'han programat amb el mateix script, l'única diferencia és que no romandran actius un cop detectat els gestos, sinó que es desactivaran en detectar que no es compleix el gest. D'aquesta manera, si tanquem el puny dreta, el personatge pujarà, si obrim el puny, el personatge es manté a la posició, el mateix passarà amb l'esquerra la única diferència és que en comptes de pujar baixarà. La forma que tindran els *ExtendedFingerDetector* de les mans serà la següent:

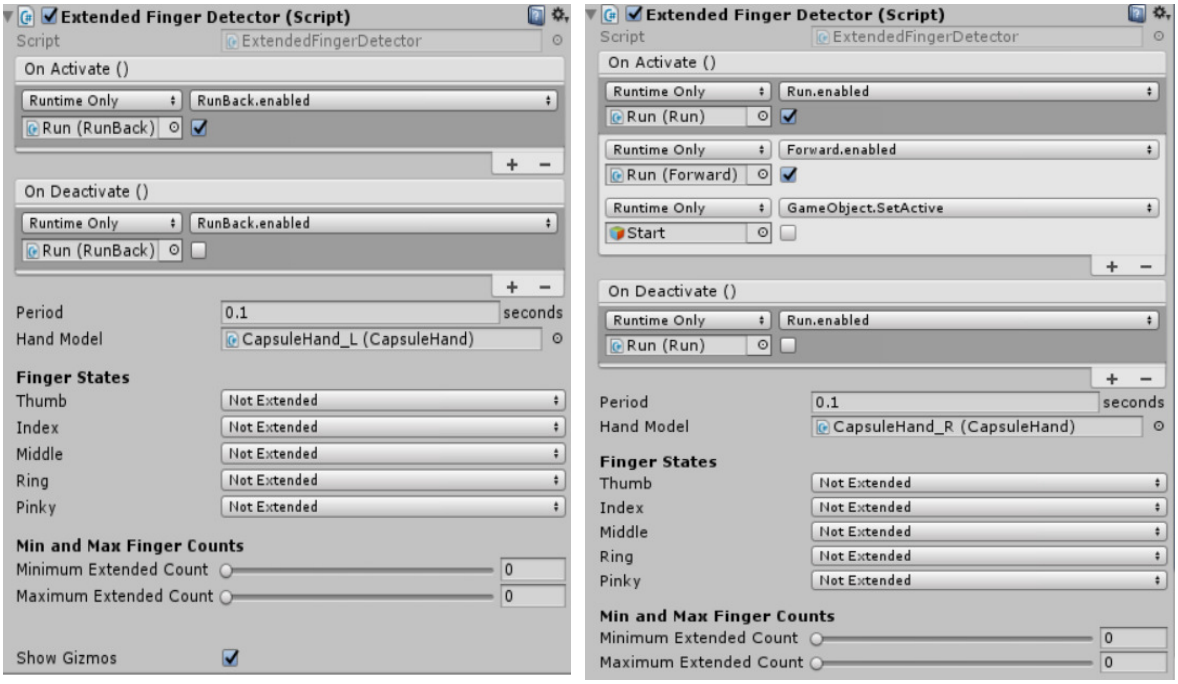


Figura 65 Configuració dels ExtendedFingerDetector de cada mà

Un cop finalitzada la programació del moviment del personatge, es van crear els *scripts* dels obstacles que fan que els jugadors perdin en xocar amb ells. Els *scripts* havien de ser capaços en cas d'impacte, de transportar la càmera de l'aplicació a una pantalla on aparegui un text de joc perdut, acabant el joc i indicant que el jugador ha perdut. El *script* dels obstacles té per nom CollisionL i és el següent:

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

public class CollisionL : MonoBehaviour {
    public GameObject canv;
    private void OnTriggerEnter(Collider col)
    {
        col.gameObject.SetActive(false);
        canv.SetActive(true);

    }
}
```

Figura 66 Script CollisionL

Al codi s'aprecia com el programa necessita que es defineixi des de la interfície de Unity un GameObject amb el nom de *canv* per inicialitzar-se. Amb la funció *OnTriggerEnter*, sempre que un objecte *collisioni* amb l'obstacle, objecte que únicament podrà ser el jugador, dit objecte *col* es deshabilitarà i desapareixerà de l'escena mentre que l'element *canv* s'activarà. En aquest cas, s'han fet ús de la lògica de la jerarquia per programar el final de la partida. Tot i que l'explicació del programa sigui confusa, amb l'ajuda de la jerarquia es pot explicar detalladament la seva lògica. Inicialment, el Canvas troba actiu el text de presentació que desapareix al tancar el puny dret per primer cop. El que passa quan s'impacta amb un obstacle és que l'objecte *Run* es desactiva, i amb ell, es desactiven els seus *childs,* per tant es desactivarà el controlador de Leap Motion i amb ell, el seu *child*, que es la *MainCamera,* la càmera que segueix al jugador. Just després s'activa l'element *canv*, que cam ja s'ha dit anteriorment és un GameObject que es defineix a la interfície. Es por observar que sota Canvas trobem un GameObject amb el nom *End*, es tracta d'un *empty* que inclou un *Text* de final de joc, pantalla de perdre, una *Camera* que en ser la única càmera activada, passarà a ser la principal, i un segon text *Text (1)* que és un petit text que indica al jugador com reiniciar el joc.

Figura 68 Jerarquia del joc

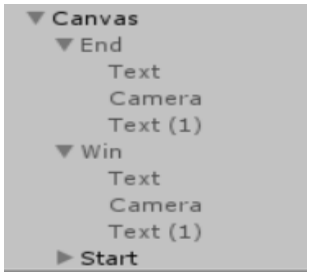


Figura 67 Imatge del Canvas

Figura 69 Imatge final del joc si es perd

Només queda definir el final del túnel, on el jugador si ha aconseguit esquivar tots els obstacles i per tant ha guanyat el joc. El final del túnel, gràficament, és un pla blanc il·luminat que simbolitza la llum de l'exterior.

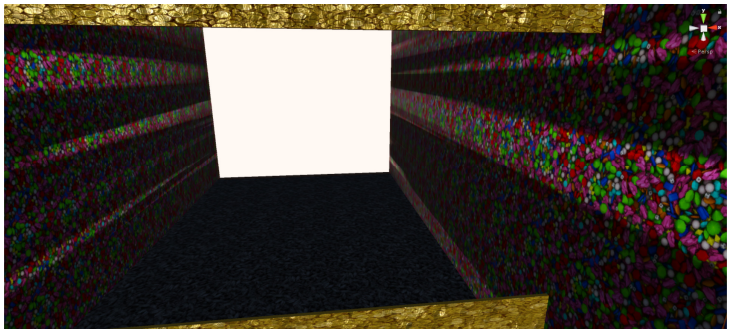


Figura 70 Final del tunel

S'ha de programar el pla per tal que quan el jugador impacti amb ell, aparegui la pantalla de guanyador del joc. *L'script* utilitzat és el mateix que s'ha utilitzat amb els obstacles, l'únic que canvia és que el paràmetre *canv* serà un altre *empty* idèntic amb un text de final de joc diferent. Aquest *empty* es diu *Win.*

Figura 71 Imatge final del joc si es guanya

7.3. VR Gesture Game

El VR Gesture Game combina l'ús del dispositiu Leap Motion i el HTC Vive en un joc controlat amb gestos de les mans. En aquest cas, no s'ha utilitzar el LMHeadMountedRig de Leap Motion de manera convencional, sinó que, a causa del disseny del joc i comoditat del jugador, s'ha canviat diversos aspectes de configuració que es comentaran a continuació.

El joc consisteix en la conducció d'un cotxe utilitzant gestos de les mans. La càmera del jugador es troba al front del cotxe i al ser un punt de vista des d'un casc VR té visió de 360 graus, tot i que no mobilitat, donat que en la conducció cal estar assegut. Quan el jugador posicioni el seu palmell dret mirant cap amunt, el cotxe accelerarà, si situa el palmell esquerre, el cotxe anirà marxa enrere. Tancant el puny dret el cotxe girarà cap a la dreta, i si tanquem l'esquerre cap a la esquerre. El circuit és un circuit d'aproximadament 3 minuts tenint en compte imprevistos i consta de senyals de velocitats per indicar al jugador cap on ha de girar i si ha d'augmentar o disminuir la velocitat.

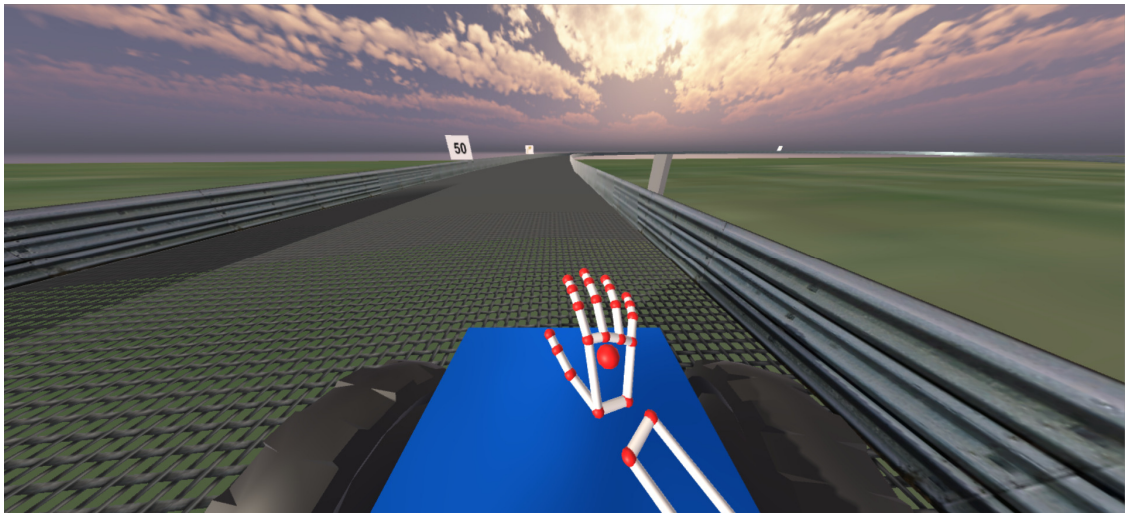


Figura 72 Imatge in-game del joc

La primera idea per la creació del joc va ser fer funcionar el cotxe amb un volant que es pogués moure amb *grasping,* creant un model que a primera vista és molt més pràctic i lògic, donat que seria una imitació del moviment que es fa amb el volant i simularia de millor manera la vida real. També es pretenia situar el dispositiu Leap Motion com es recomana a les indicacions d'ús a la posició dels ulls en el Casc HTC Vive, on les vitalitzem situant-les al nostre camp de visió. Desgraciadament, aquest model presentava una sèrie d'inconvenients, que eren:

- Detecció de les mans en posicions de gir eren poc precises, molt cops uns dits tapen uns altres i al visor no es genera una bona virtualització de la mà.

- El model de grasping no és el suficientment precís per un joc d'aquestes característiques

- No es còmode tenir que conduir mirant únicament les mans, seria l'única manera de poder virtualitzar constantment les nostres mans sense que la detecció de les mans falli.

Per aquests motius, es va decidir enfocar d'una manera diferent la creació del projecte. Es van canviar dos aspectes principals:

1- El vehicle serà dirigit en comptes de amb un volant sòlid rígid que interaccioni amb les mans, amb la gesticulació de les mans, donat que la precisió en la detecció de gestos té un funcionament més òptim que la alternativa anterior.

2- El dispositiu Leap Motion no es posicionarà al front del casc, com es recomana, es posicionarà a la falda del jugador, fent que no es requereixi estirar las mans i les mans no desapareguin tot i que el jugador no les estigui mirant.

Un cop analitzat quin mètode i en quines condicions es donaria la creació del joc, es va procedir a la seva creació. Primerament, es va situar a l'escena el *LMHeadMountedRig,* que, com ja s'ha comentat, ha de tenir un comportament diferent pel que fa al angle de detecció de les mans en VR. Ha de detectar les mans com si es tractés d'un joc d'escriptori però ha de situar la càmera a la visió del casc del jugador. Per aquest motiu, es van dur a terme algunes modificacions al seu *setup*:

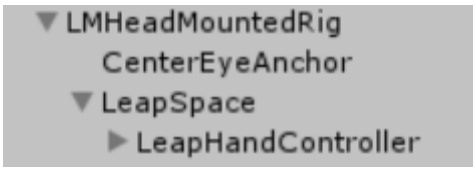


Figura 73 Canvis a la jerarquia del LMHeadMountedRig

- El *LeapHandController* es va configurar sense marcar la casella "*Is Head Mounted*", d'aquesta manera, l'angle de detecció del dispositiu és com el de la detecció escriptori.

- En la jerarquia, el *CenterEyeAnchor* no és *parent* del LeapHandController, això fa que la posició de la càmera no provoqui un canvi de posició al *LeapHandController*, que ha d'estar fix. Aquest GameObject només ha de contenir el script de *Leap VR Camera Control* que és el que el connectarà amb el casc de realitat virtual.

Amb aquests canvis es suficient per assolir la configuració desitjada. Les mans es mouran sense dependència de la posició del casc i amb un angle de detecció escriptori.

Un cop configurat el controlador del Leap Motion, es va incorporar a l'escena del joc la pista per on el cotxe circularà. En aquest cas, es va importar un pack d'assets amb el nom *KMSkyTracks,* disponible a la *Asset Store* de Unity de forma gratuïta. La pista utilitzada és un *prefab* amb el nom de *SkyTrack05.*

Com detall gràfic, també es va afegir una cúpula amb aspecte de tarda, obtingut a un altre *Asset Package* de la *Asset Store,* amb el nom *Skybox.* Aquest també es completament gratuït i l'asset utilitzar té el nom de *Skybox_Sunset*.

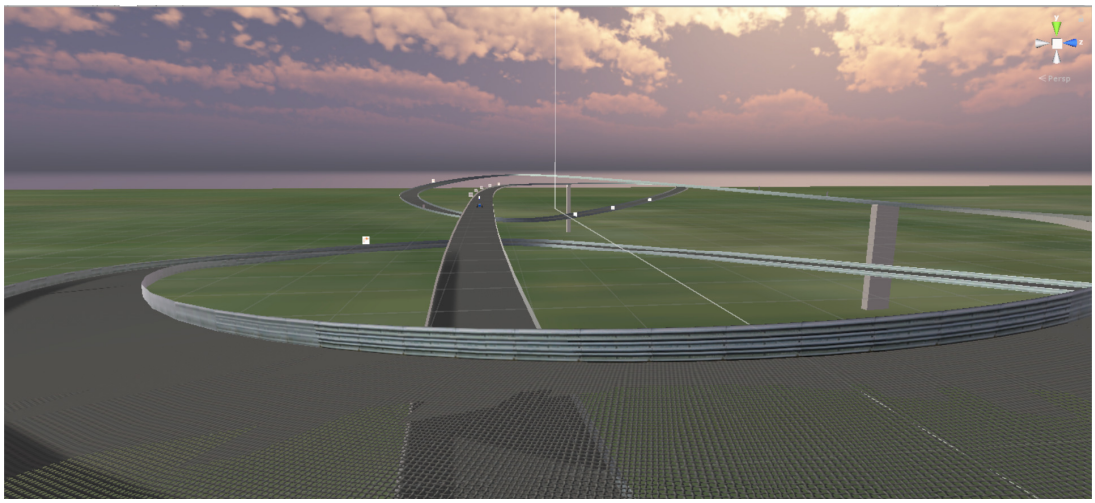


Figura 74 Imatge del SkyTrack05

Tots els cartells de gir i velocitat que ha de tenir el cotxe en tot moment ja es troben incorporats al prefab de la pista.

El següent pas es l'addició d'un cotxe que es pugui conduir. El disseny del cotxe també es va importat d'un *Asset Pack* gratuït, en aquest cas, del *Demo Car,* i el disseny escollit va ser el prefab *Car_Demo.*

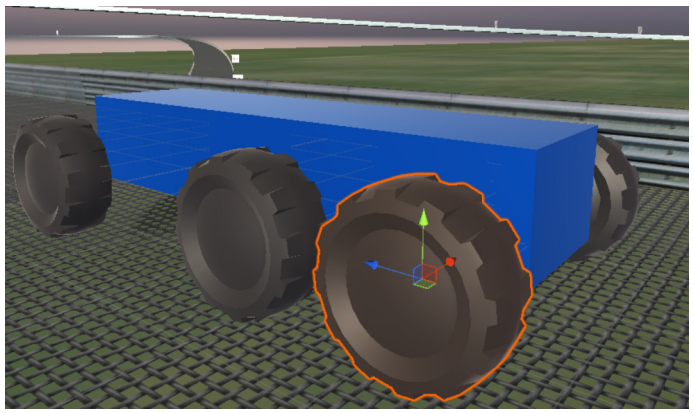


Figura 75 Model del cotxe

La programació dels cotxes consta d'opcions a la llibreria de la API de Unity que faciliten la seva programació. La funció bàsica del paquet és la funció *WheelCollider,* aquesta treballa amb el colliders de les rodes del GameObject al que es troba implementat, per que funcioni, cal que la jerarquia sigui contingui un *empty* que sigui *child* del vehicle que contingui els colliders amb el nom "*Car Colliders*". A partir de la definició de dita funció al codi, es poden manipular les rodes del cotxe amb relativa facilitat, tot i que es una llibreria que permet el seu moviment però que no programa tots els aspectes d'angles de gir de les rodes, acceleracions apropiades, angles de gir de rodes entre d'altres. La llibreria necessària per l'ús d'aquestes funcions es troba inclosa a *System.Collections.Generics.*

Com ja s'havia avançat a l'explicació dels scripts de *gestures* del *Gesture Game* en aquest cas la programació en funció de gestos es realitzarà amb petites funcions incloses en un únic script que s'anirà cridant sempre que es compleixi la condició de gest de la mà. El *script* creat té el nom de Dot_Truck_Controller, i és una adaptació del codi original del prefab:

```
public class Dot_Truck_Controller : MonoBehaviour
{

    public List<Dot_Truck> truck_Infos;
    public float maxMotorTorque;
    public float maxSteeringAngle;

    public void VisualizeWheel(Dot_Truck wheelPair)
    {
        Quaternion rot;
        Vector3 pos;
        wheelPair.leftWheel.GetWorldPose(out pos, out rot);
        wheelPair.leftWheelMesh.transform.position = pos;
        wheelPair.leftWheelMesh.transform.rotation = rot;
        wheelPair.rightWheel.GetWorldPose(out pos, out rot);
        wheelPair.rightWheelMesh.transform.position = pos;
        wheelPair.rightWheelMesh.transform.rotation = rot;
    }

    public void Up()
    {
        float motor = maxMotorTorque;
        foreach (Dot_Truck truck_Info in truck_Infos)
        {
            truck_Info.leftWheel.motorTorque = motor;
            truck_Info.rightWheel.motorTorque = motor;
            VisualizeWheel(truck_Info);
        }
    }

    public void Down()
    {
        float motor = -maxMotorTorque;
        foreach (Dot_Truck truck_Info in truck_Infos)
        {
            truck_Info.leftWheel.motorTorque = motor;
            truck_Info.rightWheel.motorTorque = motor;
            VisualizeWheel(truck_Info);
        }
    }
```

Figura 76 Script Dot_Truck_Controller, part 1

```
public void Left()
{
    float steering = maxSteeringAngle;
    foreach (Dot_Truck truck_Info in truck_Infos)
    {
        truck_Info.leftWheel.steerAngle = truck_Info.rightWheel.steerAngle =
            ((truck_Info.reverseTurn) ? 1 : -1) * steering;
        VisualizeWheel(truck_Info);
    }

}

public void Right()
{
    float steering = maxSteeringAngle;
    foreach (Dot_Truck truck_Info in truck_Infos)
    {
        truck_Info.leftWheel.steerAngle = truck_Info.rightWheel.steerAngle =
            ((truck_Info.reverseTurn) ? -1 : 1) * steering;
        VisualizeWheel(truck_Info);
    }
}

public void Rect()
{
    float steering = 0;
    foreach (Dot_Truck truck_Info in truck_Infos)
    {
        truck_Info.leftWheel.steerAngle = truck_Info.rightWheel.steerAngle =
            ((truck_Info.reverseTurn) ? -1 : 1) * steering;
        truck_Info.leftWheel.motorTorque = 0;
        truck_Info.rightWheel.motorTorque = 0;
        VisualizeWheel(truck_Info);
    }

}
```

Figura 77 Script Dot_Truck_Controller, part 2

En inicialitzar la funció es creen una sèrie de paràmetres:

- *maxMotorTorque*: És un nombre real que ha de definir-se des de la interfície de Unity, i és l'acceleració d'avanç del cotxe. Es mesura en rev/s^2.

- *maxSteeringAngle*: És l'angle màxim de gir de les rodes del cotxe definida com a nombre real. L'angle es mesura en graus.

- *truck_Infos:* Llista que conté el conjunt de característiques actualitzades de la situació de les rodes del cotxe. Tal i com es pot observar a la funció VisualizeWheel, gràcies a aquesta llista i a les funcions de la classe *wheelPair*, es pot obtenir la informació sempre que es cridi la funció sobre la posició i rotació de les rodes.

Amb l'ús de les classes implícites que inclou de *wheelPair,* podem obtenir informació sobre posició i rotació dels objectes de les rodes, *rightWheel*/*LeftWheel,* així com dels seus colliders, *rightWheelMesh/LeftWheelMesh.*

Utilitzant els mecanismes de control de les llibreries, es creen les petites funcions de la classe Dot_Truck_*Controller* encarregades de controlar el moviment del vehicle:

- *Up*/*Down:* són les funcions encarregades de dur a terme l'acceleració i la marxa enrere del vehicle sempre que el palmell dreta o esquerre estigui mirant cap a dalt, en la referència absoluta del sistema. L'algoritme de tots dos, és pràcticament el mateix, l'única diferència és que l'algoritme *Up* l'acceleració serà positiva i en el *Down* serà negativa.

Pel que fa al seu funcionament, inicialment es defineix la funció una variable *float* amb el nom *motor*, que representa l'acceleració a la que accelerarà el vehicle, i com s'observa s'obté de la variable pública *maxMotorTorque*. Després, es crida a la llista definida anteriorment com *truck_Info* i a través de les classes de la llibreria *leftWheel/RightWheel* es crida al paràmetre *motorTorque*, a través de la qual es produeix el moviment de les rodes a l'acceleració a la que es troba definit el seu valor, en aquest cas, amb el valor del paràmetre *motor.*

```
public void Up()
{
    float motor = maxMotorTorque;
    foreach (Dot_Truck truck_Info in truck_Infos)
    {
        truck_Info.leftWheel.motorTorque = motor;
        truck_Info.rightWheel.motorTorque = motor;
        VisualizeWheel(truck_Info);
    }
}
```

Figura 78 Funció Up()

Pel que fa a les funcions de gir, estan definides *Right* pel gir a dretes i *Left* pel gir a esquerres. Al igual que amb el cas de *Up* i *Down*, les dues funcions, utilitzen un algoritme igual però amb diferent sentit en els graus de rotació de les rodes. En aquest cas, inicialment, es defineix una variable *float* steering que rep el valor de la variable pública *maxSteeringAngle.* Al igual que amb les funcions anteriors, es crida la llista *truck_Info,* però en aquest cas, s'utilitza un altre paràmetre de les classes *rightWheel* i *leftWheel,* es tracta del paràmetre *steerAngle*, amb el qual es defineix l'angle de rotació de les rodes. En la funció, es fa girar les dues rodes cap al mateix sentit, definint un vector reverze turn que defineix el sentit, positiu o negatiu, i la referència, positiva o negativa, tot multiplicat per l'angle del gir de les rodes *steering*.

```
public void Left()
{
    float steering = maxSteeringAngle;
    foreach (Dot_Truck truck_Info in truck_Infos)
    {
        truck_Info.leftWheel.steerAngle = truck_Info.rightWheel.steerAngle =
            ((truck_Info.reverseTurn) ? 1 : -1) * steering;
        VisualizeWheel(truck_Info);
    }

}
```

Figura 79 Funció Left()

Finalment, es defineix una funció que rectifiqui el moviment del cotxe i provoqui que en no detectar cap dels dos algoritmes faci que l'acceleració del vehicle torni a ser nul·la i el cotxe circuli en línia recta un altre cop. Aquesta funció és una combinació de les altres dues, fent que es defineixin els *steerAngle* i *motorTorque de* les classes roda amb valor que facin que el cotxe circuli recte, que són 0 per tots dos.

```
public void Rect()
{
    float steering = 0;
    foreach (Dot_Truck truck_Info in truck_Infos)
    {
        truck_Info.leftWheel.steerAngle = truck_Info.rightWheel.steerAngle =
            ((truck_Info.reverseTurn) ? -1 : 1) * steering;
        truck_Info.leftWheel.motorTorque = 0;
        truck_Info.rightWheel.motorTorque = 0;
        VisualizeWheel(truck_Info);
    }

}
```

Figura 80 Funció Rect()

Aquesta classe només s'aplica a una fila de rodes, per tant cal fer una configuració per a cadascuna de les tres files de rodes del model del cotxe. La configuració final de cada fila de cotxes ha estar que mentre la fila davantera giri al sentit desitjar, les altres línies giraran al contrari, fent que el gir sigui més oblic.

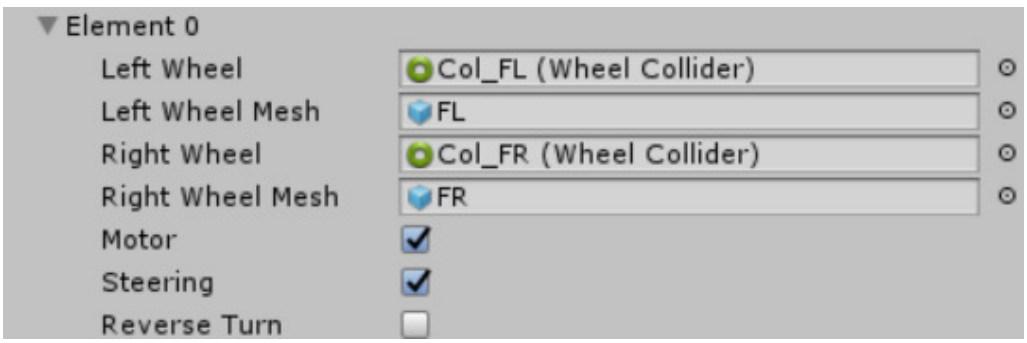


Figura 81 Exemple de Element de la classe Dot_Truck_Controller

Un cop tenim totes les funcions definides i es comprova que compleixen la funció desitjada, s'han de configurar els *scripts* de *gesture.* Tenint en compte els mètodes de configuració que ja s'han explicat anteriorment, els scripts quedarien configurats de la següent manera:

- **Ma dreta:**

  Palmell:

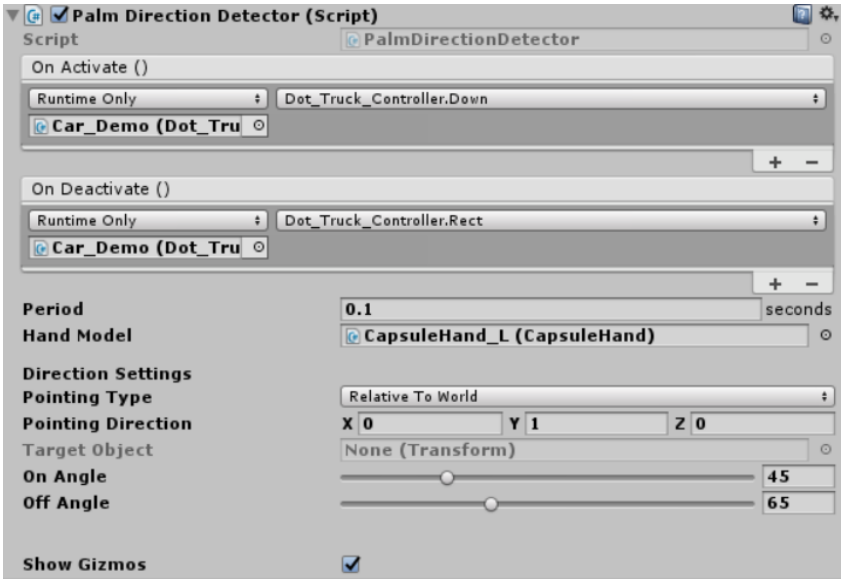


Figura 82 Configuració del PalmDirectionDetector R

  Dits:

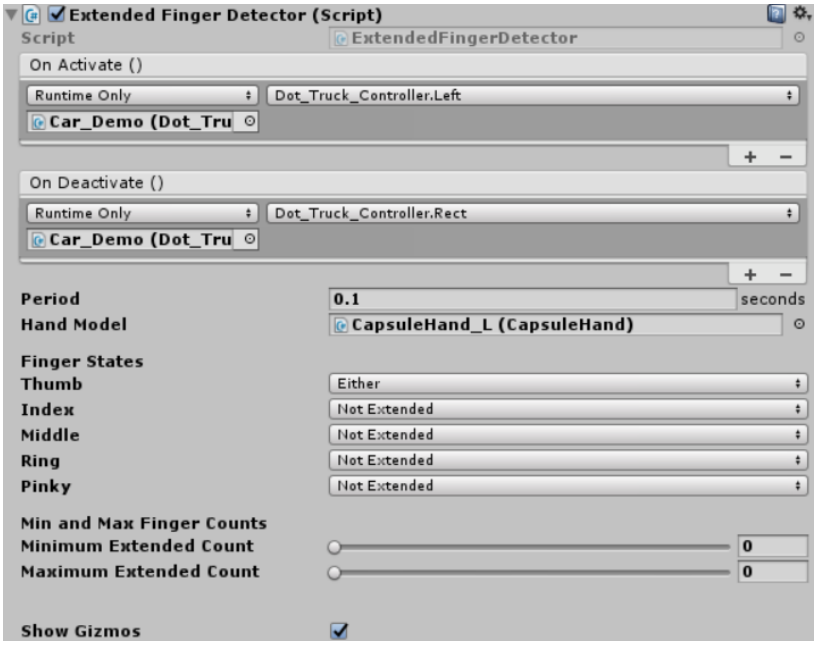


Figura 83 Configuració del ExtendedFingerDerector R

- **Ma esquerre:**

  Palmell:

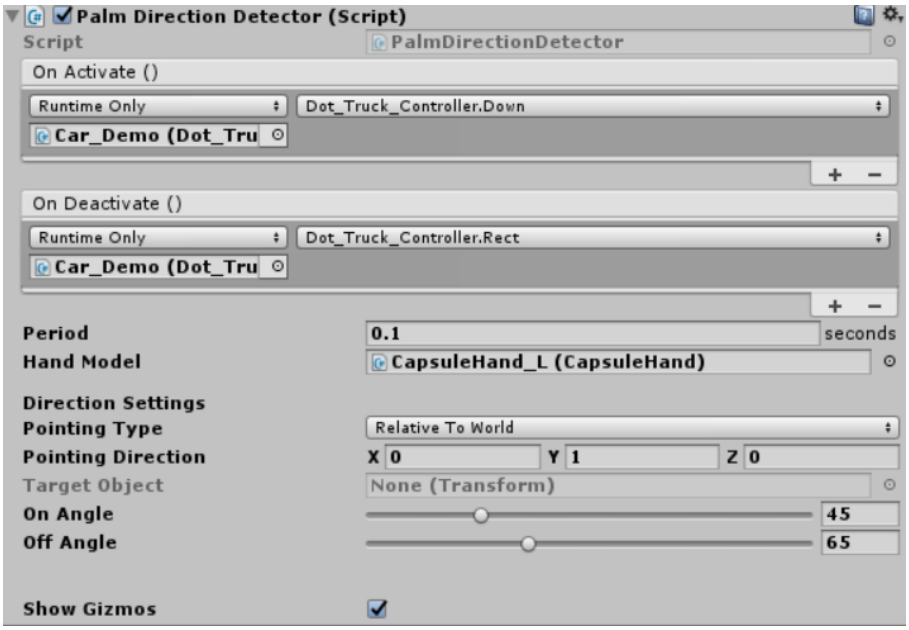


Figura 84 Configuració del PalmDirectionDetector L

Dits:

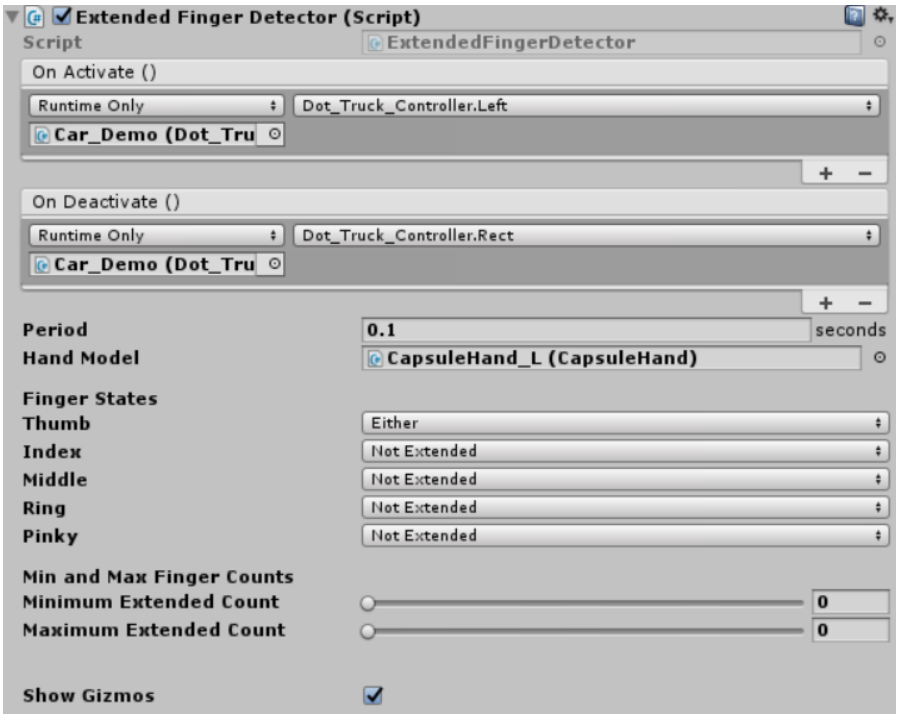


Figura 85 Configuració del FingerDirectionDetector L

Una funció addicional, afegida en última instància, és una funció que fa aparèixer el títol del joc, es farà desaparèixer quan el jugador senyali amb la direcció d'acceleració del cotxe amb la mà dreta.

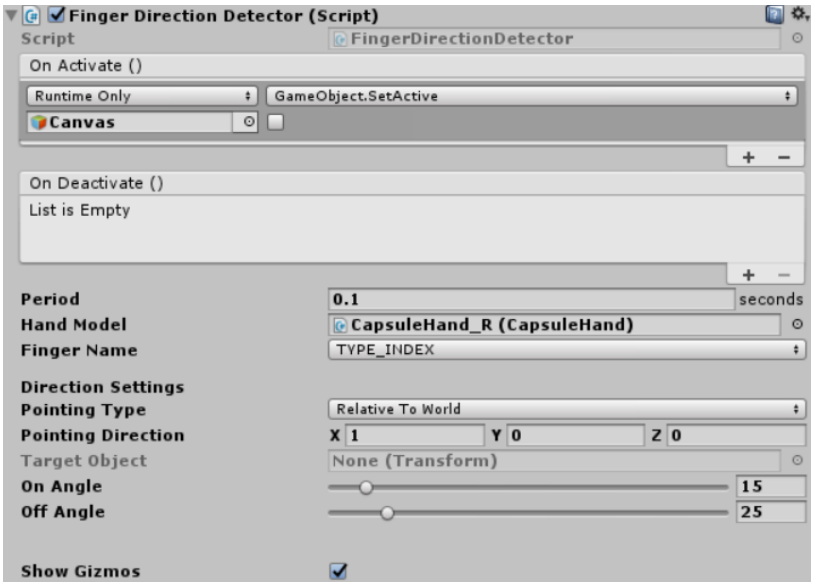


Figura 86 Configuració FingerDirectionDetector de la pantalla d'inici

El joc ja està completament definit i es van realitzar dues hores de *testeig*. El que es va comprovar amb facilitat, és que no és un joc en el que els girs hagin de ser prolongats sinó que han de ser petits girs els que facin girar el cotxe. Traduint això a la jugabilitat, no cal tancar la mà durant una llarga estona per aconseguir girar, sinó es desestabilitzarà el cotxe i xocarà, és preferible fer petits tancaments que provoquin un gir menys brusc. És molt necessari utilitzar les senyals de velocitat com a referència i no excedir-se accelerant.

7.4. Comparativa de la programació Gesture Game Vs VR Gesture Game i el Clean Code

Pel que fa als mètodes programació utilitzat en els dos joc, hi ha una diferència fonamental que els diferencia en termes d'optimització i enteniment del codi. El mètode utilitzat pel *Gesture Game* és la creació de petits *scripts* que s'activin i desactivin en funció dels gestos del jugador, mentre que en el *VR Gesture Game* es va optar per l'opció de la creació de petites funcions que realitzin les accions en funció dels gestos.

Actualment no existeixen especificacions oficials a l'hora de programar, però si que existeixen recomanacions que en la programació professional es consideren necessàries i que determinen la qualitat d'un codi. Un dels mètodes de programació òptima més utilitzat per programadors experts és el que s'explica al llibre "Clean Code: A Handbook of Agile Software Craftmanship" de Robert Martin. L'autor del llibre explica els motius pels quals és necessari que els codis siguin *clean* i s'hagi d'evitar males pràctiques i no es generin *hard codes* a la producció. Es considera un llibre de culte per les persones que es dediquen al món de la programació.

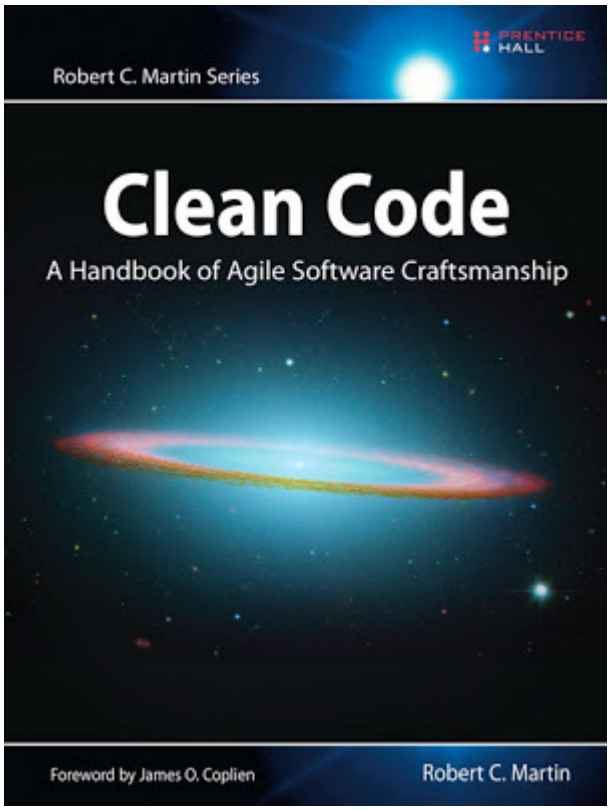


Figura 87 Portada del Llibre "Clean Code: A Handbook of Agile Software Craftmanship"

El llibre presenta una sèrie de requisits i principis bàsics per les bones pràctiques de programació, un recull molt general d'aquests principis és:

1- *DRY* o *Don't Repeat Yourself*: La repetició de codis fa que aquest sigui més difícil de mantenir i comprendre i molts cops genera inconsistències. Cal evitar sempre que sigui possible la repetició de codi en el disseny.

2- *The Principle of Least Surprise:* Les funcions han de realitzar la funció que s'esperi d'elles, i el seu nom ha d'indicar de manera obvia el seu funcionament.

3- *The Boy Scout Rule*: Sempre que es realitzi un canvi en un codi ja creat, cal tenir

cura de no realitzar repeticions i de que el codi quedi igual o més net del que ja estava. Si cal, s'accepta fer correccions del codi sempre que estiguin justificades.

4- *Single Responsibility Principle*: Fa referència al disseny de classes, on l'únic motiu que pot provocar la seva modificació ha de ser que la classe no compleixi la seva responsabilitat, que es complir la funció dins del codi per la qual ha estat creada.

5- *Open Closed Principle*: Una classe ha d'estar oberta a extensions que la potenciïn si cal, però mai a una modificació de la seva funció.

6- *Keep It Simple Stupid*: El disseny de la programació ha de ser el més senzill, pel que fa la seva complexitat de algoritme, ordre de creació i simplicitat a les crides.

Aplicant el criteri *Keep It Simple Stupid* sobre els dos disseny, veiem que en el disseny de *Gesture Game* s'està generant un nombre de crides a arxius que en el *Gesture Game VR* s'estan evitant, i aquesta petita diferencia afecta a l'optimització del codi i al seu caràcter de netedat o *clean.*

No genera una gran diferència i les dues implementacions són pràcticament iguals en quant a la seva optimització, però tenint en compte aquesta diferència, es podria dir que el codi de *Gesture Game VR* és lleugerament més *clean*.

8. Planificació temporal i pressupost

8.1. Fases de desenvolupament

El desenvolupament del projecte passa per diferents fases amb un nivell d'implicació, esforç i temps diferent. En aquesta part s'han puntuat amb punts d'esforç, seguint els paràmetres definits amb la filosofia *Scrum* de gestió de projectes, on les puntuacions d'un *sprint plan* per a cada tasca són: *0, ½ , 1, 2, 3, 5, 8, 13, 20, 40, 100, infinit, ?.* També s'ha realitzat una estimació en hores del temps d'implicació per a cada tasca. El resultat es el següent:

1- **Investigació** (5p / 25h): És la tasca inicial i consisteix en la recerca d'informació sobre el tema tractar i els aspectes que engloba.

2- **Reunions** (0p / 10h): Informar i demanar consell al tutor de projecte. La tasca en sí, requereix un esforç nul.

3- **Aprenentatge** (20p / 60h): Habituació, enteniment i aprenentatge en els mètodes de programació utilitzat al treball.

4- **Desenvolupament d'aplicacions** (40p / 85h): Creació de les aplicacions, procés creatiu, ideació del model de creació e implementació.

5- **Memòria** (8p / 80h): redacció de l'informe del projecte.

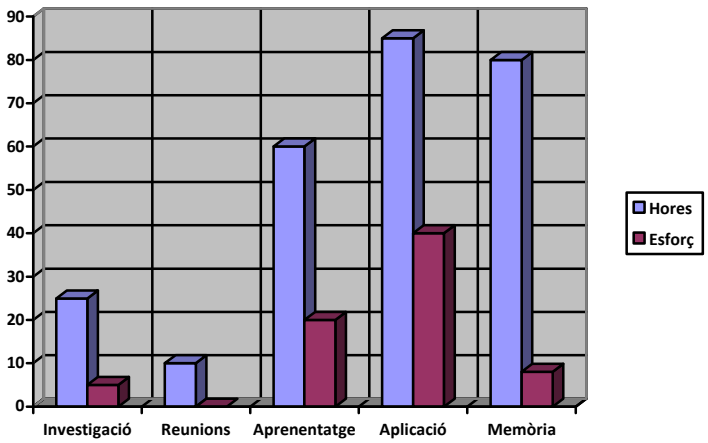


Figura 88 Gràfic comparatiu de temps de treball i punts d'esforç entre tasques

8.2. Pressupost del projecte

El personal necessari pel desenvolupament del projecte és un enginyer, informàtic o físic amb nocions de programació, l'especialitat d'estudi no és important. La seva funció és la creació de l'aplicació. El seu salari es considera de 20€. Pel que fa als costos de *hardware,* el material necessari és:

- Ordinador amb capacitats mínimes per suportar el funcionament de HTC Vive i Leap Motion.
- Dispositiu Leap Motion.
- Dispositiu HTC Vive, sense controladors.

El *software* necessari per la realització del projecte és el següent

- Programa Unity 3D, v.2017.1.0f3.
- Software HTC Vive + Leap Motion Orion
- Paquet de Microsoft Office (Word, PowerPoint i Excel). + Visual Studio

El resum de costos es troba a la taula següent:

Concepte	Cost unitari	Quantitat	Cost total (€)
Enginyer	20€/h	260h	5200
Subtotal			5200
Unity 3D	0 €/u	1u	0
Software HTC Vive + Leap Motion	0 €/u	1u	0
Paquet Office	189€/u	1u	189
Subtotal			189
Ordinador	900€/u	1u	900
Leap Motion	80€/u	1u	80
Casc HTC Vive	712€/u	1u	712
Consum elèctric	100		100
Subtotal			1792
Total			7181

Taula 4. Resum de costos del projecte

9. Impacte medi ambiental

En qualsevol projecte de l'enginyeria moderna, és necessari fer un anàlisi de l'impacte que tindrà sobre el medi ambient i avaluar si realment és un projecte que no provocarà un sobreexplotació de medis o un us excessiu/injustificat de recursos de l'àrea. Al tractar-se d'un projecte informàtic, els aspectes a analitzar sobre l'efecte mediambiental que engloba es redueix significativament.

L'únic aspecte analitzable, si considerem que l'ús de electricitat es controla per no generar un malbaratament, és l'impacte medi ambiental de la fabricació i transport de l'ordinador el dispositiu Leap Motion i el HTC Vive. L'únic aspecte controlable i que realment forma part del projecte en sí, i no del projecte de creació i distribució dels aparells, és el mètode de transport d'aquests. Al no ser un projecte completament definit, i sense dades d'informació sobre llocs de compra, no es podria definir un mètode de transport que redueixi l'impacte ambiental.

10. Conclusions

L'objectiu principal del projecte és la recerca sobre la combinació de dos potents aparells de realitat virtual, Leap Motion i HTC Vive, i l'estudi de la seva combinació. A més es pretenia crear aplicacions que mostrin el potencial d'aquests dos aparells i les seves possibilitats en l'aspecte de programació. Inicialment, es partia d'un nivell de coneixement nul en programació de videojocs, procediments habituals i funcionament tecnològic dels aparells a estudiar, des de l'inici del treball a l'entrega s'ha produït una evolució constant pel que fa als coneixements que caracteritzen aquests tipus de tecnologies i la manera de programar-les.

Una conclusió que s'extreu del projecte, és que és possible l'autoaprenentatge en el món de la programació de videojocs, sempre s'ha de tenir en compte que mai tot és senzill i la tasca de recerca ha de ser el suficientment exhaustiva per trobar-se informació amb la utilitat i veracitat suficient. Aquesta conclusió inclou, que és possible per un estudiant d'enginyeria realitzar un informe detallat d'aquest tipus de tecnologia i la seva manera específica de ser programada.

La conclusió principal que finalment exposa l'informe és que realment la combinació d'aquestes dues tecnologies pot utilitzar-se pel desenvolupament d'aplicacions molt diverses i d'usos molt variats, amb una qualitat relativament alta. No obstant, pel que fa al dispositiu Leap Motion, encara és massa aviat per dir que es troba totalment perfeccionat. Hi ha molts problemes de detecció i precisió, com és lògic, per les limitacions que presenta pel que fa als angles de detecció de les mans que es troben en un únic pla. La seva programació no es troba en una versió definitiva, però no s'allunya de la seva versió final, i no mostra una capacitat fiable per desenvolupar aplicacions realment serioses i que requereixen una gran precisió. Com podrien ser simuladors de conducció realistes, simulacions de muntatges tècnics o aplicacions destinades al món de la medicina.

Agraïments

Cal agrair per la motivació, aportació de idees i la oferta per la realització del treball al tutor del treball el professor Toni Susin. També a tot el Departament de Realitat Virtual de La facultat de Matemàtiques i Estadística de la UPC, destacant tota l'ajuda rebuda per Jordi Moyés, persona que va permetre sempre que era necessari l'accés al departament i va ajudar en el desenvolupament del treball amb tot el que va tenir a les seves mans.

I de manera personal, agrair al meu germà Federico Ruau tot el suport tècnic rebut i els seus consells orientatius, a més del gran suport moral en la realització del treball.

Bibliografia

Referències bibliogràfiques

[1] <<Unity Technologies. Lloc web del programa Unity 3D>> [En línia] Available: https://unity3d.com/es

[2] <<Manual d'usuaris del programa Unity 3D>> [En línia] Available: https://docs.unity3d.com/Manual/index.html

[3] <<Web de developers de Leap Motion Orion>> [En línia] Available: https://developer-archive.Leap Motion.com/unity

[4] <<Web oficial de HTC Vive>> [En línia] Available: http://www.htc.com/es/virtual-reality/

[5] <<Aprendre a crear videojocs amb Unity 3D>> [En línia] Available: https://www.tutellus.com/tecnologia/videojuegos/creacion-de-videojuegos-con-unity-3d-3291

[6] <<Introducció al Clean Code>> [En línea] Available: https://www.adictosaltrabajo.com/tutoriales/clean-code-reglas-principios/

[7] <<Asset Store de Unity>> [En línia] Available: https://www.assetstore.unity3d.com/en/#!/content/12639

Bibliografía complementària

[-] “Clean Code: A Handbook of Agile Software Craftsmanship” – Robert C. Martin [Llibre]

[-] “Diseño de videojuegos” – Daniel Gonzalez [Llibre]

Annexos

Annex 1: script de optimització de frames, tancat i reiniciació d'una escena

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

public class Optimize : MonoBehaviour
{

    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        Application.targetFrameRate = 60;
        if (Input.GetKey("escape"))
            Application.Quit();
        if (Input.GetKey("r"))
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        }
    }
}
```

Annex 2: Scripts Run i RunBack de Gesture Game

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

public class RunBack : MonoBehaviour
{
    public float start = .1f;
    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        Vector3 run = new Vector3(0, -start, 0);
        transform.Translate(run * Time.deltaTime);
    }
}
```

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

public class Run : MonoBehaviour
{
    public float start = .1f;
    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        Vector3 run = new Vector3(0, start, 0);
        transform.Translate(run * Time.deltaTime);
    }
}
```