Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Desenvolupament d'un videojoc de ritme en realitat virtual sobre tocar la bateria.

Document: Memòria

Alumne: Joan Tíscar Verdiell

Tutor: Gustavo Patow
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i sistemes informàtics

Convocatòria (mes/any): Setembre 2023

Índex

1. Introducció i objectius

1.1. Introducció

En el món dels videojocs, un dels sectors que està creixent enormement en els darrers anys, és el dels jocs en realitat virtual. Els avenços en aquestes tecnologies fan que sigui cada cop més plausible fer jocs que permetin simular activitats que necessiten molt de hardware i/o espai per a poder-se practicar en el món real.

L'objectiu d'aquest TFG és crear un videojoc de ritme, en el que el jugador pugui tocar una bateria virtual, sense els inconvenients d'adquirir una bateria electrònica, com pot ser el cost o l'espai.

Els videojocs de ritme són un gènere que consisteix en simular l'execució d'un instrument musical o d'un ball, on el jugador ha d'intentar aconseguir la major puntuació possible, executant aquests moviments amb la millor sincronització possible amb la música.

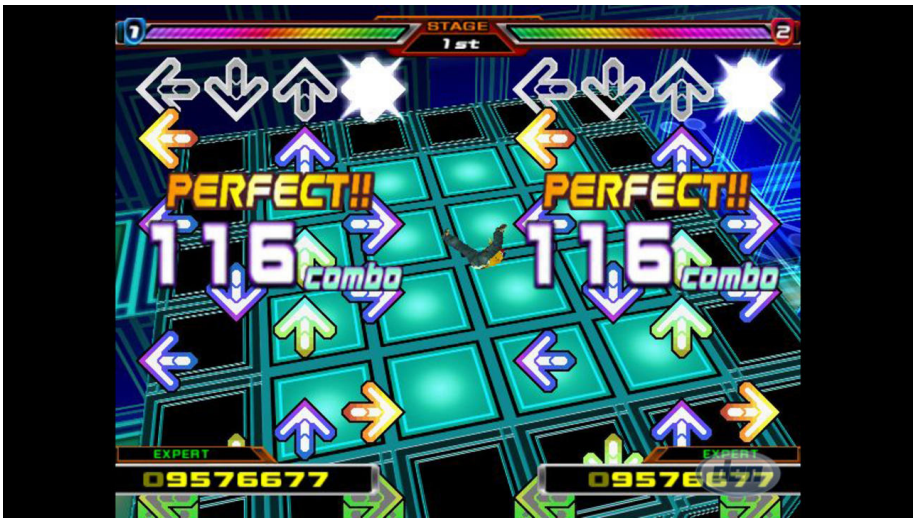


Figura 1: Captura del joc Dance Dance Revolution

1.2. Motivacions

Per a triar el tema per a fer aquest treball, hi van tenir pes les següents motivacions personals

1. Voler treballar amb tecnologies de realitat virtual
2. Interès en fer un treball amb relació amb la música
3. Obtenir un producte final que serà d'utilitat per a gent que no disposa de l'espai necessari per a tenir una bateria a casa o què canvien freqüentment de pis (per exemple, estudiants)

1.3. Propòsit i objectius del projecte

El propòsit del projecte és el de crear un prototip que sigui capaç de proporcionar l'entorn per a que el jugador pugui practicar amb la bateria i que sigui capaç de poder introduir al joc les cançons o lliçons que ell vulgui per a practicar.
De manera més concreta, els objectius són els següents:

- Aprendre i dominar els complements de VR a Unity
- Creació del terreny de joc
- Creació de les diferents parts de la bateria i les interaccions amb el jugador
- Disseny de les interfícies i elements del joc
- Definició i implementació de les diferents modalitats de joc
- Disseny d'un sistema de puntuació
- Implementació del format DTX per a poder importar cançons d'altres jocs que utilitzin aquest format

1.4. Quadre d’autovaloració

Donat el propòsit del projecte, el punt principal i on s’hi ha dedicat la majoria del temps serà l’aspecte tecnològic i les seves mecàniques, per tant, és on creiem que hi és tot el valor del projecte.

Estètica	0%
Narrativa	0%
Mecàniques	20%
Tecnologia	80%

Figura 2: Quadre d’autovaloració

2. Estudi de viabilitat

Abans de començar a dissenyar i implementar un videojoc, cal valorar si el projecte és viable i si es pot completar el desenvolupament.

En aquest apartat explorarem els recursos necessaris per a poder desenvolupar aquest videojoc en la seva totalitat. A més, es farà un estudi de mercat examinant els jocs disponibles en aquest sector i identificarem al públic objectiu i el perfil de jugador.

2.1. Recursos tècnics

Per a treballar en aquest projecte, cal un ordinador amb suficient potència per a poder treballar amb entorns de realitat virtual a una velocitat adequada.

L'ordinador que hem utilitzat té les següents especificacions:

- CPU: AMD Ryzen 7 2700X
- RAM: 32 GB DDR4
- GPU: AMD Vega 56
- SSD: Samsung 870 EVO 1TB

A més, s'han utilitzat unes ulleres de realitat virtual Oculus Quest 2, i per als pedals de la bateria es poden utilitzar qualsevols pedals USB.

Figura 3: Oculus Quest 2

Figura 4: Pedals USB

2.2 Recursos humans

Per a desenvolupar qualsevol videojoc, es necessita un equip multidisciplinari amb professionals que cobreixin tots els rols del projecte. Aquests rols són:

- Dissenyador: l'encarregat de definir el joc, els objectius i les mecàniques
- Programador: encarregat de la implementació del joc, desenvolupament del codi i dels aspectes tècnics
- Artista: encarregat de fer els components visuals del joc: textures, modelats 3D, concept art, interfície...

Com que aquest projecte ha estat realitzat per només una persona, li han correspost tots tres rols.

2.3. Recursos tecnològics

Pel desenvolupament d'aquest projecte, només s'ha utilitzat el hardware mencionat a l'apartat 2.1, Recursos tècnics.

En quant a software, s'ha utilitzat el motor de videojocs Unity, en la seva versió 2022.1.23f1, i els plugins OpenXR i XRInteractionToolkit.

Tota la programació s'ha realitzat amb C#, utilitzant Visual Studio Code i Notepad++ com a editors de text.

Els diferents modelats 3D s'han creat utilitzant Blender i certs elements de la interfície s'han creat amb Gimp.

2.4. Cost econòmic

En aquesta secció definirem el pressupost necessari per a poder crear la versió completa d'aquest videojoc.

En el cas dels recursos tècnics, necessitaríem que cadascun dels treballadors tingui accés a un ordinador prou potent per a poder compilar i executar el projecte, com l'equipament de realitat virtual i els pedals USB.

Pel que fa al software, tots els programes utilitzats són gratuïts.

En el cas dels recursos humans, necessitaríem una persona per cadascun dels perfils mencionats a l'apartat 2.2, Recursos humans.

Creiem que el temps de desenvolupament d'aquest joc seria d'aproximadament uns 2 anys, ja que amb una sola persona s'ha pogut programar un prototip en aproximadament 3 mesos, que conté un 20% del joc acabat.

Tenint en compte aquests costos i aquest marc de temps, el pressupost per a desenvolupar el joc seria el següent:

Concepte	**Cost**
Equips informàtics	5.000€
Salaris	36.000€ per any per treballador: 216.000€
Sistemes de realitat virtual	450€ per treballador: 1.350€
Total	**222.350€**

Figura 5: Quadre del pressupost

2.5. Estudi de mercat

Abans d'arriscar-se a invertir per a produir un videojoc, és necessari estudiar la possible viabilitat d'aquest al mercat. En aquest apartat, es mirarà de trobar la resposta a diferents qüestions sobre les possibilitats d'èxit econòmic del producte al mercat.

2.5.1. Cerques realitzades

Donat que aquest projecte intenta ser tant un videojoc com una eina d'aprenentatge per a aprendre a tocar un instrument, tindrem en compte com a competència tots aquells softwares que siguin una de les següents coses:

- Videojocs de ritme
- Software didàctic per a bateries electròniques

Tenint en compte això, s'han utilitzat les següents paraules i combinacions d'elles com a paraules clau per a fer les cerques:

- Rhythm Game
- Drums Game
- Drumming Game
- Drums learning
- E-drums software learning
- Electronic drums game

Les cerques s'han realitzat utilitzant Google, Steam i fòrums especialitzats en

bateries i bateries electròniques.

2.5.2 Resultats obtinguts

Els resultats obtinguts de les cerques s'han analitzat i hem decidit centrar-nos en els jocs de ritme més populars i en els softwares més semblants al projecte:

Beat Saber

Videojoc de ritme en realitat virtual que consisteix en destruir cubs amb unes espases làser al ritme de la música. Tot i no representar cap instrument, és un joc de ritme força popular que ha creat la seva pròpia forma única i innovadora d'interactuar amb la música en un videojoc.

Ha estat un gran referent per al desenvolupament del projecte en aspectes d'interfícies en realitat virtual.

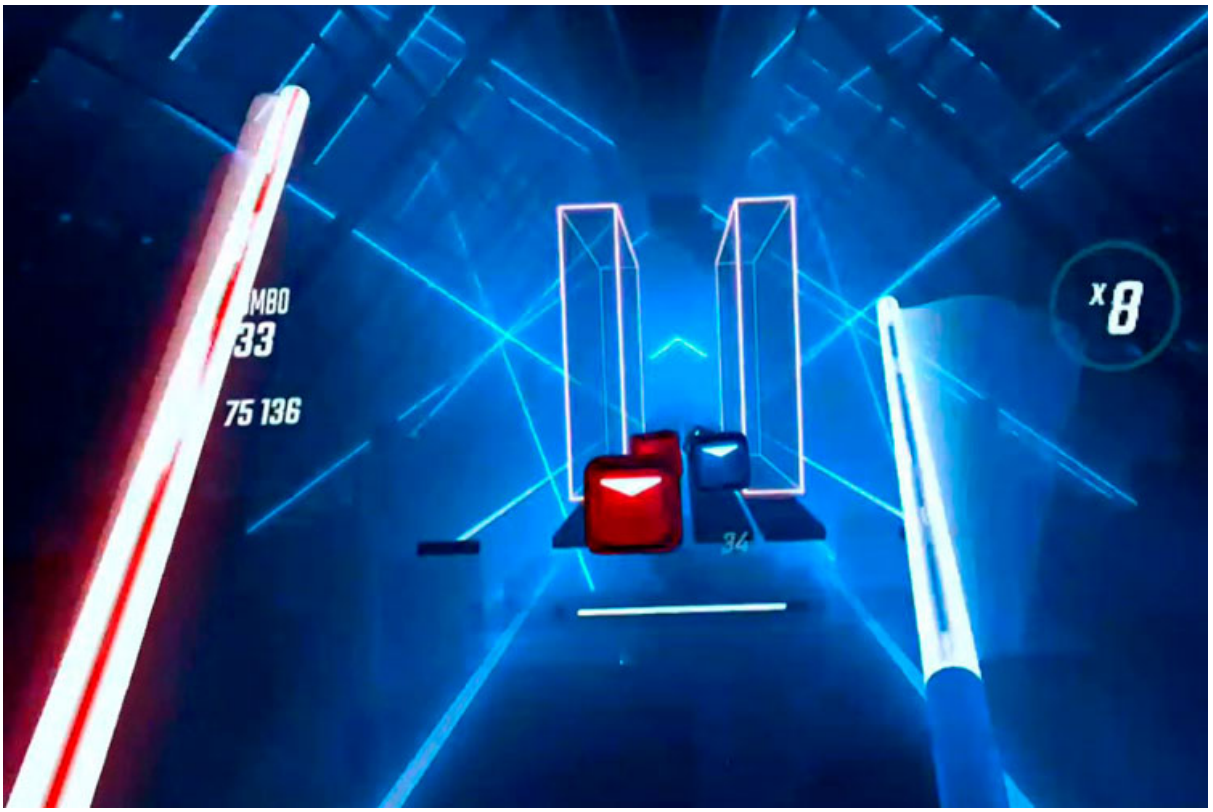


Figura 6: captura del joc Beat Saber

DTXMania/Drummania/GITADORA

Videojoc de ritme en el qual es pot jugar amb una bateria electrònica. DTXMania és el software per a poder jugar al videojoc a un ordinador personal, utilitzant una bateria electrònica, mentre que Drummania / GITADORA és el nom que el joc rep a les màquines recreatives. Tot i no ser molt popular a occident, al Japó és força comú trobar aquests jocs a llocs on hi ha màquines recreatives.

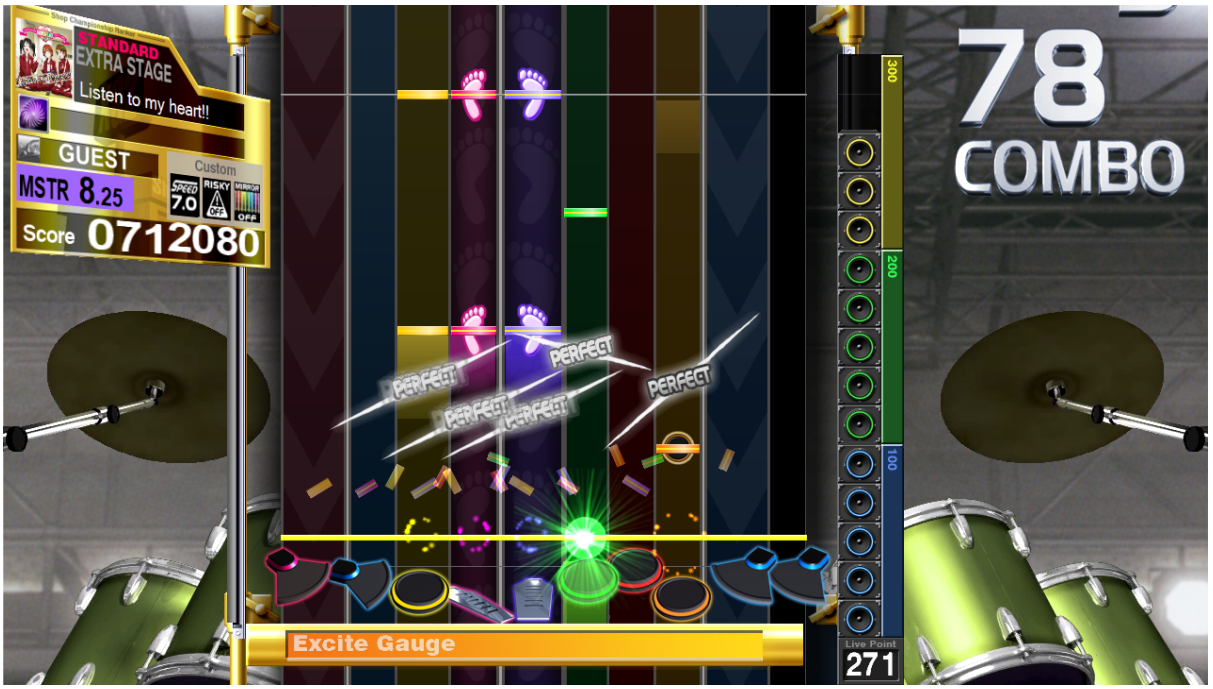


Figura 7 : Captura del joc DTXMania

Clone Hero

Videojoc de ritme en el qual es pot jugar amb una bateria simplificada feta específicament per al joc, o amb una bateria electrònica. Va ser desenvolupat per fans de la saga de videojocs de ritme Guitar Hero com a una alternativa Open Source del joc. Disposa tant d'una modalitat per tocar la bateria com d'una per tocar la guitarra.

Figura 8: Captura del joc Clone Hero

Drums Rock

Videojoc de ritme en realitat virtual que consisteix en tocar cançons de rock amb una bateria simplificada. El joc disposa de moltes cançons originals que es pot tocar juntament amb algunes cançons de grups de Rock i Metal força populars.

Tot i que aquest videojoc va sortir al mercat quan el projecte ja portava prou temps en desenvolupament, l'hem inclós a la llista donada les grans similituds amb el projecte.

Té limitacions com a eina per aprendre a tocar la bateria (no té suport per a pedals, selecció de cançons limitada i tancada).

Figura 9: Captura del joc Drums Rock

Taiko no tatsujin

Videojoc de ritme en el qual es poden jugar diferents cançons amb un comandament de consola o amb un tambor. A occident ara comença a ser una mica popular, gràcies als majors esforços publicitaris que s'han fet per a la versió per a la Nintendo Switch, però aquest joc es juga principalment al Japó, a màquines recreatives.

Figura 10: Captura del joc Taiko no Tatsujin

Osu

Un dels jocs de ritme més jugats i més populars tant a occident com a la resta del món, principalment per la gran comunitat que hi ha al voltant del joc (fer mapes, skins, tornejos…) i pels 4 modes de joc diferents que hi ha:

- Osu!Standard: fer clic a cercles amb el ratolí al ritme de la música
- Osu!Taiko: clon de Taiko no Tatsujin
- Osu!Mania: joc ritme de 4 a 7 tecles
- Osu!Catch: recollir fruites que cauen a ritme de la música

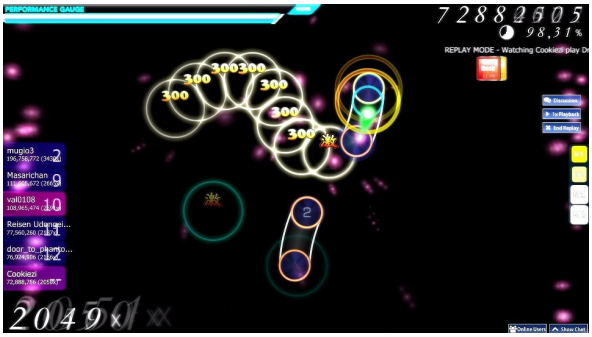


Figura 11: Captura d'Osu!Standard

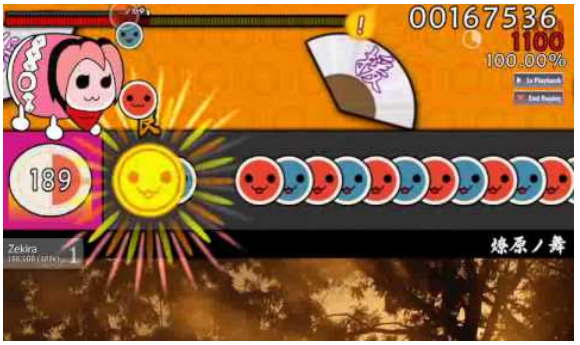


Figura 12: Captura d'Osu!Taiko

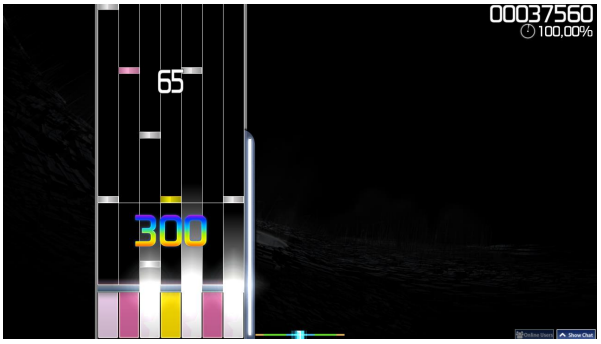


Figura 13: Captura d'Osu!Mania

Figura 14: Captura d'Osu!Catch

Melodics

Software amb diferents lliçons per a tocar la bateria i un mode estil “Rock band” on es pot tocar les cançons que hi ha a l’app amb una partitura dinàmica.

Figura 15: Captura del software Melodics

Drumsthesia

Software de codi obert que permet tocar les cançons que es vulgui amb format MIDI amb una bateria electrònica.

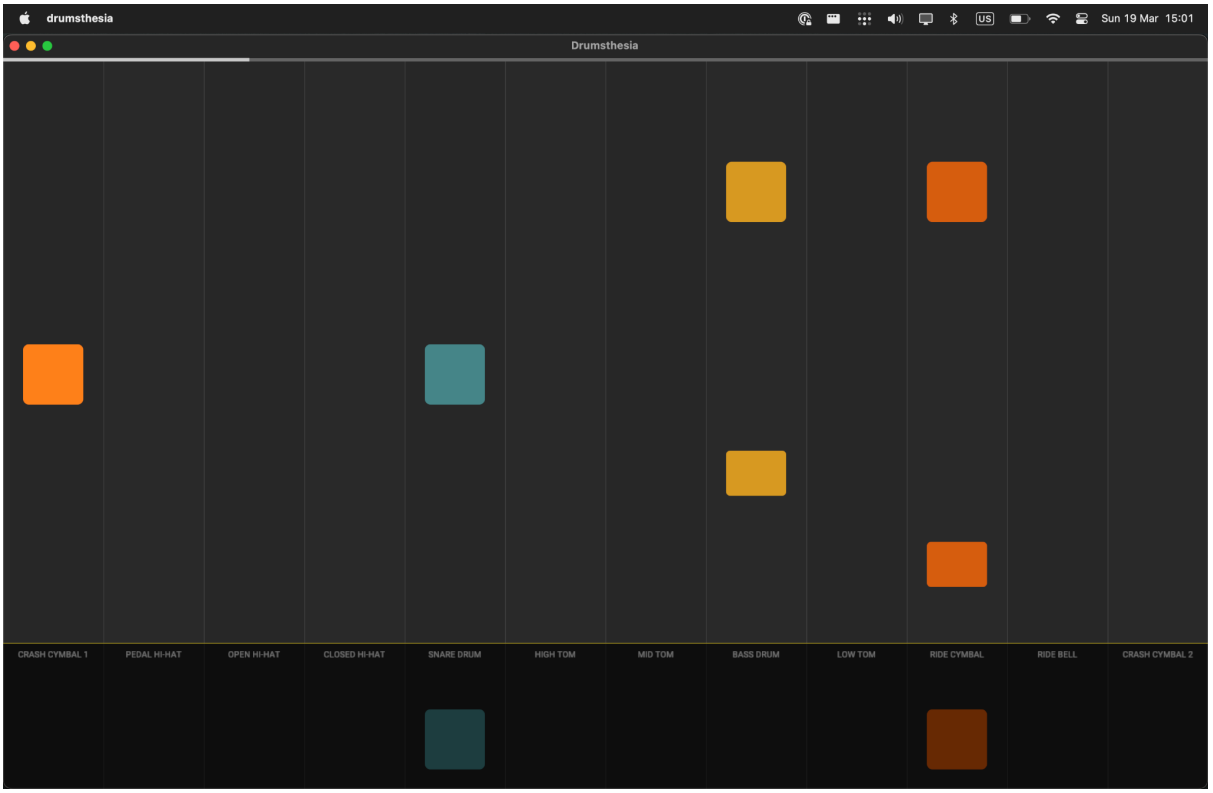


Figura 16: Captura del software Drumsthesia

2.5.3 Matriu de competitivitat

A partir de les cerques realitzades i tenint en compte els objectius del videojoc, s'han definit els següents requisits:

- Ha de ser un videojoc
- Ha de ser sobre tocar la bateria
- La bateria ha de ser completa
- Ha de permetre al jugador jugar les cançons que ell vulgui
- Ha de ser en realitat virtual (no requerir d'una bateria física)

La matriu de competitivitat resultant és la mostrada a la figura 17

	Videojoc	Bateria	Completa	Cançons de l'usuari	VR	Preu software	Preu software + hardware *
Beat Saber	Si	No	No	Si	Si	29,99€	379,99€
DTXMania	Si	Si	Si	Si	No	0€	700€
Clone hero	Si	Si	Si*	Si	No	0€	700€
Drums rock	Si	Si	No	Si	Si	19,99€	369,99€
Taiko no Tatsujin	Si	No	No	No	No	45€	159,99€
Osu	Si	No	No	Si	No	0€	0€
Melodics	No	Si	Si	No	No	159,99€ anuals	700€ + 159,99€ anuals
Drumsthesia	No	Si	Si	Si	No	0€	700€

Figura 17: Matriu de competitivitat

**** Els preus de referència per al hardware han sigut del dispositiu de realitat virtual Oculus Quest 2 i de la bateria electrònica Roland TD-07DMK. Aquests dos productes han estat triats per ser els més populars entre els usuaris d'aquests videojocs i programes.***

2.5.4 Conclusions de l'estudi de mercat

Com es pot veure amb l'estudi realitzat i la matriu de competitivitat (Figura 17), no hi ha cap joc al mercat amb totes les característiques del joc proposat.

La majoria de jocs sobre tocar la bateria requereixen d'una bateria real, i els que trobem en realitat virtual no intenten ser ferramentes d'aprenentatge per a un instrument real, per tant son o bé instruments més "abstractes" (com és el cas de Beat Saber) o bé versions simplificades dels instruments (com a Drums Rock).

2.6 Públic objectiu i perfil del jugador

El públic objectiu del videojoc proposat es pot classificar en dos grups:

- Gent que vol un videojoc de ritme per a realitat virtual
- Gent que vol una forma de practicar amb la bateria en casos que tenir-hi una de veritat o una electrònica no sigui viable, ja sigui per espai, portabilitat o cost.

També és important fixar-se amb l'edat d'aquest públic. Segons diferents estudis i articles, més del 70% d'usuaris de realitat virtual tenen entre 15 i 35 anys, així que orientarem el joc cap a la gent d'aquesta edat.

Per definir el perfil de jugador del joc proposat, s'analitzarà la tipologia de jugadors definida per Richard Burtle i que s'ha estudiat durant la carrera. Segons Burtle, es poden classificar els jugadors en 4 tipus:

- Achievers: tenen com a objectiu resoldre reptes. Obtenen plaer al resoldre situacions més complexes.
- Explorers: Els agrada explorar a fons el videojoc, descobrint els secrets i aprendre coses desconegudes del videojoc.
- Socializers: els hi agrada interactuar amb altres persones més que interactuar amb els altres elements del joc.
- Killers: els hi agrada competir contra altres jugadors.

Creiem que el perfil que més pot disfrutar del nostre joc és el de Achiever. L'estructura del joc es basa en oferir al jugador un entorn de pràctica per a poder millorar l'habilitat amb la bateria, permetent cada cop poder tocar cançons més difícils.

3. Planificació

3.1. Pla de treball

En aquest apartat es plantejaran les coses que es faran, així com el temps que es creu que fa falta per dur-les a terme.

3.2. Abast del projecte

Per poder fer una bona planificació, primer cal tenir clar el que es vol aconseguir amb el treball.

L'objectiu d'aquest treball és crear un prototip del joc proposat, implementant les mecàniques principals, deixant de banda elements més costosos que no són essencials per a poder avaluar si el concepte del videojoc és viable.

Els objectius que ens vam marcar per al prototip son els següents:

- Ha de ser en realitat virtual
- El jugador ha de poder tocar una bateria
- El joc ha de ser compatible amb pedals USB i altres mètodes d'entrada per a utilitzar els peus
- El jugador ha de poder carregar cançons amb el format DTX
- Ha de tenir un sistema de puntuació
- La bateria ha de ser "responsive"

3.3. Tasques planificades

Tasca	Temps estimat
Pluja d'idees	1 hora
Investigació	10 hores
Experimentació amb tecnologies VR	20 hores
Implementació del format DTX	50 hores
Construcció de la bateria en VR	5 hores
Sistema de puntuació	2 hores
Menus, interficies i Game Loop	10 hores
Pulir bugs, petites millores	10 hores
Total	108 hores

Figura 18: Tasques planificades

3.4 Diagrama de Gantt

A partir de les tasques planificades i el seu temps assignat, hem creat un diagrama de Gantt amb la distribució de les tasques durant els mesos del projecte.

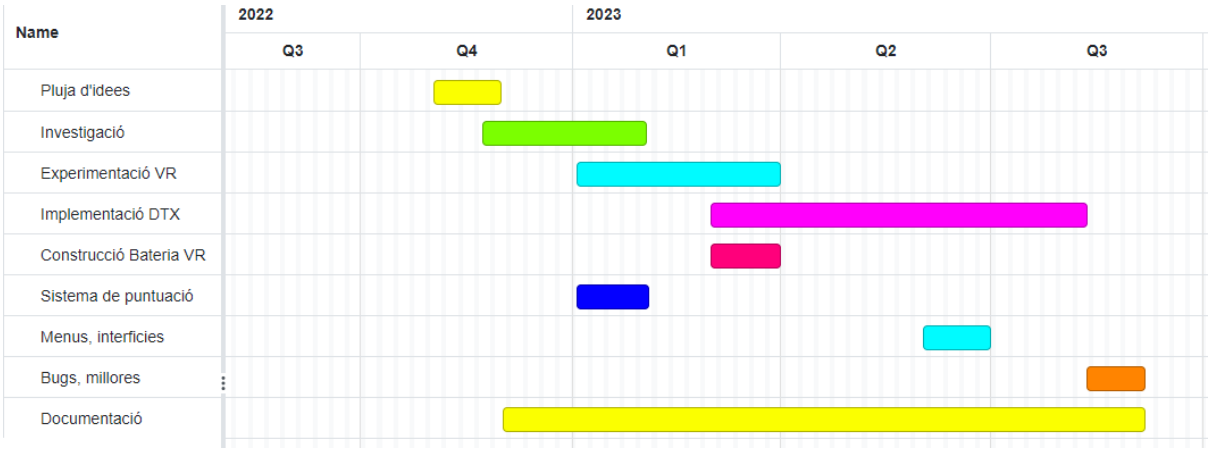


Figura 19: Diagrama de Gantt

4. Marc de treball i conceptes previs

En aquest capítol s'explicarà el marc de treball del projecte.

Explorarem els videojocs que han servit de referents per a fer aquest projecte i parlarem sobre les eines que hem utilitzat i les raons per les quals han estat escollides.

4.1 Referències

4.1.1 DTXMania

Figura 20: Imatge DTXMania

DTXMania és un videojoc que es pot jugar a ordinadors personals i consisteix en tocar cançons amb una bateria electrònica. Ha sigut la major font d'inspiració per a aquest projecte, que ben bé es podria resumir com "DTXMania, en realitat virtual".

El joc no disposa de història ni narrativa de cap tipus, és simplement un entorn on practicar amb la bateria amb un sistema de puntuació.

Un dels principals objectius del projecte ha estat fer-lo compatible amb el format de fitxers d'aquest videojoc (.dtx) per a poder jugar amb les mateixes cançons que hi ha per a aquest. Aquest format obert és el que ha permès que la comunitat pugui introduir cançons que no hi eren al joc originalment i poder crear diferents eines que permeten, fins i tot, crear-los automàticament.

4.2 Motor de videojoc: Unity 3D

Figura 21: Logo Unity

El prototip s'ha implementat amb el motor Unity 3D.
Els motius per a triar aquest motor han estat els següents:

- Familiaritat prèvia amb l'entorn de desenvolupament
- Excel·lent documentació
- Experiència prèvia amb els plugins de realitat virtual
- Existència de més documentació i tutorials online (sobretot per a coses amb realitat virtual)
- Preu (gratuït)

4.3 Eines de disseny

4.3.1 Blender

Figura 22: Logo Blender

Els diferents modelats 3D dels elements de la bateria s'han realitzat amb Blender.
Els motius per a triar aquest programa han estat els següents:

- Experiència prèvia durant la carrera
- Preu (gratuït)

4.3.2 Gimp

Figura 23: Logo Gimp

Els diferents elements 2D de la interfície han estat creats o editats amb GIMP.
Els motius per a triar aquest programa han estat els següents:

- Experiència prèvia durant la carrera
- Preu (gratuït)

4.4 Vocabulari específic utilitzat

- Tempo / BPM: Velocitat a la qual s'interpreta una cançó
- XROrigin: Objecte que conté l'origen de coordenades en realitat virtual
- Chart: En videojocs de ritme, el fitxer amb el conjunt de notes que formen una cançó
- Charting: En videojocs de ritme, crear una chart
- Reading / llegir: En videojocs de ritme, mirar la chart per a saber quines notes o quins instruments tocar

5. Disseny del videojoc

En aquest apartat veurem tots els elements que formen el videojoc i com interactuen entre ells per aconseguir els objectius del projecte.

5.1 Maquinari

Donat que aquest és un treball en realitat virtual, hem considerat necessari explicar el hardware utilitzat i els motius pels quals ha estat triat.

5.1.1 Ulleres VR

El projecte ha estat realitzat utilitzant unes ulleres de realitat virtual Oculus Quest 2. Aquestes ulleres disposen d'una sèrie de qualitats que són idònies tant per al desenvolupament d'aquest videojoc, com per a jugar al producte final:

- Disposa de dos comandaments (un per a cada mà)
- Utilitza seguiment intern
    - Mitjançant càmeres a les ulleres i sensors als comandaments, evita la necessitat que tenen altres sistemes de realitat virtual de tenir sensors en diferents llocs de l'habitació
- Donat que és el sistema més utilitzat, hi ha molta més documentació online
- Preu assequible (349,99€)
- Molt portable gràcies a la seva mida i pes tant de les ulleres com dels comandaments

5.1.2 Pedals USB

El projecte ha estat realitzat utilitzant uns pedals USB de la marca *Staright*.
Han estat triats per la funcionalitat i el baix cost.

5.2. Llibreries i Plugins

5.2.1 OpenXR

OpenXR és un estàndard obert que facilita l'accés a les tecnologies de VR (Realitat Virtual) i AR (Realitat Augmentada), conegudes en conjunt com a XR. Unity disposa d'una implementació d'aquest estàndard, i l'hem utilitzat en aquest projecte ja que dóna, de forma pràcticament automàtica, compatibilitat amb la majoria de ulleres de VR i AR.

5.2.2 XR Interaction Toolkit

Aquest paquet d'Unity ofereix un sistema d'interacció basat en components per a crear experiències de VR i AR. El nucli d'aquest sistema consisteix en una sèrie de components Interactor i Interactable base, i un Interaction Manager que permet que els components d'aquests dos tipus interactuïn. També conté components que es poden utilitzar per al moviment del jugador i efectes visuals.

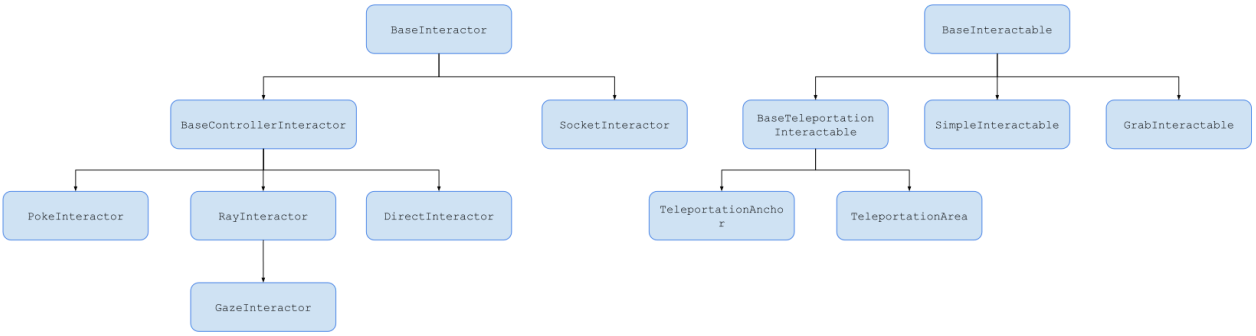


Figura 24: **Diagrama de classes dels components de XR Interacion Toolkit**

A més, el paquet ofereix components per a facilitar les següents tasques:

- Input multiplataforma
    - Funciona amb comandaments de Meta Quest (Oculus), OpenXR, Windows Mixed Reality i d'altres
- Comportament bàsic al posar la mà virtual a sobre d'un objecte, agafar-lo i deixar-lo
- Haptic feedback mitjançant els comandaments
- Feedback visual per indicar interaccions possibles i actives
- Canvas bàsic per a fer interfícies interactuables amb comandaments XR

5.2.3 XR Hands

Aquest paquet conté els models 3D i les animacions d'unes mans per a ser utilitzades com a representació de les mans físiques en un món virtual. Veure figura 25.

Figura 25: **Imatge de les *XR Hands***

5.2.4 VR Button

Aquest paquet conté un exemple de com fer un botó en realitat virtual.
Tot i que aquest botó no ha estat utilitzat, s'ha utilitzat com a referència de com fer elements interactuables mitjançant física amb realitat virtual.

5.2.5 Runtime File Browser

Component que permet carregar un fitxer a l'ordinador del jugador en temps d'execució. Aquest component s'utilitza per a poder carregar fitxers .dtx amb la informació de la cançó que vol interpretar el jugador.

Originalment, va ser desenvolupat per a aplicacions amb ratolí i teclat, però amb un parell de canvis al component es pot utilitzar en entorns de realitat virtual. Veure figura 26.

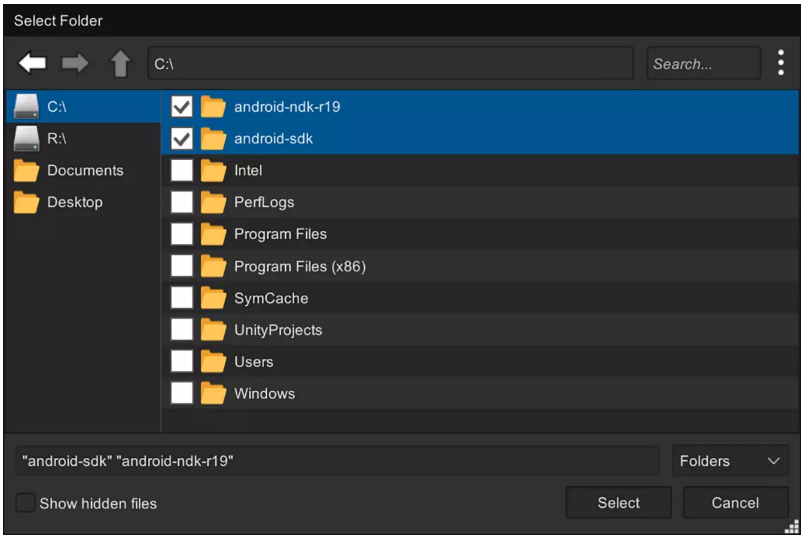


Figura 26: Imatge del component Runtime File Browser

5.3 Mecàniques

En aquest apartat explorarem com funciona la bateria virtual i tots els elements amb els quals el jugador pot interactuar i com interactuen entre ells.

5.3.1 Espai de joc

El jugador té un únic escenari que conté el XROrigin i els diferents elements de la interfície. Durant el transcurs del joc, els diferents elements de la interfície i la bateria van apareixent i desapareixent segons sigui necessari. Veure figures 27 i 28.

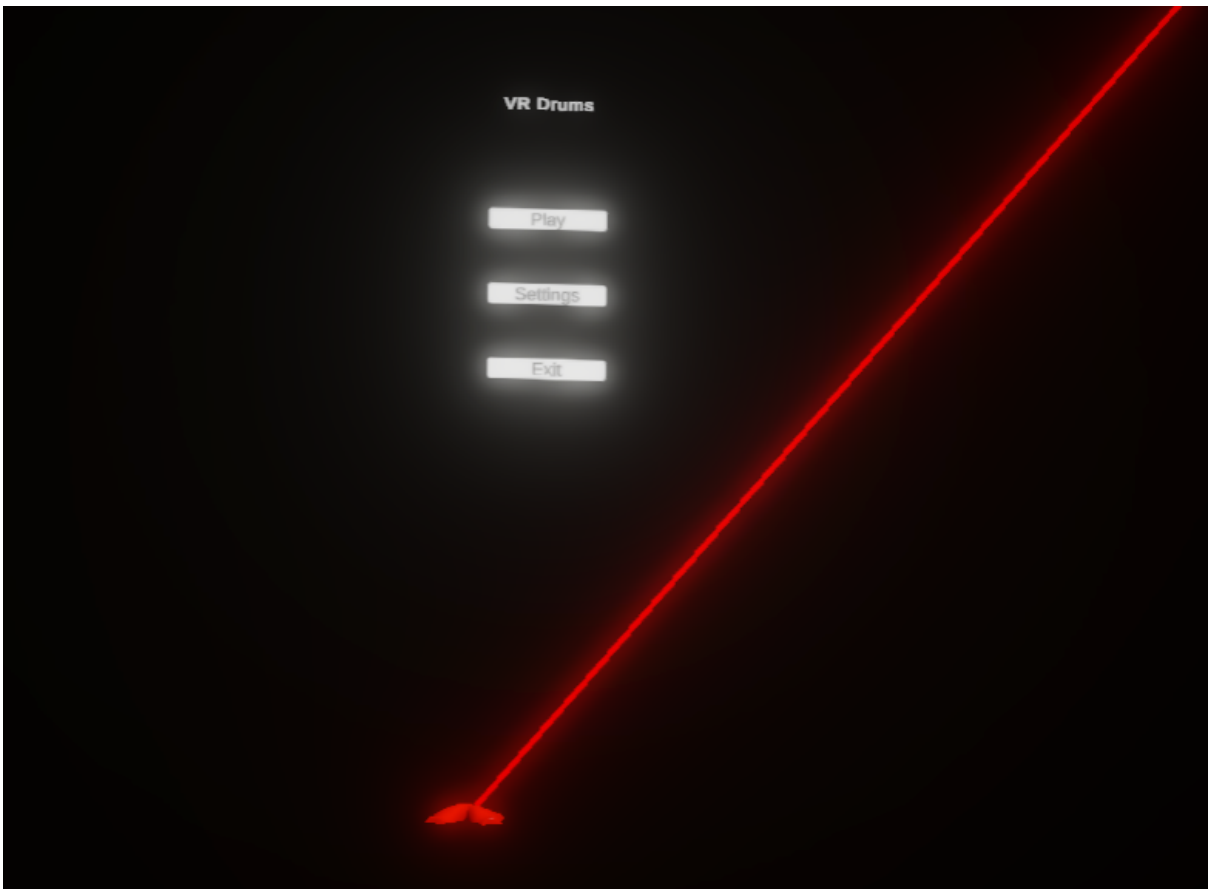


Figura 27: Imatge de l'estat inicial de l'espai de joc

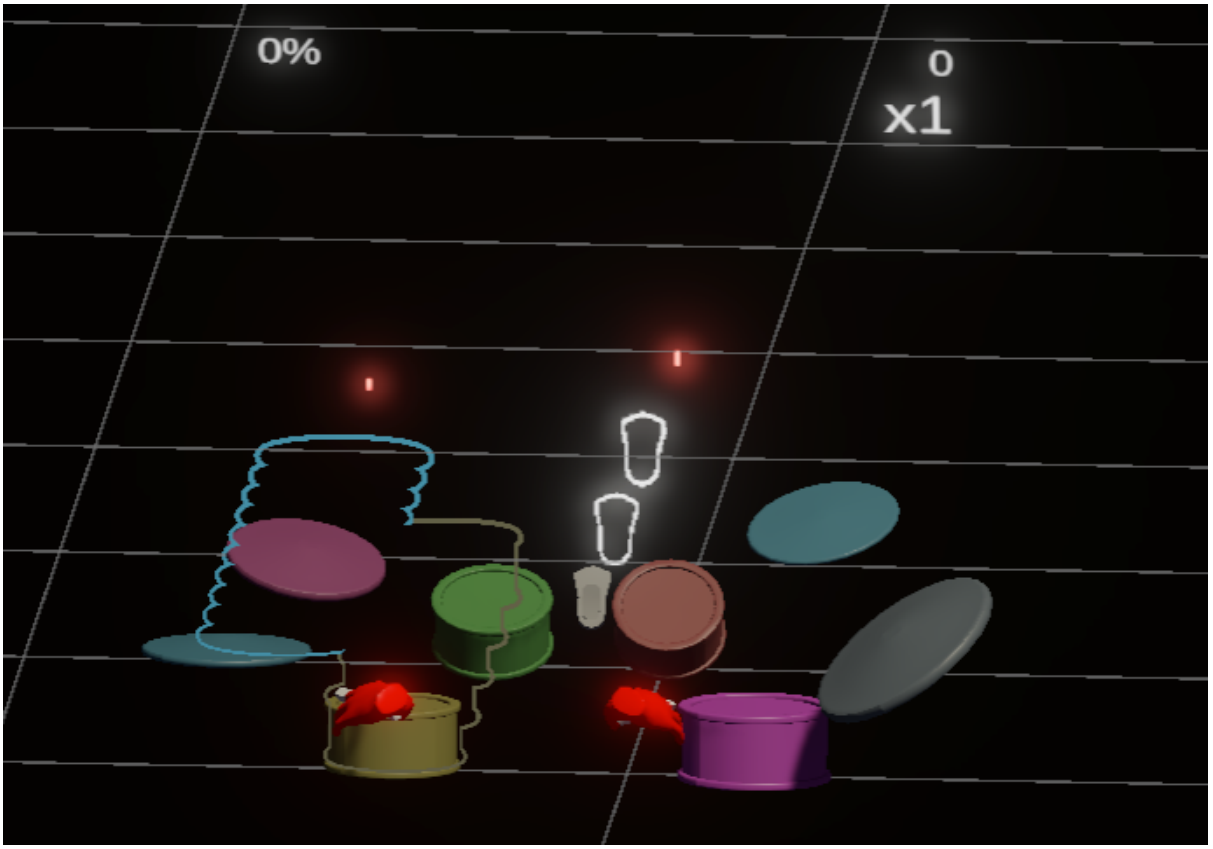


Figura 28: imatge de l'espai de joc durant una partida

5.3.2 Mans

El jugador disposa de les seves mans per interactuar dintre del món virtual. Aquestes mans tenen diferents animacions al prémer diferents botons del comandament per a simular que el jugador agafa un objecte. Aquesta funcionalitat s'utilitza per a poder agafar les baquetes al començar una partida. Quan el jugador és a un menú, d'aquestes mans surten dos raigs per a poder interactuar amb els elements de la interfície. Veure figura 29.

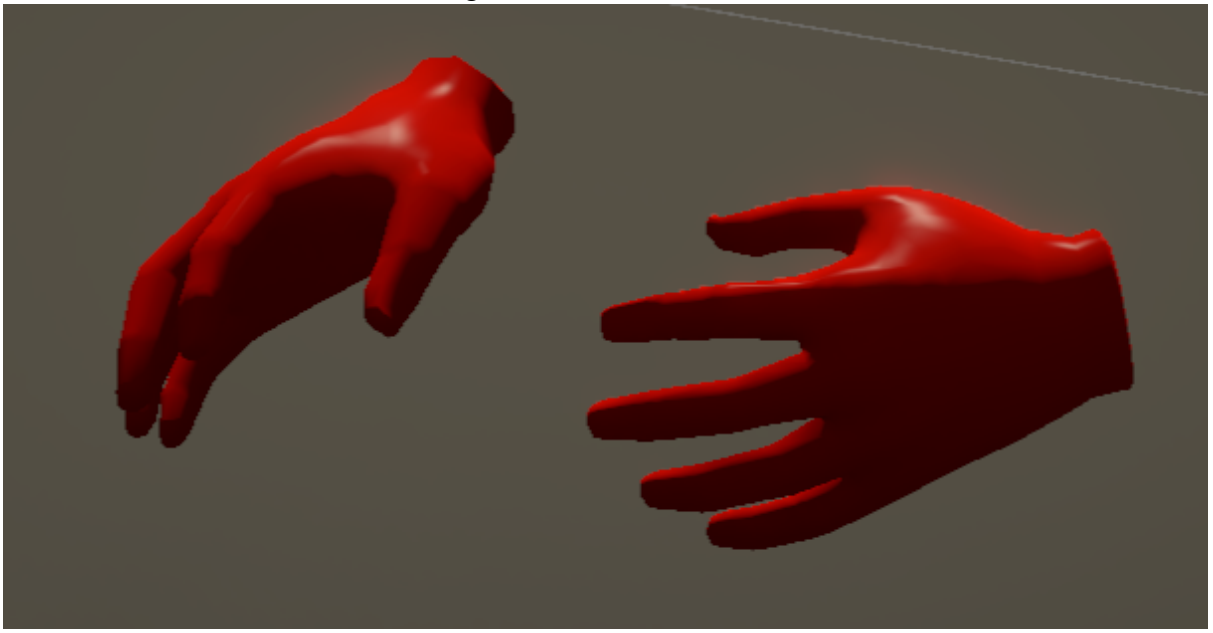


Figura 29: Imatge del component Mans

5.3.3 Baquetes

Quan el jugador tria una cançó per a interpretar, apareixen dues baquetes al món del joc i la partida no comença fins que el jugador no agafi les dues baquetes. Aquestes baquetes són les que permeten que el jugador pugui interactuar amb els diferents elements de la bateria. Veure figura 30.

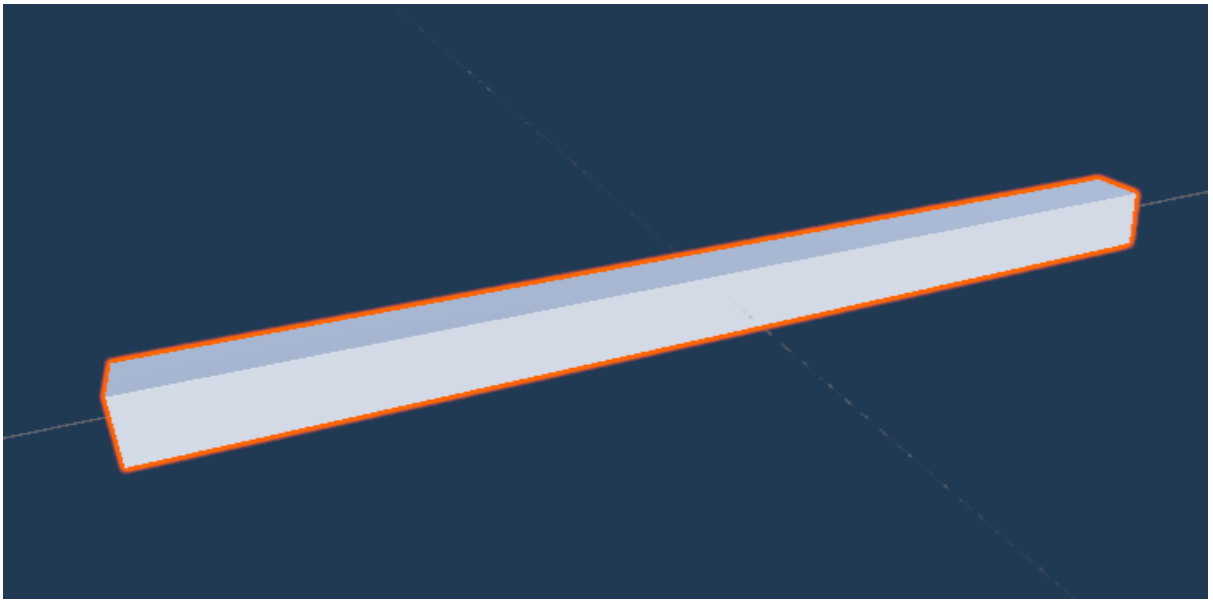


Figura 30: Imatge de les baquetes

5.3.4 Bateria

Quan el jugador tria una cançó per a interpretar, també apareix davant seu la bateria, compresa de 9 instruments: 3 plats, 3 tom-toms, un bombo, un xarleston i la caixa. Quan el jugador toca un d'aquests instruments amb una de les baquetes, el joc hi registra un impacte i reprodueix el so d'aquell element, juntament amb efectes visuals i vibració al comandament. Veure figura 31.

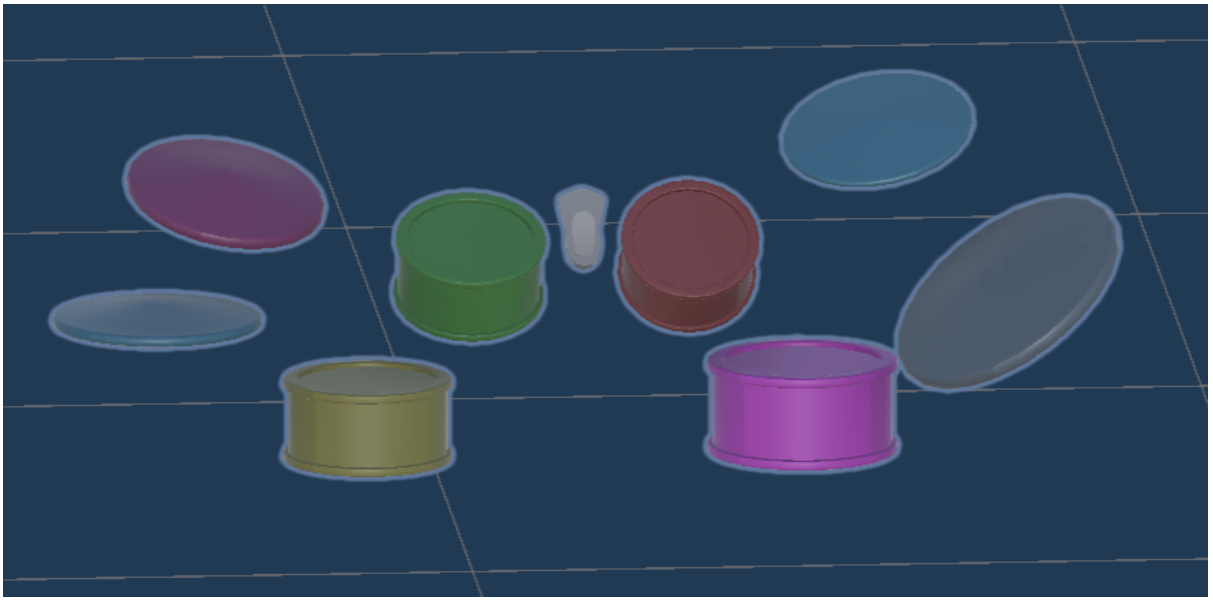


Figura 31: Imatge de la bateria

5.3.5 Chart

Component que mostra quin o quins instruments ha de tocar el jugador en cada moment. Consisteix en una sèrie de "canals" que contenen totes les notes. Aquestes es mouen cap al jugador i aquest ha de tocar l'instrument quan la nota hi sigui damunt d'un punt en concret anomenat "Judgment line". Veure figura 32.

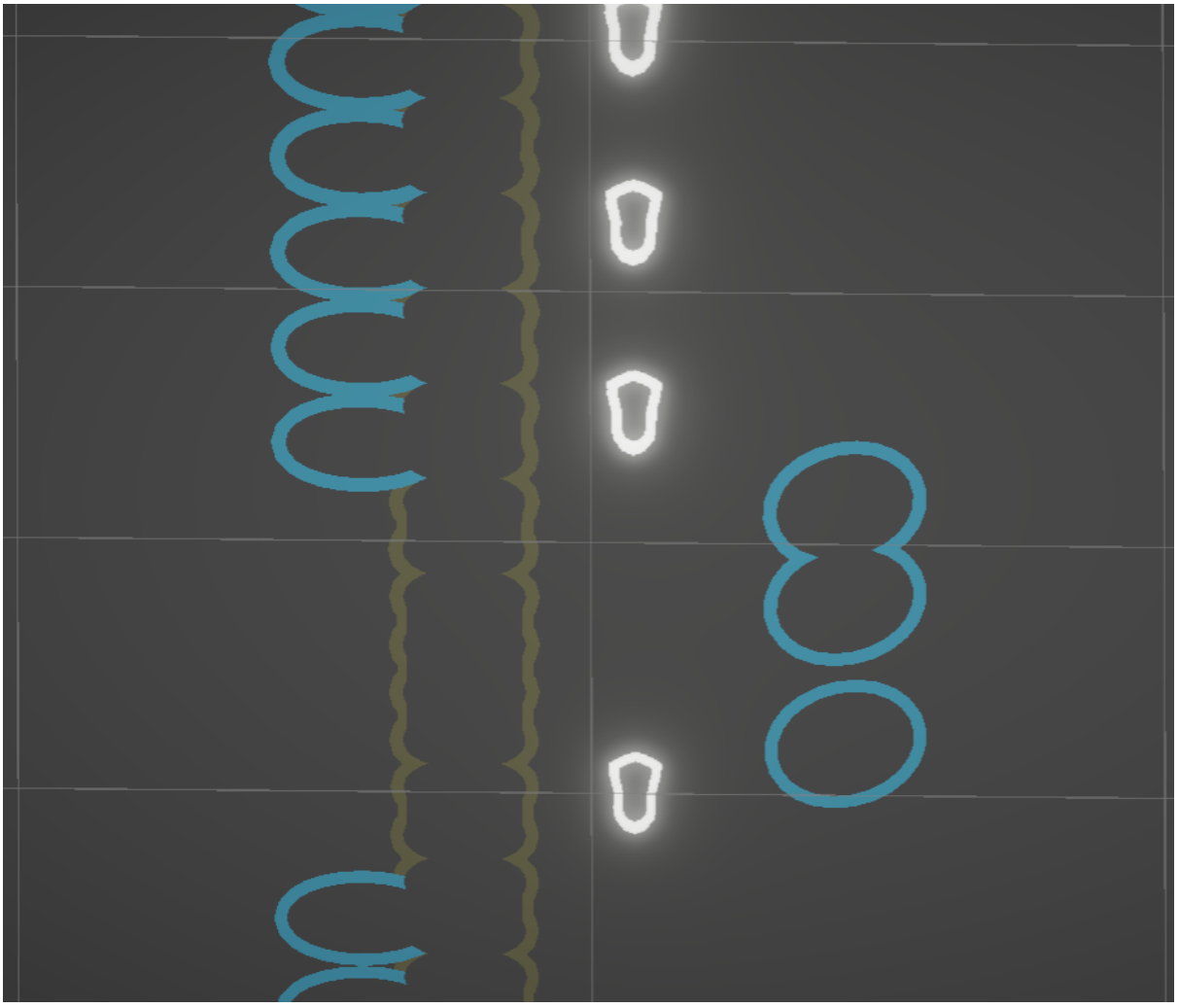


Figura 32: Imatge d'una secció d'una Chart

5.3.6 Judgement Line

Volem recompensar al jugador per tocar les notes en el moment precís, seguint el ritme de la cançó. Per a incentivar això, el jugador rebrà més punts com més a prop hi sigui la nota de la Judgement line. Si el jugador ha tocat la nota una mica lluny de la Judgement line, també guanya punts, però molts menys, i si la toca molt a lluny, conta com una nota errònia. També existeix la DeathLine, si una nota hi arriba s'entén que ja és massa tard per a que el jugador la pugui tocar i es converteix en una nota fallida. Veure figura 33.

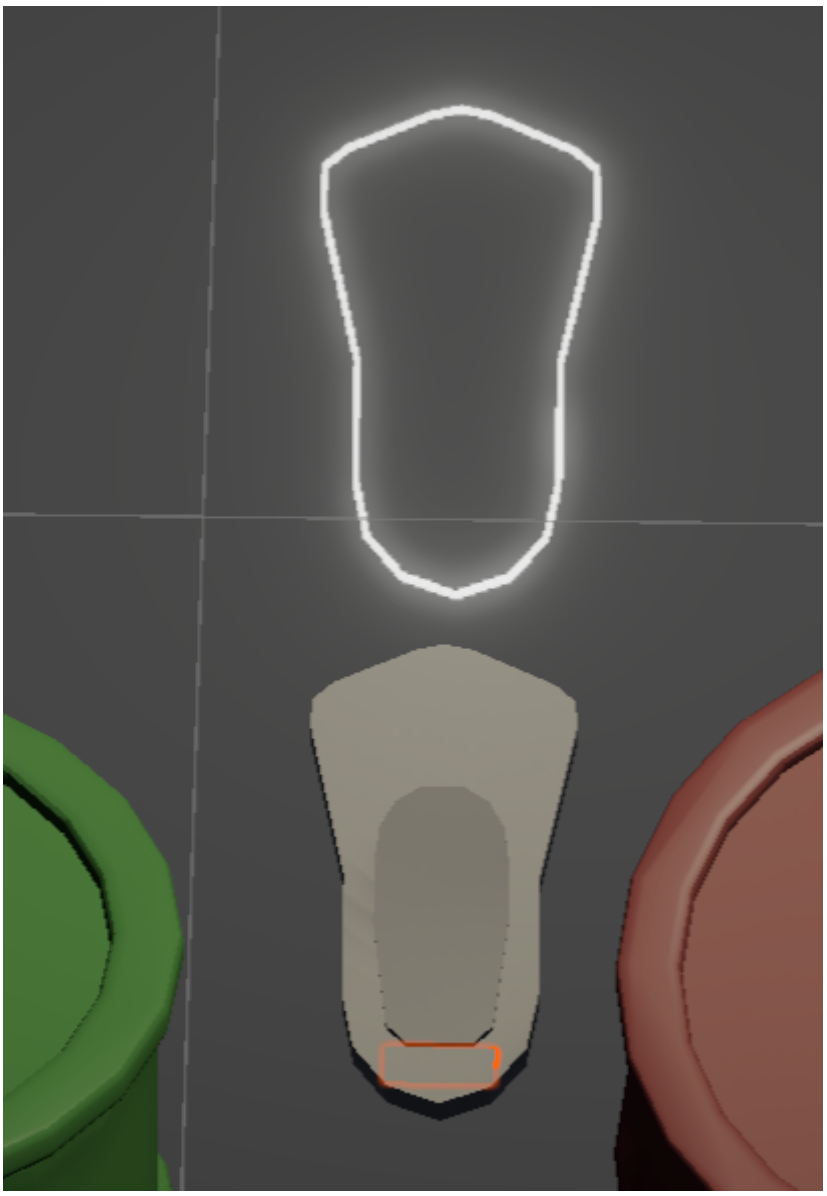


Figura 33: Imatge de la Judgement line (Rectangle taronja)

5.3.7 Puntuació

L'objectiu del jugador és aconseguir el major nombre de punts possible. Els punts funcionen de la següent manera:

- Una nota "Perfecta" dona 300 punts
- Una nota "Bona" dona 150 punts
- Hi ha un comptador de "combo", quantes notes seguides el jugador ha tocat sense fallar
- A l'arribar a suficients punts de combo, el jugador és recompensat amb un multiplicador, que fa que les següents notes que encerti donen més punts
- Si el jugador falla una nota, el combo passa a ser 0 i el multiplicador passa a ser 1

A més, el joc també informa al jugador la seva precisió, quin percentatge de notes ha aconseguit tocar.

5.4 Interficies

5.4.1 Menú principal

Figura 34: imatge del menú principal

És el menú que el jugador es troba quan s'obre l'aplicació. En aquest menú (veure Figura 34) el jugador pot començar una partida, obrir el menú d'opcions o tancar el joc.

5.4.2 Chart

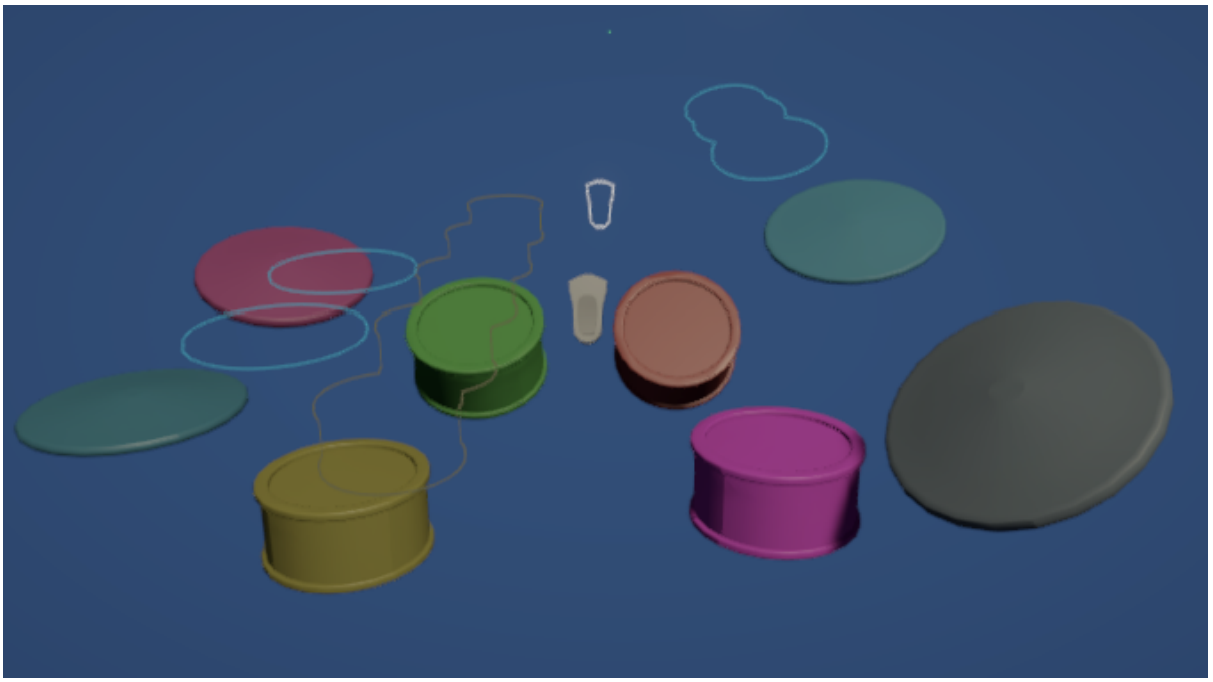


Figura 35: Imatge del chart dintre de la partida

Després que el jugador hagi triat la cançó que vol interpretar, el joc crea elements dintre del món del joc, amb totes les notes que el jugador ha d'interpretar. Aquestes tenen la mateixa forma i color que l'instrument que representen de la bateria. Durant la duració de la partida, aquestes es mouen cap a la Judgement Line. Veure figura 35.

5.4.3 Puntuació dintre de la partida

Figura 36: Imatge del component de la puntuació

Mentre el jugador es troba dintre d'una partida, al món del joc hi apareix un component amb la informació rellevant sobre la puntuació del jugador. Veure figura 36.

5.4.4 Pantalla de puntuació final

Figura 37: Imatge de la pantalla de puntuació final

Un cop el jugador ha acabat d'interpretar una cançó, apareix una pantalla resum amb informació sobre la cançó que s'acaba d'interpretar i la puntuació obtinguda. Veure figura 37.

5.5 Feedback

En aquest apartat veurem els diferents elements que s'han implementat al joc per a poder millorar la "Responsiveness" de la bateria amb l'objectiu d'incrementar la sensació d'immersió del videojoc.

5.5.1 Sound chips

En el moment que el jugador activi un dels instruments, el joc reprodueix el so d'aquell instrument. Aquest so varia depenent de la cançó que el jugador estigui interpretant per tal que s'hi assembli al so de la bateria a la gravació original de la cançó.

5.5.2 Vibració

En el moment que el jugador activi un dels instruments, el joc fa que vibri el

comandament que ha utilitzat el jugador per a fer-ho durant una quantitat petita de temps. La Intensitat d'aquesta vibració dependrà de la velocitat amb la qual el jugador hagi activat l'instrument.

5.6 Reptes i objectius

L'objectiu principal del joc és poder interpretar cançons cada cop més difícils i interpretar-les millor. El repte per al jugador és sempre el mateix: poder reaccionar al chart i tocar els instruments que toca al ritme de la música.

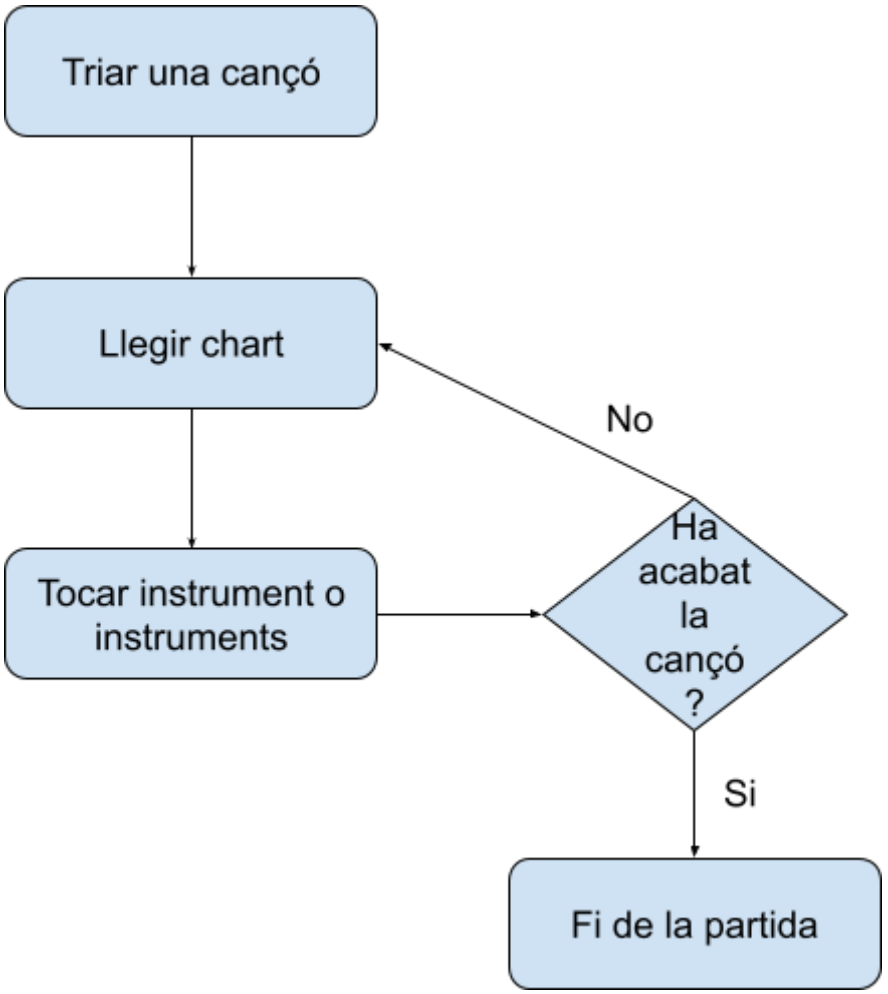


Figura 38: Diagrama de reptes

5.7 Definició d’accions

Per a navegar les interfícies: el jugador compta amb les següents accions

- Apuntar: apuntar amb el raig que surt de la mà dintre del món virtual.
- Seleccionar: prémer el botó de seleccionar al comandament

Dintre d’una partida, el jugador compta amb les següents accions:

- Agafar les baquetes
- Prémer el pedal dret per a tocar el bombo
- Prémer el pedal esquerre per a separar els plats del Xarleston
- Colpejar un instrument amb les baquetes per a activar-lo

5.8 Flowchart

Hem creat un *flowchart* per a representar com es desenvolupa el joc i el pas d’una secció a una altra.

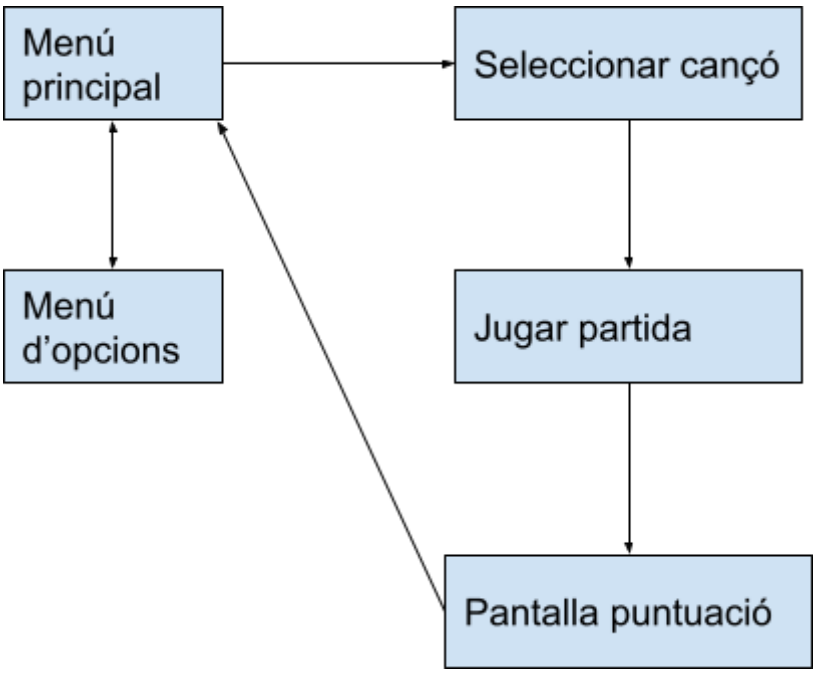


Figura 39: Flowchart d’una sessió de joc

Des del menú principal on s’inicia el joc, el jugador pot visitar el menú d’opcions o triar la cançó per a començar a jugar. Un cop triada la cançó, la partida comença. Un cop la partida s’ha acabat, se li mostra al jugador la pantalla amb la puntuació i aquest pot tornar al menú principal.

6. Implementació

En aquest apartat explicarem com s'han implementat tots els aspectes del joc. També explicarem les tècniques utilitzades i els problemes que s'han trobat durant el desenvolupament del prototip i com s'han resolt.

6.1 Realitat virtual

6.1.1 Escena principal

Per a poder utilitzar ulleres de realitat virtual dintre d'un joc a Unity i poder interactuar amb els objectes del món, l'escena ha de tenir tota una sèrie de components dels plugins mencionats als Apartats 5.2.1 i 5.2.2. Veure Figura 40.

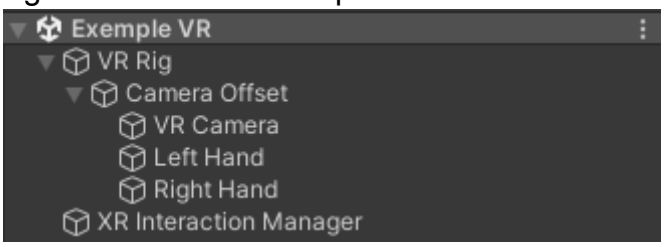


Figura 40: Components necessaris per a VR

6.1.2 Mans

Per defecte, els comandaments del jugador no tenen representació visual dintre d'Unity. Per a resoldre això, els hem donat el de XRHands (Apartat 5.2.3). Veure Figura 41.

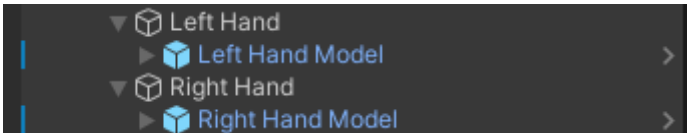


Figura 41: Component de les Mans amb el seu model

6.1.3 Interactors

Per a que els objectes puguin interactuar amb les mans, cal que aquestes tinguin un Interactor. Aquest script (Figura 42) és el que s'encarrega de gestionar les interaccions i el volem principalment per a que el jugador pugui agafar les baquetes, ja que és l'únic objecte del joc que es pot agafar. La mà també necessita un collider per a poder detectar els objectes que són prou a prop com per a poder agafar-los (Figura 43)

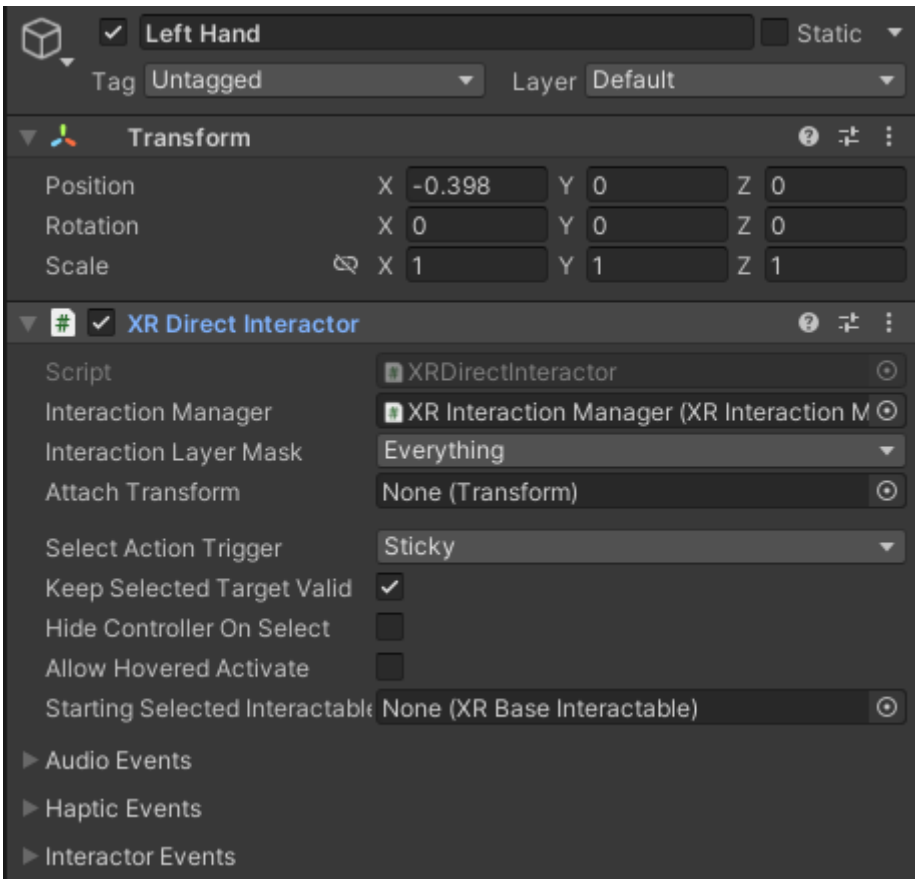


Figura 42: Script XRDirectInteractor dintre de la mà esquerra

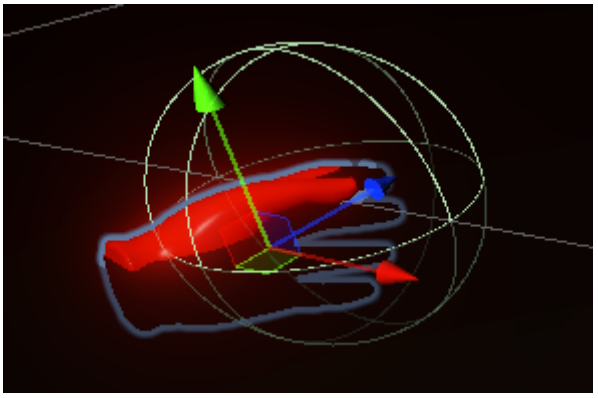


Figura 43: Collider per a detectar objectes propers

6.2 Simulació i format DTX

6.2.1 Introducció format DTX

El format DTX és el format de fitxer que utilitzen videojocs com DTXMania per a guardar informació sobre les notes que hi ha a una cançó. Aquest format funciona de la següent manera:

- El fitxer comença amb diferents **headers** amb informació sobre la cançó (nom, artista, any…) (Figura 44, primera meitat)
    - Alguns d'aquests headers indiquen diferents fitxers que s'han de carregar per a poder reproduir sons durant la partida (la cançó de fons,

sons dels instruments, sons d'ambient). Aquests sons s'anomenen **chips**.
    - Alguns d'aquests headers ens indiquen altra informació útil sobre la cançó, com els diferents BPM que pot arribar a tenir.
    - Tant les **chips** com els **BPM** es guarden numerats en **base36 (0-Z)**.
    - També poden contenir informació per a diferents efectes visuals o vídeos de fons, però han estat ignorats per a aquest treball
- Un cop s'acaben els headers, comencen els **descriptors d'objectes** (Figura 44, segona meitat i Figura 45)
    - Aquests descriptors indiquen. per a cada compàs de la cançó, quins instruments es toquen en cada moment
    - Consta de dues parts: el "**Descriptor**" (**nombre del compàs + canal**) i la **Object List**, una sèrie de **parelles de valors en base36**, que contenen els **objectes** que hi ha al **compàs separats de manera uniforme**.
    - Hi ha un màxim de 3599 compassos
    - També conté els moments en els que es canvia el BPM de la cançó, així com el compàs d'aquesta (4/4, 3/4 …)
    - Els objectes estan assignats a un **Canal**. Depenent de a quin canal pertany un objecte, sabem de quin tipus és. Els canals que ens interessen són:
        - 01 per a la cançó de fons
        - 02 per als canvis de compàs
        - 03 i 08 per als canvis de BPM
        - 11 - 1A per als diferents instruments

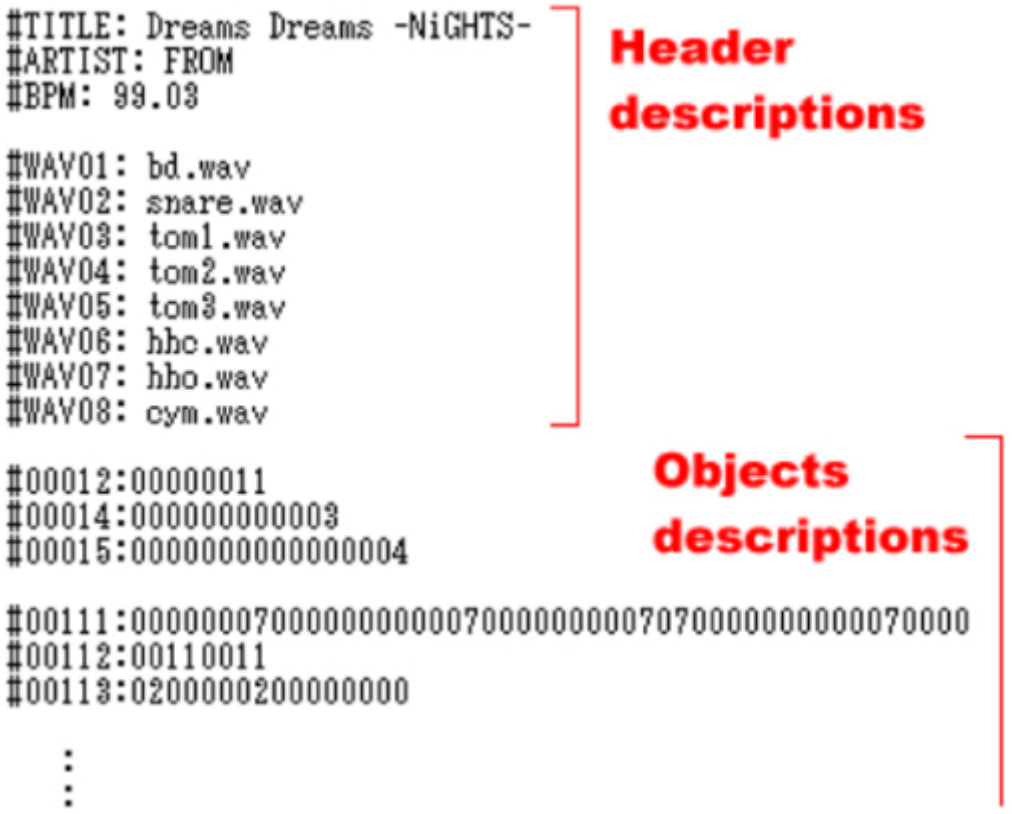


Figura 44: Exemple de fitxer DTX

```
"#00111 0101010101010101" -> "#00111 01 01 01 01 01 01 01 01"
```

Figura 45: Exemple d'Object descriptor

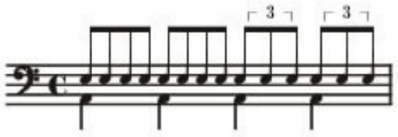
Aquesta instrucció es tradueix com a:

- Al compàs 001,
- Al canal 11,
- Col·locar 8 objectes 01 distribuïts de manera uniforme al compàs

6.2.2 Lectura del fitxer

Tant la lectura del fitxer com la simulació de la cançó es fan al script **DTXConverter**. Per a llegir el fitxer, executem la funció parseFile. Aquesta funció examina el fitxer que es troba a la ruta guardada a la variable path. Un cop llegit el contingut, l'examina **línia a línia**. Per a cada línia guardem els seus continguts en una variable del tipus String i la netegem. Busquem el caràcter ";" i si existeix, esborrem aquest caràcter i tot el que hi ha després, ja que en aquest format s'utilitza per a escriure comentaris als fitxers. Un cop fet això, examinem com comença aquesta línia i decidim que fer en cada cas:

- Si no comença amb un **#**, significa que és una línea errónea i la ignorem.
- Si comença amb BPM, guardem el valor que hi ha a continuació a la llista de BPMs a la posició que indica.
- Si comença amb Title o Artist, guardem aquesta informació per a mostrar-la a la pantalla de resultats.
- Si comença amb WAV, ens guardem el valor a la llista de chips.
- Si comença amb VOLUME, ens guardem el valor a la llista de volums.
- Si comença amb 5 digits decimals, sabem que és un **object descriptor**:
    - Disposem d'una variable anomenada **map**, que consisteix en un **diccionari** on la clau és la **combinació d'un canal i un nombre de compàs**.
    - Dintre d'aquest diccionari, a cada entrada hi ha una **llista de Strings**. Dintre d'aquesta llista hi **guardem els valors de la línia actual**. El motiu pel qual això és necessari és perquè **la mateixa combinació de canal + compàs pot aparèixer més d'un cop** i s'han de **conservar tots els valors** (figura 46).




```
e.g.
  #00112 02020202_02020202_00000000_00000000   -> first half parts (sixteenth notes)
  #00112 000000___000000___020202___020202     -> latter half parts (triplet notes)
  #00113 03_______03_______03_______03
```

Figura 46: Exemple de més d'un Object Descriptor per combinació de compàs+canal

Un cop acaba aquest procés disposem de:

- Tots els BPMs de la cançó a la variable BPMs
- Els noms de tots els fitxers de so a la variable chips

- Els nivells de volum d'aquests chips a la variable volumes
- La variable map amb totes les notes de la cançó, tots els moments on es canvia de BPM, els moments on es reprodueix la cançó de fons i els diferents canvis de compàs.

Veure Figura 47.

```
void parseFile()
{
    if (path.Length != 0)
    {
        fileContent = File.ReadAllLines(path);
        foreach (string line in fileContent)
        {
            string lineClean = line;
            // Si conté un comentari, l'eliminem
            int index = lineClean.LastIndexOf(";");
            if (index >= 0) lineClean = lineClean.Substring(0, index);
            if (lineClean.Length == 0) continue;
            // Si no comença amb un #, es una linea incorrecta
            if (lineClean[0] == '#')
            {
                // Si comença amb 5 digits, es un objecte
                if (Regex.IsMatch(lineClean, "^#\\d{5}"))
                {
                    string key = lineClean.Substring(1, 5);
                    string objects = lineClean.Substring(lineClean.LastIndexOf(':') + 1).TrimEnd().TrimStart();
                    if (!map.ContainsKey(key))
                    {
                        map.Add(key, new List<string>());
                    }
                    map[key].Add(objects);
                    if (int.Parse(key.Substring(0, 3)) > lastMeasure) lastMeasure = int.Parse(key.Substring(0, 3));

                }
                else if (lineClean.StartsWith("#TITLE")) songName = lineClean.Substring(lineClean.LastIndexOf(':') + 1).TrimEnd().TrimStart();
                else if (lineClean.StartsWith("#ARTIST")) artist = lineClean.Substring(lineClean.LastIndexOf(':') + 1).TrimEnd().TrimStart();
                else if (lineClean.StartsWith("#BPM"))
                {
                    // Ignorar format antic
                    if (!lineClean.Contains(':')) continue;
                    int pos;
                    // Si no especifica un numero despres de BPM és el primer
                    if (lineClean[4] == ':') pos = 0;
                    else pos = base36ToDecimal(lineClean.Substring(4, 2));
                    string t = lineClean.Substring(lineClean.LastIndexOf(':') + 1).TrimEnd().TrimStart();
                    if (t.Contains('.')) BPMs[pos] = float.Parse(t, CultureInfo.InvariantCulture);
                    else BPMs[pos] = (float)int.Parse(t, CultureInfo.InvariantCulture);
                }
                else if (lineClean.StartsWith("#WAV"))
                {
                    int zz = base36ToDecimal(lineClean.Substring(4, 2));
                    string t = lineClean.Substring(lineClean.LastIndexOf(':') + 1).TrimEnd().TrimStart();
                    chips[zz] = t;
                }
                else if (lineClean.StartsWith("#VOLUME"))
                {
                    int zz = base36ToDecimal(lineClean.Substring(7, 2));
                    string t = lineClean.Substring(lineClean.LastIndexOf(':') + 1).TrimEnd().TrimStart();
                    volumes[zz] = int.Parse(t);
                }
            }
        }
```

Figura 47: Funció parseFile del script DTXConverter

6.2.3 Càrrega de fitxers

Un cop llegit el fitxer DTX, el DTXConverter s'encarrega de carregar tots els fitxers de so per a la partida. A la funció createSoundSpawners (Figura 48) fa el següent:

1. Per a tots els valors de 0 fins 1295 (ZZ, el valor més gran en base36, en decimal) mira si hi ha una chip.
2. Si existeix, executa la funció asíncrona LoadClip i li passa com a paràmetre el camí d'aquest fitxer. Aquesta funció retorna un objecte del tipus AudioClip.
3. Es crea un GameObject amb un component d'AudioSource i el guardem a una llista anomenada chipSpawners.
4. Un cop creat aquest objecte, se li assigna el AudioClip obtingut al pas 2

5. Busquem si a la llista de Volums hi tenim una entrada per aquesta chip. Si la tenim, li assignem aquest volum.

```
async Task<bool> createSoundSpawners()
{
    for (int i = 1; i < 1295; i++) // Per cada possible chip
    {
        if (chips[i].Length > 0) // Si La chip existeix
        {
            if (chips[i].StartsWith("#WAV")) chips[i] = chips[i].Substring(7); // Si no hem borrat el #WAV, el borrem
            AudioClip c = await LoadClip(Path.GetFullPath(Path.Combine(path, @"..\")) + chips[i]);
            chipSpawners[i] = new GameObject();
            chipSpawners[i].transform.parent = AudioSpawners.transform;
            chipSpawners[i].AddComponent<AudioSource>();
            chipSpawners[i].name = chips[i] + "(" + i + ")";
            chipSpawners[i].GetComponent<AudioSource>().clip = c;
            if (volumes[i] != 0) // Si hi ha indicat el volum per aquest chipA
            {
                chipSpawners[i].GetComponent<AudioSource>().volume = (float)volumes[i] / 100;
            }

        }
    }
    return true;
}
```

Figura 48: Funció createSoundSpawners de DTXConverter

La funció LoadClip (Figura 49) funciona de la següent manera:

1. Primer comprova l'extensió del fitxer de so. El format DTX és compatible amb fitxers .WAV .MP3 .OGG i .XA, però Unity no és compatible amb aquest últim. Per tant, si el fitxer és .XA, intenta buscar una alternativa a la mateixa carpeta amb format .WAV, que haurà de ser proporcionada anteriorment per l'usuari.
2. Un cop tenim l'extensió, utilitzem la funció GetAudioClip de la llibreria UnityWebRequestMultimedia per a carregar el fitxer a una variable del tipus AudioClip.
3. Si no hi ha cap error, retorna l'AudioClip

```
async Task<AudioClip> LoadClip(string clipPath)
{
    AudioClip clip = null; AudioType x;
    if (clipPath.ToUpper().EndsWith("WAV")) x = AudioType.WAV;
    else if (clipPath.ToUpper().EndsWith("MP3")) x = AudioType.MPEG;
    else if (clipPath.ToUpper().EndsWith("OGG")) x = AudioType.OGGVORBIS;
    else if (clipPath.ToUpper().EndsWith("XA"))
    {
        // Si el fitxer es .XA, intentem buscar una alternativa .WAV
        clipPath = clipPath.Replace(".xa", ".WAV");
        clipPath = clipPath.Replace(".XA", ".WAV");
        clipPath = clipPath.ToUpper();
        x = AudioType.WAV;
    }
    else x = AudioType.UNKNOWN;
    using (UnityWebRequest uwr = UnityWebRequestMultimedia.GetAudioClip(clipPath, x))
    {
        uwr.SendWebRequest();
        // wrap tasks in try/catch, otherwise it'll fail silently
        try
        {
            while (!uwr.isDone) await Task.Delay(5);
            if (uwr.isNetworkError || uwr.isHttpError) Debug.Log($"{uwr.error}" + " File name: " + clipPath);
            else clip = DownloadHandlerAudioClip.GetContent(uwr);
        }
        catch (Exception err){ Debug.Log($"{err.Message}, {err.StackTrace} File name: {clipPath}");}
    }
    return clip;
}
```

Figura 49: Funció LoadClip de DTXConverter

6.2.4 Estructura de la simulació dintre de l'escena d'Unity

Per tal de mantenir la major semblança amb el funcionament del format DTX, s'ha creat la següent estructura de nodes dintre de l'escena d'Unity:

- Un prefab anomenat **Map**.
- Aquest prefab conté un node buit anomenat **AudioSpawners**, on s'hi col·loquen els AudioSpawners amb les diferents chips
- També conté un node anomenat **Channels**. Aquest conté els 12 canals que hi ha al joc (10 per als instruments, un per als canvis de BPM i un per a la música de fons)
- Cadascun d'aquests channels es troba dintre d'un element "**container**" on també **hi col·loquem els altres elements** necessaris per al funcionament: la Judgement Line, la posició on apareixen els efectes de llum i el modelat de l'element de la bateria
- Dintre de cada canal hi ha també un node buit anomenat **NotesContainer**.
- L'objectiu d'aquesta organització és poder moure totes les notes movent un sol objecte, en aquest cas el canal. D'aquesta manera estalviem rendiment comparat a moure totes les notes individualment.

Veure Figura 50.

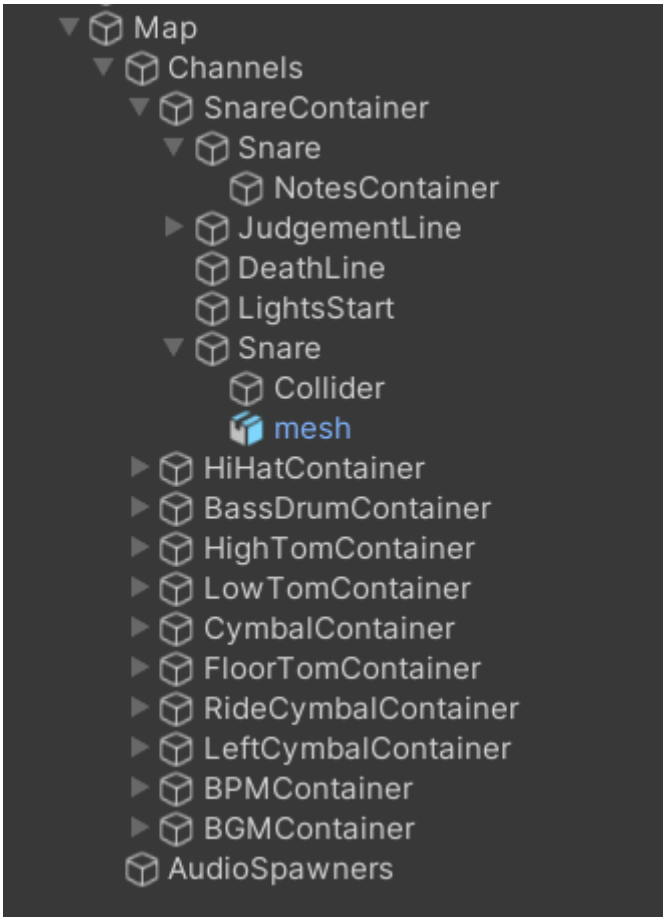


Figura 50: Prefab de "Map" que conté els elements de la simulació

6.2.5 Script Channel

Aquest script conté tot el codi per al comportament dels canals. Consta de 3 funcions:

1. Update: Gestiona el moviment del canal (i donat que les notes formen part d'aquest, també es mouen). Veure Figura 51
    a. Per a calcular la **velocitat** a la que s'ha de moure el canal, fem el següent càlcul: Delta T * BPM / 240 * noteSeparationValue.
    b. El motiu de dividir per 240 és que BPM significa notes per **minut**, i en un compàs hi ha 4 notes, per tant, si volem saber la velocitat a la que ha d'avançar hem de dividir el BPM per 60 * 4.
2. TurnLightsOn: Crea un objecte del tipus Llum amb un cert color en funció de com de bé el jugador ha tocat una nota. Veure Figura 51.
3. HandleInput: Funció que es crida quan el jugador activa l'instrument associat a aquest canal (Figura 52).
    a. Primer localitza la nota més propera a la Judgement Line.
    b. Un cop la té, calcula la distància entre la nota i la Judgement Line.
    c. En funció de com de a prop hi són, pot:
        i. Donar la nota per Perfecta
        ii. Donar la nota per Bona
        iii. Declarar la nota com a Fallada
        iv. Si la nota és prou lluny, reprodueix el so però no fa res més. Això es fa per evitar castigar al jugador per tocar notes en moments de silenci per a portar el ritme.

d. Si cal, afegeix punts a la puntuació i elimina la nota.
e. També s'encarrega de decidir quina chip s'ha de reproduir. Si el jugador toca una nota, és la chip que hi té assignada. En cas que hagi tocat en un moment de silenci, sonarà l'última chip reproduïda per aquell canal.

```
// Update is called once per frame
void Update()
{
    if (moving) transform.Translate(Vector3.Scale(Vector3.Scale(new Vector3(0, -1, 0),
    (transform.parent.transform.localScale)), c.transform.localScale) * Time.deltaTime * BPM / 240 * c.noteSeparationValue);
    if (Input.GetKeyDown(button)) handleInput();

}
public IEnumerator TurnLightsOn(int c)
{
    if (defaultChip != -1 && lightPrefab){
        Color col = color * 2;
        var a = Instantiate(lightPrefab, LightsStart);
        Material mat = lightPrefab.GetComponent<Renderer>().sharedMaterial;
        mat.EnableKeyword("_EMISSION");
        if (c == 1) col = a.GetComponent<TravelingLight>().Perfect;
        if (c == 2) col = a.GetComponent<TravelingLight>().Good;
        if (c == 3) col = a.GetComponent<TravelingLight>().Miss;
        mat.SetColor("_EmissionColor", col);
        yield return new WaitForSecondsRealtime(0.0125f);
    }
}
```

Figura 51: Funcions Update i TurnLightsOn del script Channel

```
public void handleInput()
{
    int co = 0;
    if (notesContainer.transform.childCount > 0)
    {
        Nota a = notesContainer.transform.GetChild(0).GetComponent<Nota>();
        for (int i = 0; i < notesContainer.transform.childCount; i++)
        {
            Transform g = notesContainer.transform.GetChild(i);
            if (g.position.y < a.transform.position.y) a = g.GetComponent<Nota>();
        }
        if (a.DTXConverter)
        {
            float distance = Math.Abs(Vector3.Distance (a.transform.position, judgement.transform.position));
            if (distance < 50)
            {
                c.PerfectNote();
                Destroy(a.gameObject);
                co = 1;
            }
            else if (distance < 100)
            {
                c.GoodNote();
                Destroy(a.gameObject);
                co = 2;
            }
            else if (distance < 150)
            {
                c.MissedNote();
                Destroy(a.gameObject);
                co = 3;
            }
            defaultChip = a.objectNumber;
        }
        else Destroy(a.gameObject);

    }
    if (defaultChip != -1)
    {
        c.playChip(defaultChip);
        if (lightPrefab) StartCoroutine(TurnLightsOn(co));
    }
}
```

Figura 52: Funció handleInput del script Channel

6.2.6 Script Nota

Aquest script és el que tenen totes les notes. Consta de 2 funcions:

1. Update: S'encarrega de fer que la nota només sigui visible quan sigui a prop del jugador
2. OnTriggerEnter: S'encarrega de gestionar que passa si la nota entra en contacte amb la JudgementLine o la DeathLine. Si la nota és del canal 1 o dels canals 3 o 8 (cançó de fons i canvi de BPM, respectivament) aquestes s'activen. Si és de qualsevol altre canal i arriben a la DeathLine, el joc la destrueix i la contem com a nota fallida. A més si el mode "Auto" és activat, la nota es toca tot sola al arribar a la Judgement Line.

Veure Figura 53.

```
// Update is called once per frame
void Update()
{
    if (visible) m.enabled = (transform.parent.transform.parent.localPosition.y + transform.localPosition.y) < 80;
}
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject == channel.judgement){
        if (objectChannel == DTXConverter.eBPMChannel){
            DTXConverter.changeBPM(objectNumber);
            Destroy(gameObject);
        }else if (objectChannel == DTXConverter.eBGMChannel){
            DTXConverter.playChip(objectNumber);
            Destroy(gameObject);
        }else if (DTXConverter.auto) {
            DTXConverter.playChip(objectNumber);
            DTXConverter.PerfectNote();
            Destroy(gameObject);
            channel.StartCoroutine(channel.TurnLightsOn(1));
        }

    }else if (other.gameObject == channel.death){
        DTXConverter.MissedNote();
        channel.StartCoroutine(channel.TurnLightsOn(3));

        Debug.Log("Missed note: objectNumber " + objectNumber + "   objectChannel " + objectChannel);
        Destroy(gameObject);
    }
}
```

Figura 53: Funcions del script Nota

6.2.7 Creació del Chart

Un cop carregades les dades del fitxer DTX i carregats els fitxers de so, la classe DTXConverter ja té tot el necessari per a poder crear el chart. L'algorisme segueix els següents passos per a cada entrada al diccionari map:

1. Obté el nombre del compàs i del canal i els guarda en variables
2. Extreu tots els Strings que hi ha guardats a aquella entrada del diccionari i els guarda a una llista
3. Comença una iteració d'aquesta llista
4. Primer comprova si és un element del canal 2 (canvis de compàs). Si ho és, actualitza la variable timeSignature
5. Si no ho és, separa el valor en parelles de 2 nombres en base36. Aquests

valors representen les "chips" que hi ha dintre d'aquell compàs, amb l'excepció del valor 00 que significa NULL.

6. Per cadascuna d'aquestes parelles de valors, creem un objecte del tipus Nota i li assignem tots els valors necessaris: nom, color, canal al que pertany, posició, si és visible o no, quina chip ha de reproduir i quin prefab ha d'utilitzar per al model. Tots aquests valors varien en funció de a quin canal pertanyen
7. Per saber on col·locar la nota dintre de l'espai de món, hem de tenir en compte diferents coses. Volem que els compassos es vagin col·locant cada cop més a lluny i que al fer baixar el chart sencer, arribin ordenats a la Judgement Line, per tant, la posició Y parteix de el nombre de compàs. A més, dintre d'un propi compàs l'espai ha d'estar dividit per a les diferents notes (Si hi ha 8 notes, dividit en 8 segments). Per a fer això, al nombre de compàs li hem de sumar un valor que va de 0 a 1 depenent de a quin punt del compàs es troba la nota. Per això, dividim 1 per el nombre de notes que hi ha al compàs i anomenem aquest valor distància. Multipliquem distància pel nombre de nota dintre del compàs, començant a comptar per 0 (per exemple, si hi ha 4 notes dintre d'un compàs, i volem saber on col·locar la 3a, seria 0.25 * 2). També volem poder controlar la distància entre les notes, ja que depenent de la preferencia de l'usuari les pot voler més a prop o més separades, per tant, es multiplica aquest valor per la variable noteSeparationValue. A més volem que hi hagi un petit espai de marge per a que el jugador pugui veure com comença la cançó amb uns segons d'antelació. Això dóna la fórmula final de : Y = ((compàs + (nombreNota * distància)) * noteSeparationValue) + 4.
8. Si és un objecte d'un compàs que s'activa amb un pedal, dintre de la nota activem una icona que mostra un peu.
9. També assignem la rotació i l'escala per a que siguin iguals que l'instrument que representen.
10. També assignem com a pare de la nota al container dintre del canal.

Un cop s'acaba aquest procés, tenim tot el chart de la cançó. Veure les Figures 54, 55 i 56.

Figura 54: Exemple d'una secció d'un chart

```
void generateMap()
{
    foreach (String s in map.Keys)
    {
        int i = int.Parse(s.Substring(3));
        int j = int.Parse(s.Substring(0, 3));
        List<string> objectsArray;
        map.TryGetValue(s, out objectsArray);
        foreach (string objects in objectsArray)
        {
            if (i == 2)
            {
                timeSignature = double.Parse(objects, CultureInfo.InvariantCulture);
                continue;
            }
            List<string> notes = GetChunks(objects, 2);
            double distance = (double)1 / notes.Count;
            for (int m = 0; m < notes.Count; m++)
            {
```

Figura 55: Funció generateMap de DTXConverter (part 1)

```
for (int m = 0; m < notes.Count; m++)
{
    if (!notes[m].Equals("00"))
    {
        Channel newParent = GetChannelByNumber(i);
        Color c = newParent.color;
        if (newParent.GetComponent<Channel>().defaultChip == -1)
        {
            newParent.GetComponent<Channel>().defaultChip = base36ToDecimal(notes[m]);
        }
        var newObj = Instantiate(newParent.notePrefab, newParent.gameObject.transform);
        newObj.transform.localPosition = new Vector3(0, ((j + (float)(m * distance)) * noteSeparationValue) + 4, 0);
        if (newParent.instrumentTransform){
            var thing = newObj.transform.Find("Model").transform;
            thing.rotation = newParent.instrumentTransform.rotation;
            thing.parent = newParent.instrumentTransform.parent;                    // Detach
            thing.localScale = newParent.instrumentTransform.localScale;
            thing.SetParent(newObj.transform, true);
        }
        newObj.GetComponent<Nota>().objectNumber = base36ToDecimal(notes[m]);
        newObj.GetComponent<Nota>().objectChannel = i;
        newObj.GetComponent<Nota>().DTXConverter = this;
        if (i == HiHatOpen)
        {
            newObj.GetComponent<Nota>().FootIcon.SetActive(true);
        }
        if (i == 1 || i == 8)
        {
            newObj.GetComponent<Nota>().visible = false;
        }
        else
        {
            newObj.name = "Nota " + totalNotes;
            totalNotes++;
        }
        //newObj.transform.Find("Model").GetComponent<Renderer>().material.SetColor("_BaseColor", c);
        newObj.transform.parent = newParent.notesContainer.transform;

    }
}
```

Figura 56: Funció generateMap de DTXConverter (part 2)

6.3 Mecàniques

6.3.1 Baquetes

Les baquetes només compleixen 2 funcions: ser agafades pel jugador i detectar les col·lisions amb elements de la bateria. Per tal de fer que la baqueta es pugui agafar, només cal agregar-li un script del tipus XRGrabInteractable. Per defecte l'objecte s'agafa des del centre de l'objecte, però nosaltres volem que s'agafi d'un dels extrems. Per a situacions com aquesta, el script XRGrabInteractable permet donar una posició i utilitzar-la com a punt des d'on agafar l'objecte. Nosaltres anomenem aquest punt "pivot". Veure Figura 57.

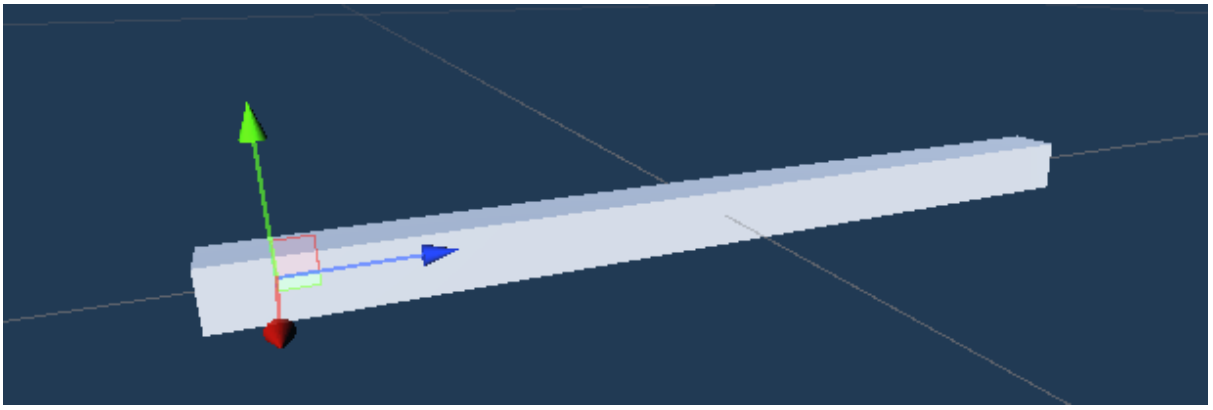


Figura 57: Posició del "Pivot" dintre de la baqueta

A més volem fer un altre canvi al comportament per defecte a l'agafar un objecte. Per defecte l'objecte només es manté agafat si el jugador prem constantment el botó per agafar un objecte. Donat que prémer un botó constantment durant molts de minuts és incòmode per al jugador, volem que només faci falta prémer el botó un cop. El primer que vam provar va ser canviar l'opció "Select Action Trigger" del component XRDirectInteractor de la mà (Figura 42) a Sticky. Això va ver que no fes falta mantenir el botó activat constantment, ja que fa que l'objecte agafat "s'apegui" a la mà, però si el jugador en qualsevol moment tornava a prémer el botó per accident, l'objecte es soltava. Per a resoldre això, hem creat un script que hereta de XRGrabInteractable i l'hem anomenat BlockGrab. Aquest script sobreescriu la funció OnSelectEntered, que es crida al moment d'agafar un objecte. Dintre d'aquesta funció, deshabilitem l'habilitat del comandament que ha agafat la baqueta de poder realitzar accions (Figura 58). D'aquesta manera, la baqueta segueix a la mà gràcies al "Sticky" i encara que el jugador premi el botó, no es soltarà la baqueta.

```
public class BlockGrab : XRGrabInteractable
{
    protected override void OnSelectEntered(SelectEnterEventArgs args)
    {
        // Deshabilitem l'input del comandament
        args.interactorObject.transform.gameObject.
        GetComponent<UnityEngine.XR.Interaction.Toolkit.ActionBasedController>().enableInputActions = false;
        // Cridem la funció original
        base.OnSelectEntered(args);
        // Marquem una de les baquetes com a agafades al singleton, i cridem la funció CheckStart del singleton
        if (!Singleton.GetInstance().Baqueta1Grabbed) Singleton.GetInstance().Baqueta1Grabbed = true;
        else if (!Singleton.GetInstance().Baqueta2Grabbed) Singleton.GetInstance().Baqueta2Grabbed = true;
        Singleton.GetInstance().checkStart();
    }
}
```

Figura 58: Script BlockGrab

Per fer la interacció amb la bateria, li hem donat a la baqueta un Collider. Aquest cobreix tota la baqueta excepte per la part on el jugador l'agafa. Veure Figura 56. A més, necessitem que el paràmetre "Movement type" del BlockGrab tingui el valor "Velocity Tracking". Veure Figura 60.

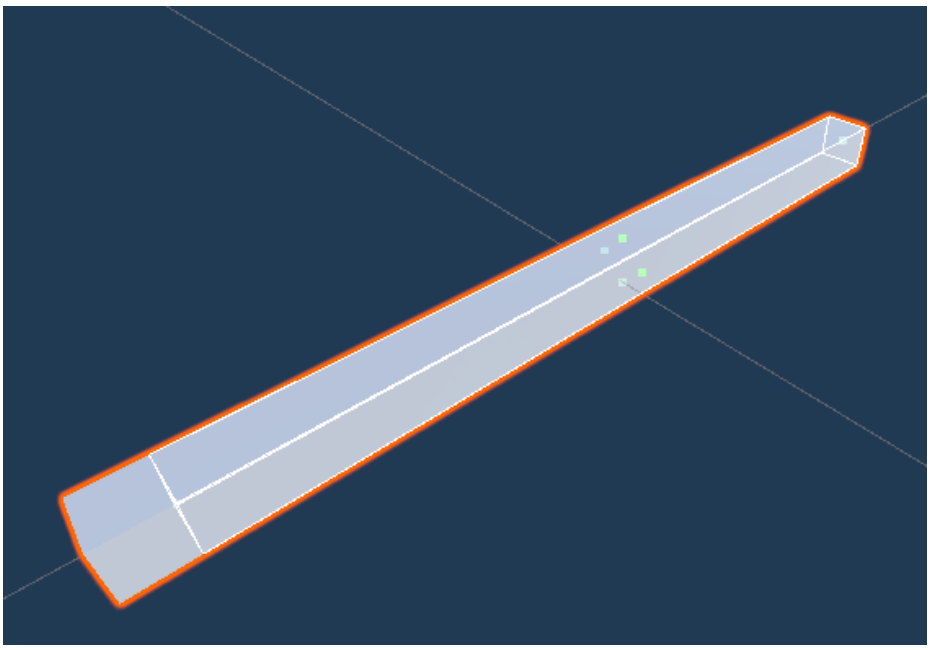


Figura 59: Collider de la baqueta

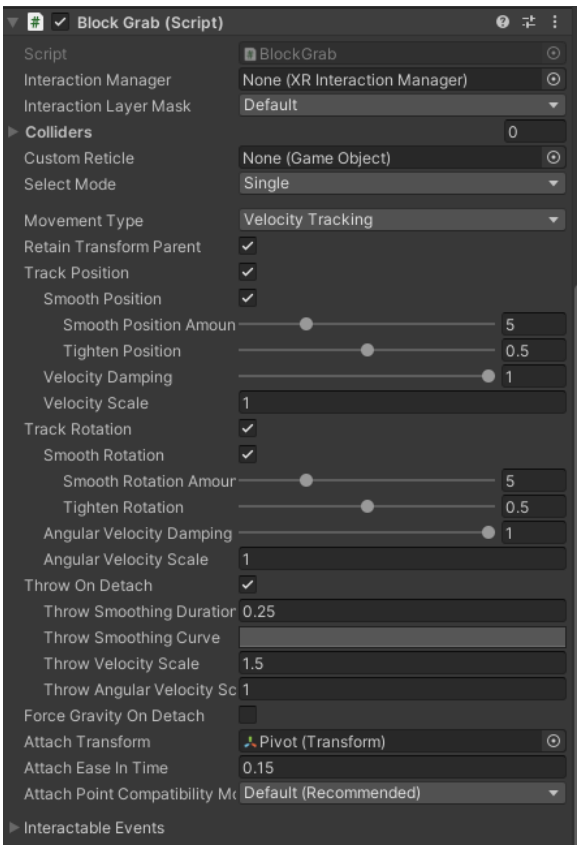


Figura 60: Component Block Grab de la baqueta

6.3.3 Bateria

La bateria està composada de diferents elements. Exceptuant el bombo, la resta consisteixen d'un model 3D de l'instrument i d'un Collider. Aquest Collider conté a més un script Instrument que serveix per a vincular aquest instrument amb la resta del codi i un script ButtonVR, que s'encarrega de cridar la funció "Play" del script

Instrument cada cop que l'usuari el toqui. Aquesta funció executa la funció handleInput del canal corresponent.

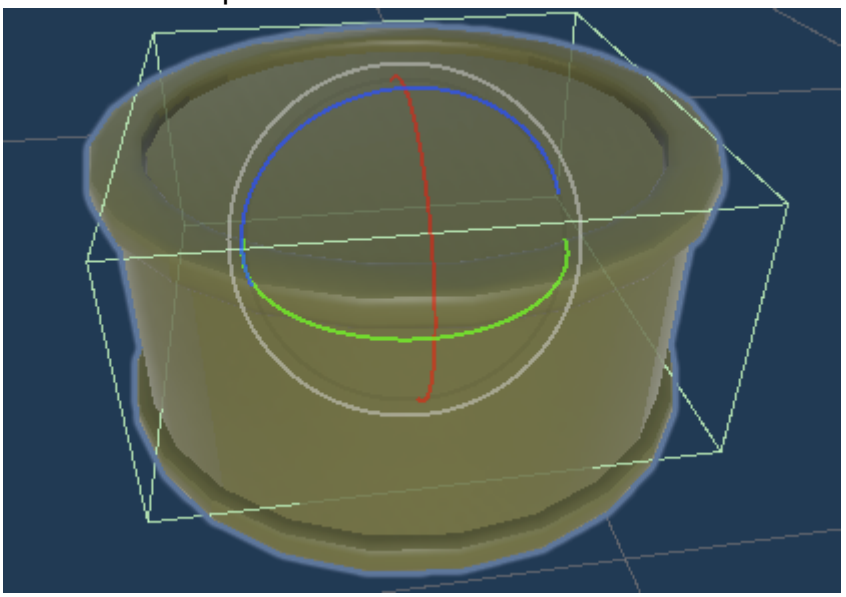


Figura 61: Instrument de la bateria i el seu Collider

```
public void Play(){
    channel.GetComponent<Channel>().handleInput();
}
```

Figura 62: Funció Play del script Instrument

Per tal de prevenir els "doble hits" que es produeixen al tornar la baqueta a sobre de l'instrument, activant la hitbox dos cops, comprovem la velocitat de la baqueta i la seva direcció, i només registrem un cop si aquesta es mou cap a baix. Veure Figura 63.

```
private void OnTriggerEnter(Collider other)
{
    if (!isPressed1 && other.name.Equals("Baqueta1") && other.gameObject.GetComponent<Rigidbody>().velocity.y < -0.1)
    {
        onPress.Invoke();
        if (sound != null) sound.Play();
        isPressed1 = true;
    }
    if (!isPressed2 && other.name.Equals("Baqueta2") && other.gameObject.GetComponent<Rigidbody>().velocity.y < -0.1)
    {
        onPress.Invoke();
        if (sound != null) sound.Play();
        isPressed2 = true;
    }
}

private void OnTriggerExit(Collider other)
{
    if (other.name.Equals("Baqueta1"))
    {
        onRelease.Invoke();
        isPressed1 = false;
    }
    if (other.name.Equals("Baqueta2"))
    {
        onRelease.Invoke();
        isPressed2 = false;
    }
}
```

Figura 63: Codi del script ButtonVR

També existeix dintre de la bateria un component PedalsController que s'encarrega d'enregistrar si s'estan prement les tecles 5 o shift esquerre, que representen el bombo i el pedal del charleston respectivament. L'usuari ha de configurar els seus pedals per a que activin aquestes tecles. Si s'activa el 5, el script crida la funció handleInput del bombo. A més, depenent de si el shift esquerre és actiu o no, el xarleston està en la seva posició "oberta" o "tancada", que en el nostre cas implica fer aparèixer o desaparèixer l'element HiHatOpen i el HiHatClose. Veure Figura 64.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Alpha5))
    {
        bassDrum.handleInput();

    }
    HiHatClosed.SetActive(!Input.GetKey(KeyCode.LeftShift));
    HiHatOpen.SetActive(Input.GetKey(KeyCode.LeftShift));
}
```

Figura 64: Funció Update del script PedalsController

6.3.4 Puntuació

Dintre del script DTXController tenim totes les variables i les funcions que tenen a veure amb la puntuació. Tenim les funcions PerfectNote, GoodNote i MissedNote que agreguen la puntuació que toca per a cada tipus de nota i pugen o eliminen el combo en cas d'una nota errònia. PerfectNote i GoodNote també criden a una funció anomenada IncreaseScore, que s'encarrega d'afegir la puntuació al total i augmentar el combo. El sistema comença amb un ComboThreshold de 5. Sempre que el jugador arribi al ComboThreshold, es duplica el multiplicador de puntuació i el nou ComboThreshold. En el moment que el jugador falli una nota, el combo torna a 0, el multiplicador de puntuació a 1 i el ComboThreshold a 5, els seus valors originals. Veure Figura 65.

```
public void IncreaseScore(int ammount)
{
    comboCounter += 1;
    combo += 1;
    if (comboCounter >= comboThreshold){
        comboMultiplier *= 2;
        comboCounter = 0;
        comboThreshold *= 2;
    }
    score += ammount * comboMultiplier;
}

public void PerfectNote()
{
    IncreaseScore(300);
    notesPassed++;
    perfectNotes++;
    checkForGameEnd();
}

public void GoodNote()
{
    IncreaseScore(150);
    notesPassed++;
    goodNotes++;
    checkForGameEnd();
}

public void MissedNote()
{
    notesPassed++;
    missedNotes++;
    combo = 1;
    comboCounter = 0;
    comboMultiplier = 1;
    comboThreshold = ogComboThreshold;
    checkForGameEnd();
}
```

Figura 65: Funcions relacionades amb la puntuació de DTXController

6.4 Singleton

El joc compta amb un Singleton dintre de l'escena principal. Dintre d'aquest és on es gestiona l'aparició dels elements del joc, la transició entre estats de la partida i dels diferents elements de la interfície.

6.4.1 Estat inicial

Al començar el joc, el menú principal és actiu així com els raigs de les mans per a interactuar amb la interfície. El joc es queda en aquest estat fins que el jugador selecciona l'opció de jugar al menú. Veure Figura 66.

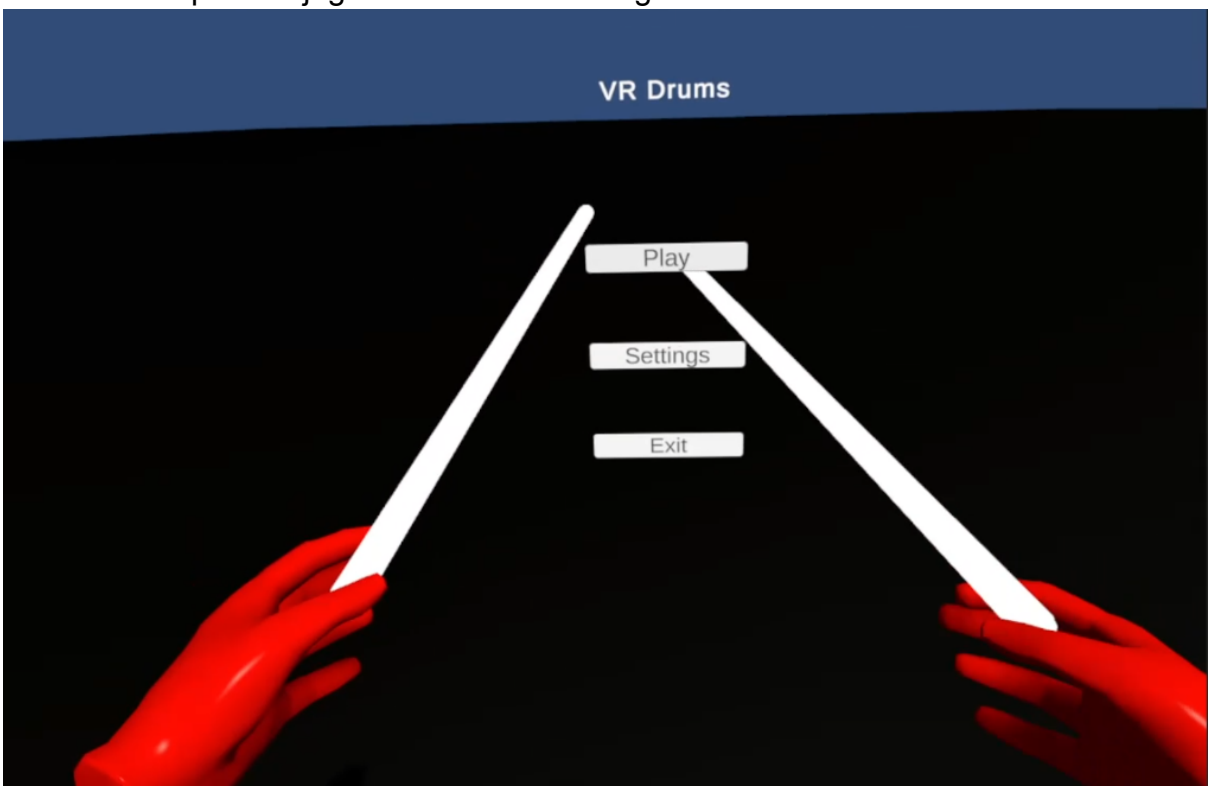


Figura 66: Menú principal

6.4.2 Selecció de cançó

Un cop el jugador tria l'opció de jugar, el Singleton fa desaparèixer el menú principal i fa aparèixer el menú de selecció de cançó. Veure Figura 67.

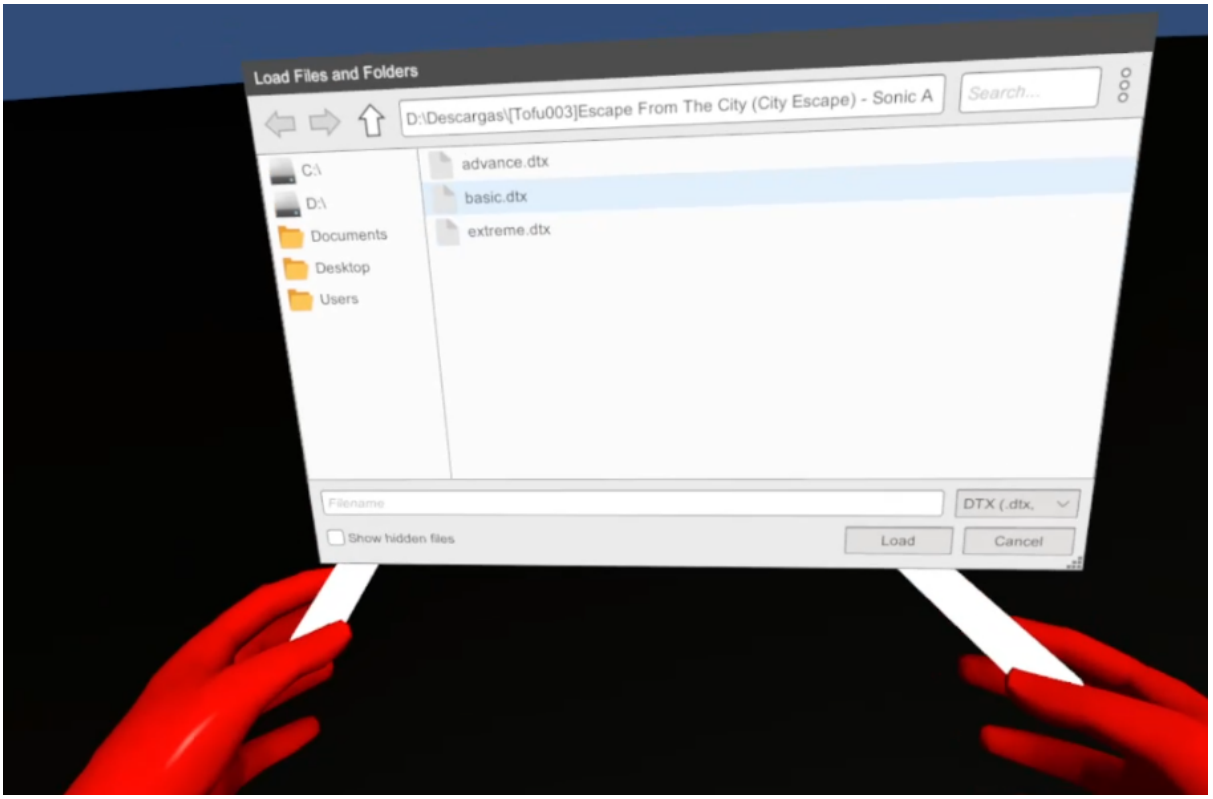


Figura 67: Selecció de cançó

6.4.3 Començament de la partida

Un cop el jugador ha triat la cançó, el Singleton crea una instància del component Map, que conté el DTXController, la bateria i el chart. Li passa al DTXController el camí de la cançó que ha triat l'usuari i aquest comença a carregar el fitxer i els sons. El Singleton també fa aparèixer dues baquetes, i no dóna l'ordre al DTXController de començar la partida fins que el jugador no les ha agafat. Veure Figura 68.

Figura 68: Imatge de la bateria i les baquetes abans de que les agafi el jugador

6.4.4 Final de la partida

Un cop s'ha acabat la partida, el DTXController crida la funció CloseGameAndShowScore del Singleton, que s'encarrega de destruir el component Map i mostra la pantalla de resultats.

7. Resultats

En aquest apartat analitzarem el grau d'assoliment dels objectius del projecte (Veure Apartat 1.3).

- Aprendre i dominar els complements de VR a Unity

S'ha experimentat bastant amb els diferents tipus d'interaccions possibles en VR a Unity amb els plugins OpenXR i XRInteractionToolkit per tal de trobar la millor combinació d'opcions per al projecte.

- Creació del terreny de joc

El prototip compta amb un terreny de joc on el jugador te al seu abast tots els

elements necessaris per a jugar al joc.

- Creació de les diferents parts de la bateria i les interaccions amb el jugador

El prototip conté tots els elements típics d'una bateria i es pot interactuar amb ells d'una forma prou semblant a com seria una bateria real, incloent-hi el funcionament del xarleston amb els seus dos estats controlats amb un pedal.

- Disseny de les interfícies i elements del joc

El prototip conté totes les interfícies necessàries per accedir a les diferents seccions d'aquest, així com la informació necessària per a oferir feedback al jugador.

- Definició i implementació de les diferents modalitats de joc

S'ha definit i implementat la modalitat principal de joc, el joc lliure amb cançons triades per l'usuari. S'ha decidit deixar de banda altres possibles modalitats (com un mode lliure o una campanya) per a treball futur amb l'objectiu de prioritzar altres aspectes més importants.

- Disseny d'un sistema de puntuació

El prototip conté un sistema de puntuació força comparable en quant a funcionament i complexitat al d'altres jocs de ritme.

- Implementació del format DTX per a poder importar cançons d'altres jocs que utilitzin aquest format

S'ha implementat la part més important del format DTX, que és la llista de notes i canvis de BPM i reproducció de sons d'una cançó. Altres elements més estètics s'ha decidit deixar-los per a treball futur amb l'objectiu de prioritzar altres aspectes més importants.

En conclusió, creiem que els objectius que s'havien plantejat s'han assolit.

8. Conclusions

Hem aconseguit desenvolupar amb èxit un prototip per a un videojoc en VR per a poder aprendre a tocar, complint amb els objectius principals del projecte.

Aquest projecte ha sigut el més ambiciós que hem realitzat en l'àmbit dels videojocs. Ha requerit l'ús de totes les habilitats tècniques apreses durant el grau. Malgrat l'altíssima dificultat per a programar algun dels aspectes més tècnics (principalment el format DTX), creiem que hem gaudit molt al desenvolupar-lo i creiem que aquest prototip pot ser una molt bona base per a desenvolupar un videojoc complet.

Al llarg del projecte ens hem trobat amb una gran quantitat de problemes, especialment en dos apartats:

- Desenvolupament per a VR: aquest projecte ha estat desenvolupat per només una persona i això ha endarrerit el projecte bastant ja que per a cada iteració dels elements de VR feia falta posar les ulleres i després treure-les.
- Format DTX: comprendre el funcionament d'aquest format ha sigut increïblement complex, entre altres motius perquè la documentació d'aquest format només existeix en japonès. Hi ha una versió traduïda a l'anglès d'una de les versions, però és força desactualitzada i conté molts d'errors de traducció que ens han fet perdre molt de temps.

Finalment, el projecte conté la majoria de coses que havíem planificat i les mecàniques estan totes implementades i preparades per a ser expandides en un futur.

9. Treball futur

En aquest apartat explicarem totes les funcionalitats que s'afegirien en cas d'expandir el projecte en un futur:

- Donar una estètica al joc.
- Oferir altres modalitats de joc (mode pràctica, tutorial i campanya entre altres)
- Millora dels gràfics
- Millora de la sensació d'impacte al tocar els instruments
- Millora de la interfície
- Oferir compatibilitat amb cançons que tinguin més compassos a banda de 4/4
- Oferir compatibilitat amb cançons d'altres formats a banda de DTX (per exemple el format .chart de Clone Hero, joc que s'ha mencionat a l'apartat 2.5.2)
- Oferir opcions de personalització de la bateria: posició dels elements, tamany…
- Probar que el joc funciona amb altres ulleres de realitat virtual
- Solucionar un bug que provoca que les cançons llargues es dessincronitzen
- Solucionar un bug que impedeix resoldre correctament quina nota és la més propera a un instrument quan el jugador el toca

10. Bibliografia

Jasmine Katatikarn (21 de març de 2023). Virtual Reality Statistics: The Ultimate List in 2023. *Academy of Animated Art*.
https://academyofanimatedart.com/virtual-reality-statistics/

Nick Cesarz (5 de juny de 2023). Best Beginner Drum Sets of 2023: Top Kits for New Drummers. *Drumming Review*.
https://drummingreview.com/best-beginner-drum-set/

Iván Asensio. ¿Cuánto se gana en la industria del videojuego? *Master.D*.
https://www.masterd.es/blog/sueldo-industria-videojuego

Valem. (18 de juny de 2020). *Introduction to VR in Unity - UNITY XR TOOLKIT*. [Llista de reproducció].
https://www.youtube.com/watch?v=NU_cLqYrYjo&list=PLrk7hDwk64-a_gf7mBBduQb3PEBYnG4fU&ab_channel=Valem

Unity Technologies. (25 d'agost de 2023) Create with VR. *Unity Blog*.
https://learn.unity.com/course/create-with-vr

Unity Technologies. (23 d'abril de 2023) VR development in Unity. *Unity Documentation*.
https://docs.unity3d.com/2020.3/Documentation/Manual/VROverview.html

Unity Technologies. (7 de març de 2023) OpenXR Plugin. *Unity Documentation*.
https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.7/manual/index.html

Unity Technologies. (24 d'octubre de 2022) XR Interaction Toolkit. *Unity Documentation*.
https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.2/manual/index.html

DTX file format specifications. *OSDN*.
https://osdn.net/projects/dtxmania/wiki/DTX+data+format#:~:text=DTXMania%20can%20handle%20with%20WAV,DTXMania%20Release%20062b060416%20or%20later.

11 Annexos

Aquí es recopilen altres scripts realitzats que no s'han mencionat en l'apartat d'implementació, així com la totalitat dels fitxers més grans. També s'ha adjuntat el projecte sencer de Unity, així com una build del joc. També s'ha adjuntat un enllaç a un vídeo del prototip. Aquest vídeo no té so, ja que a l'hora d'executar el joc, el so es reprodueix a les ulleres de VR, no a l'ordinador on s'executa el joc.

AnimateHandOnInput.cs

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

public class AnimateHandOnInput : MonoBehaviour
{
    public InputActionProperty pinchAnimationAction;
    public InputActionProperty gripAnimationAction;
    public Animator handAnimator;
    // Start is called before the first frame update
    void Start()
    {

    }

    void Update()
    {
        float triggerValue =
pinchAnimationAction.action.ReadValue<float>();
        handAnimator.SetFloat("Trigger", triggerValue);

        float gripValue =
gripAnimationAction.action.ReadValue<float>();
        handAnimator.SetFloat("Grip", gripValue);
    }
}
```

BlockGrab.cs

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

public class BlockGrab : XRGrabInteractable
{
    protected override void
OnSelectEntered(SelectEnterEventArgs args)
    {
        // Deshabilitem l'input del comandament
        args.interactorObject.transform.gameObject.

GetComponent<UnityEngine.XR.Interaction.Toolkit.ActionBasedCon
troller>().enableInputActions = false;
        // Cridem la funció original
        base.OnSelectEntered(args);
        // Marquem una de les baquetes com a agafades al
singleton, i cridem la funció CheckStart del singleton
        if (!Singleton.GetInstance().Baqueta1Grabbed)
Singleton.GetInstance().Baqueta1Grabbed = true;
        else if (!Singleton.GetInstance().Baqueta2Grabbed)
Singleton.GetInstance().Baqueta2Grabbed = true;
        Singleton.GetInstance().checkStart();
    }
}
```

Channel.cs

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

public class Channel : MonoBehaviour
{
    public float BPM = 0;
    public bool moving = false;
    public GameObject converter;
    public GameObject judgement;
    public GameObject death;
    public GameObject notesContainer;
    public int defaultChip = -1;
    public KeyCode button;
    public int instrumentChannel;
    public bool togleable = false;
    public bool togleValue = false;
    [ColorUsage(true, true)]
    public Color color;
    public float maxTimeLight = 0.1f;
    public Transform instrumentTransform;
    public GameObject notePrefab;
    public GameObject lightPrefab;
    public Transform LightsStart;


    DTXConverter c;
    // Start is called before the first frame update
    void Start()
    {
        c = converter.GetComponent<DTXConverter>();
    }

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

```
    {
        if (moving)
transform.Translate(Vector3.Scale(Vector3.Scale(new Vector3(0,
-1, 0),
        (transform.parent.transform.localScale)),
c.transform.localScale) * Time.deltaTime * BPM / 240 *
c.noteSeparationValue);
        if (Input.GetKeyDown(button)) handleInput();

    }
    public IEnumerator TurnLightsOn(int c)
    {
        if (defaultChip != -1 && lightPrefab){
            Color col = color * 2;
            var a = Instantiate(lightPrefab, LightsStart);
            Material mat =
lightPrefab.GetComponent<Renderer>().sharedMaterial;
            mat.EnableKeyword("_EMISSION");
            if (c == 1) col =
a.GetComponent<TravelingLight>().Perfect;
            if (c == 2) col =
a.GetComponent<TravelingLight>().Good;
            if (c == 3) col =
a.GetComponent<TravelingLight>().Miss;
            mat.SetColor("_EmissionColor", col);
            yield return new WaitForSecondsRealtime(0.0125f);
        }
    }
    public void handleInput()
    {
        int co = 0;
        if (notesContainer.transform.childCount > 0)
        {
            Nota a =
notesContainer.transform.GetChild(0).GetComponent<Nota>();
            for (int i = 0; i <
notesContainer.transform.childCount; i++)
```

```
            {
                Transform g =
notesContainer.transform.GetChild(i);
                if (g.position.y < a.transform.position.y) a =
g.GetComponent<Nota>();
            }
            if (a.DTXConverter)
            {
                float distance = Math.Abs(Vector3.Distance
(a.transform.position, judgement.transform.position));
                if (distance < 50)
                {
                    c.PerfectNote();
                    Destroy(a.gameObject);
                    co = 1;
                }
                else if (distance < 100)
                {
                    c.GoodNote();
                    Destroy(a.gameObject);
                    co = 2;
                }
                else if (distance < 150)
                {
                    c.MissedNote();
                    Destroy(a.gameObject);
                    co = 3;
                }
                defaultChip = a.objectNumber;
            }
            else Destroy(a.gameObject);

        }
        if (defaultChip != -1)
        {
            c.playChip(defaultChip);
            if (lightPrefab) StartCoroutine(TurnLightsOn(co));
```

```
        }
    }
}
```

DisapearWhenNotNeeded.cs

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

public class DisapearWhenNotNeeded : MonoBehaviour
{
    MeshRenderer m;
    public Transform judgementLine;
    // Start is called before the first frame update
    void Start()
    {
        m = this.GetComponent<MeshRenderer>();

    }

    // Update is called once per frame
    void Update()
    {
        if (transform.position.y <
transform.parent.parent.position.y) Destroy(gameObject);
        m.enabled = (transform.parent.localPosition.y +
transform.localPosition.y) < 40;
    }
}
```

DTXConverter.cs

```
using System.IO;
using UnityEngine;
using UnityEngine.Networking;
using UnityEditor;
using System.Collections.Generic;
```

```
using System.Collections;
using System.Text.RegularExpressions;
using System.Globalization;
using System;
using System.Threading.Tasks;
using TMPro;

public class DTXConverter : MonoBehaviour
{
    public string path;
    string[] fileContent;
    public float[] BPMs = new float[1296];
    public string[] chips = new string[1296];
    public int[] volumes = new int[1296];
    public float currentBPM = 0;
    public GameObject channels;
    public GameObject AudioSpawners;
    public GameObject notaPrefab;
    public GameObject quarterPrefab;
    public GameObject beginingPrefab;
    public GameObject[] chipSpawners;

    bool over = true;


    public GameObject BGMChannel;
    public GameObject BPMChannel;
    public GameObject HiHatCloseChannel;
    public GameObject SnareChannel;
    public GameObject BassDrumChannel;
    public GameObject HighTomChannel;
    public GameObject LowTomChannel;
    public GameObject CymbalChannel;
    public GameObject FloorTomChannel;
    public GameObject HiHatOpenChannel;
    public GameObject RideCymbalChannel;
    public GameObject LeftCymbalChannel;
```

```
    public int eBGMChannel = 1;
    public int eBPMChannel = 8;
    public int HiHatClose = 11;
    public int Snare = 12;
    public int BassDrum = 13;
    public int HighTom = 14;
    public int LowTom = 15;
    public int Cymbal = 16;
    public int FloorTom = 17;
    public int HiHatOpen = 18;
    public int RideCymbal = 19;
    public int LeftCymbal = 0x1A;
    public int CompasChannel = 02;

    public int noteSeparationValue = 12;

    double timeSignature = 1;

    public TMP_Text AccLabel;
    public TMP_Text ScoreLabel;
    public TMP_Text ComboLabel;



    public int lastMeasure = 0;
    public int totalNotes = 0;
    public int notesPassed = 0;
    public int perfectNotes = 0;
    public int goodNotes = 0;
    public int missedNotes = 0;
    public int score = 0;
    public int combo = 0;
    public int comboMultiplier = 1;
    public int comboCounter = 0;
    public int comboThreshold = 5;
    public int ogComboThreshold = 5;
```

```
    string artist;
    string songName;

    public bool auto = false;

    SortedDictionary<string, List<string>> map = new
SortedDictionary<string, List<string>>();
    private const string CharList =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int base36ToDecimal(string input)
    {
        input = input.ToUpper();
        int result = 0;
        int pos = 0;
        for (int i = input.Length - 1; i >= 0; i--)
        {
            result += CharList.IndexOf(input[i]) *
(int)Math.Pow(36, pos);
            pos++;
        }
        return result;

    }

    public void changeBPM(int a)
    {
        currentBPM = BPMs[a];
    }


    public void playChip(int chip)
    {
        var a = chipSpawners[chip];
        a.GetComponent<AudioSource>().Play();
    }
```

```
    void createBarForChannelAtHeight(int pos, GameObject
channel)
    {
        bool transf =
channel.GetComponent<Channel>().instrumentTransform;
        var newObj = Instantiate(beginingPrefab,
channel.transform);
        newObj.transform.localPosition = new Vector3(0, (pos *
noteSeparationValue), 0);
        if (transf) newObj.transform.rotation =
channel.GetComponent<Channel>().instrumentTransform.localRotat
ion;
        newObj = Instantiate(quarterPrefab,
channel.transform);
        newObj.transform.localPosition = new Vector3(0, ((pos
+ (float)0.25) * noteSeparationValue), 0);
        if (transf) newObj.transform.rotation =
channel.GetComponent<Channel>().instrumentTransform.localRotat
ion;
        newObj = Instantiate(quarterPrefab,
channel.transform);
        newObj.transform.parent = channel.transform;
        newObj.transform.localPosition = new Vector3(0, ((pos
+ (float)0.50) * noteSeparationValue), 0);
        if (transf) newObj.transform.rotation =
channel.GetComponent<Channel>().instrumentTransform.localRotat
ion;
        newObj = Instantiate(quarterPrefab,
channel.transform);
        newObj.transform.localPosition = new Vector3(0, ((pos
+ (float)0.75) * noteSeparationValue), 0);
        if (transf) newObj.transform.rotation =
channel.GetComponent<Channel>().instrumentTransform.localRotat
ion;

    }
```

```
void generateBars()
{
    var newObj = channels;
    for (int p = 0; p < lastMeasure + 2; p++)
    {
        createBarForChannelAtHeight(p, HiHatCloseChannel);
        createBarForChannelAtHeight(p, SnareChannel);
        createBarForChannelAtHeight(p, BassDrumChannel);
        createBarForChannelAtHeight(p, HighTomChannel);
        createBarForChannelAtHeight(p, LowTomChannel);
        createBarForChannelAtHeight(p, CymbalChannel);
        createBarForChannelAtHeight(p, FloorTomChannel);
        createBarForChannelAtHeight(p, RideCymbalChannel);
        createBarForChannelAtHeight(p, LeftCymbalChannel);
    }
}

Channel GetChannelByNumber(int n)
{
    Channel c = BGMChannel.GetComponent<Channel>();
    switch (n)
    {
        case 01:
            c = BGMChannel.GetComponent<Channel>();
            break;
        case 08:
            c = BPMChannel.GetComponent<Channel>();
            break;
        case 11:
            c = HiHatCloseChannel.GetComponent<Channel>();
            break;
        case 12:
            c = SnareChannel.GetComponent<Channel>();
            break;
        case 13:
```

```
                c = BassDrumChannel.GetComponent<Channel>();
                break;
            case 14:
                c = HighTomChannel.GetComponent<Channel>();
                break;
            case 15:
                c = LowTomChannel.GetComponent<Channel>();
                break;
            case 16:
                c = CymbalChannel.GetComponent<Channel>();
                break;
            case 17:
                c = FloorTomChannel.GetComponent<Channel>();
                break;
            case 18:
                c = HiHatOpenChannel.GetComponent<Channel>();
                break;
            case 19:
                c = RideCymbalChannel.GetComponent<Channel>();
                break;
            case 20:
                c = LeftCymbalChannel.GetComponent<Channel>();
                break;
        }
        return c;
    }

    void generateMap()
    {
        foreach (String s in map.Keys)
        {
            int i = int.Parse(s.Substring(3));
            int j = int.Parse(s.Substring(0, 3));
            List<string> objectsArray;
            map.TryGetValue(s, out objectsArray);
            foreach (string objects in objectsArray)
            {
```

```
                if (i == 2)
                {
                    timeSignature = double.Parse(objects,
CultureInfo.InvariantCulture);
                    continue;
                }
                List<string> notes = GetChunks(objects, 2);
                double distance = (double)1 / notes.Count;
                for (int m = 0; m < notes.Count; m++)
                {
                    if (!notes[m].Equals("00"))
                    {
                        Channel newParent =
GetChannelByNumber(i);
                        Color c = newParent.color;
                        if
(newParent.GetComponent<Channel>().defaultChip == -1)
                        {

newParent.GetComponent<Channel>().defaultChip =
base36ToDecimal(notes[m]);
                        }
                        var newObj =
Instantiate(newParent.notePrefab,
newParent.gameObject.transform);
                        newObj.transform.localPosition = new
Vector3(0, ((j + (float)(m * distance)) * noteSeparationValue)
+ 4, 0);
                        if (newParent.instrumentTransform){
                            var thing =
newObj.transform.Find("Model").transform;
                            thing.rotation =
newParent.instrumentTransform.rotation;
                            thing.parent =
newParent.instrumentTransform.parent;                    //
Detach
                            thing.localScale =
```

```
newParent.instrumentTransform.localScale;
                            thing.SetParent(newObj.transform,
true);
                        }

newObj.GetComponent<Nota>().objectNumber =
base36ToDecimal(notes[m]);

newObj.GetComponent<Nota>().objectChannel = i;

newObj.GetComponent<Nota>().DTXConverter = this;
                        if (i == HiHatOpen)
                        {

newObj.GetComponent<Nota>().FootIcon.SetActive(true);
                        }
                        if (i == 1 || i == 8)
                        {

newObj.GetComponent<Nota>().visible = false;
                        }
                        else
                        {
                            newObj.name = "Nota " +
totalNotes;
                            totalNotes++;
                        }

//newObj.transform.Find("Model").GetComponent<Renderer>().mate
rial.SetColor("_BaseColor", c);
                        newObj.transform.parent =
newParent.notesContainer.transform;

                    }
                }

            }
```

```
        }


    }



    void Start()
    {

        chipSpawners = new GameObject[1295];
        parseFile();
        StartCoroutine(LoadThings());

    }


    IEnumerator LoadThings()
    {
        generateMap();

        createSoundSpawners();

        yield return new WaitForSeconds(1);

        if (auto) startGame();


    }

    async Task<bool> createSoundSpawners()
    {
        for (int i = 1; i < 1295; i++) // Per cada possible
chip
        {
```

```
            if (chips[i].Length > 0) // Si la chip existeix
            {
                if (chips[i].StartsWith("#WAV")) chips[i] =
chips[i].Substring(7); // Si no hem borrat el #WAV, el borrem
                AudioClip c = await
LoadClip(Path.GetFullPath(Path.Combine(path, @"..\")) +
chips[i]);
                chipSpawners[i] = new GameObject();
                chipSpawners[i].transform.parent =
AudioSpawners.transform;
                chipSpawners[i].AddComponent<AudioSource>();
                chipSpawners[i].name = chips[i] + "(" + i +
")";

chipSpawners[i].GetComponent<AudioSource>().clip = c;
                if (volumes[i] != 0) // Si hi ha indicat el
volum per aquest chipA
                {

chipSpawners[i].GetComponent<AudioSource>().volume =
(float)volumes[i] / 100;
                }

            }
        }
        return true;
    }

    async Task<AudioClip> LoadClip(string clipPath)
    {
        AudioClip clip = null; AudioType x;
        if (clipPath.ToUpper().EndsWith("WAV")) x =
AudioType.WAV;
        else if (clipPath.ToUpper().EndsWith("MP3")) x =
AudioType.MPEG;
        else if (clipPath.ToUpper().EndsWith("OGG")) x =
AudioType.OGGVORBIS;
```

```
        else if (clipPath.ToUpper().EndsWith("XA"))
        {
            // Si el fitxer es .XA, intentem buscar una
alternativa .WAV
            clipPath = clipPath.Replace(".xa", ".WAV");
            clipPath = clipPath.Replace(".XA", ".WAV");
            clipPath = clipPath.ToUpper();
            x = AudioType.WAV;
        }
        else x = AudioType.UNKNOWN;
        using (UnityWebRequest uwr =
UnityWebRequestMultimedia.GetAudioClip(clipPath, x))
        {
            uwr.SendWebRequest();
            // wrap tasks in try/catch, otherwise it'll fail
silently
            try
            {
                while (!uwr.isDone) await Task.Delay(5);
                if (uwr.isNetworkError || uwr.isHttpError)
Debug.Log($"{uwr.error}" + " File name: " + clipPath);
                else clip =
DownloadHandlerAudioClip.GetContent(uwr);
            }
            catch (Exception err){ Debug.Log($"{err.Message},
{err.StackTrace} File name: {clipPath}");}
        }
        return clip;
    }

    public void startGame()
    {
        currentBPM = BPMs[0];
        if (currentBPM == 0) currentBPM = BPMs[1];
        BGMChannel.GetComponent<Channel>().moving = true;
        BPMChannel.GetComponent<Channel>().moving = true;
        HiHatCloseChannel.GetComponent<Channel>().moving =
```

```
true;
        SnareChannel.GetComponent<Channel>().moving = true;
        BassDrumChannel.GetComponent<Channel>().moving = true;
        HighTomChannel.GetComponent<Channel>().moving = true;
        LowTomChannel.GetComponent<Channel>().moving = true;
        CymbalChannel.GetComponent<Channel>().moving = true;
        FloorTomChannel.GetComponent<Channel>().moving = true;
        HiHatOpenChannel.GetComponent<Channel>().moving =
true;
        RideCymbalChannel.GetComponent<Channel>().moving =
true;
        LeftCymbalChannel.GetComponent<Channel>().moving =
true;
    }

    // Update is called once per frame
    void Update()
    {
        BGMChannel.GetComponent<Channel>().BPM = currentBPM;
        BPMChannel.GetComponent<Channel>().BPM = currentBPM;
        HiHatCloseChannel.GetComponent<Channel>().BPM =
currentBPM;
        SnareChannel.GetComponent<Channel>().BPM = currentBPM;
        BassDrumChannel.GetComponent<Channel>().BPM =
currentBPM;
        HighTomChannel.GetComponent<Channel>().BPM =
currentBPM;
        LowTomChannel.GetComponent<Channel>().BPM =
currentBPM;
        CymbalChannel.GetComponent<Channel>().BPM =
currentBPM;
        FloorTomChannel.GetComponent<Channel>().BPM =
currentBPM;
        HiHatOpenChannel.GetComponent<Channel>().BPM =
currentBPM;
        RideCymbalChannel.GetComponent<Channel>().BPM =
currentBPM;
```

```
        LeftCymbalChannel.GetComponent<Channel>().BPM =
currentBPM;
        ComboLabel.text = "x" + combo.ToString();
        ScoreLabel.text = score.ToString();
        if (notesPassed > 0) AccLabel.text = (((perfectNotes +
(goodNotes / 2)) / notesPassed) * 100).ToString() + "%";
        else AccLabel.text = "100%";

    }

    public static List<string> GetChunks(string value, int
chunkSize)
    {
        List<string> triplets = new List<string>();
        while (value.Length > chunkSize)
        {
            triplets.Add(value.Substring(0, chunkSize));
            value = value.Substring(chunkSize);
        }
        if (value != "")
            triplets.Add(value);
        return triplets;
    }

    public void channelPressed(int channel)
    {

    }

    public void IncreaseScore(int ammount)
    {
        comboCounter += 1;
        combo += 1;
        if (comboCounter >= comboThreshold){
            comboMultiplier *= 2;
            comboCounter = 0;
            comboThreshold *= 2;
```

```
        }
        score += ammount * comboMultiplier;
    }

    public void PerfectNote()
    {
        IncreaseScore(300);
        notesPassed++;
        perfectNotes++;
        checkForGameEnd();
    }

    public void GoodNote()
    {
        IncreaseScore(150);
        notesPassed++;
        goodNotes++;
        checkForGameEnd();
    }

    public void MissedNote()
    {
        notesPassed++;
        missedNotes++;
        combo = 1;
        comboCounter = 0;
        comboMultiplier = 1;
        comboThreshold = ogComboThreshold;
        checkForGameEnd();
    }

    void checkForGameEnd()
    {
        if (notesPassed >= totalNotes && !over)
        {
            StartCoroutine(FinishGame());
        }
```

```
    }

    IEnumerator FinishGame()
    {
        over = true;
        Singleton.GetInstance().lastScore = score;
        Singleton.GetInstance().lastTotalNotes = totalNotes;
        Singleton.GetInstance().lastPerfectNotes =
perfectNotes;
        Singleton.GetInstance().lastGoodNotes = goodNotes;
        Singleton.GetInstance().lastMissedNotes = missedNotes;
        Singleton.GetInstance().CloseGameAndShowScore();
        yield return new WaitForSeconds(2);
    }

    void parseFile()
    {
        if (path.Length != 0)
        {
            fileContent = File.ReadAllLines(path);
            foreach (string line in fileContent)
            {
                string lineClean = line;
                // Si conté un comentari, l'eliminem
                int index = lineClean.LastIndexOf(";");
                if (index >= 0) lineClean =
lineClean.Substring(0, index);
                if (lineClean.Length == 0) continue;
                // Si no comença amb un #, es una línia
incorrecta
                if (lineClean[0] == '#')
                {
                    // Si comença amb 5 digits, es un objecte
                    if (Regex.IsMatch(lineClean, "^#\\d{5}"))
                    {
                        string key = lineClean.Substring(1,
5);
```

```
                        string objects =
lineClean.Substring(lineClean.LastIndexOf(':') +
1).TrimEnd().TrimStart();
                        if (!map.ContainsKey(key))
                        {
                            map.Add(key, new List<string>());
                        }
                        map[key].Add(objects);
                        if (int.Parse(key.Substring(0, 3)) >
lastMeasure) lastMeasure = int.Parse(key.Substring(0, 3));

                    }
                    else if (lineClean.StartsWith("#TITLE"))
songName = lineClean.Substring(lineClean.LastIndexOf(':') +
1).TrimEnd().TrimStart();
                    else if (lineClean.StartsWith("#ARTIST"))
artist = lineClean.Substring(lineClean.LastIndexOf(':') +
1).TrimEnd().TrimStart();
                    else if (lineClean.StartsWith("#BPM"))
                    {
                        // Ignorar format antic
                        if (!lineClean.Contains(':'))
continue;
                        int pos;
                        // Si no especifica un numero despres
de BPM és el primer
                        if (lineClean[4] == ':') pos = 0;
                        else pos =
base36ToDecimal(lineClean.Substring(4, 2));
                        string t =
lineClean.Substring(lineClean.LastIndexOf(':') +
1).TrimEnd().TrimStart();
                        if (t.Contains('.')) BPMs[pos] =
float.Parse(t, CultureInfo.InvariantCulture);
                        else BPMs[pos] = (float)int.Parse(t,
CultureInfo.InvariantCulture);
                    }
```

```
                    else if (lineClean.StartsWith("#WAV"))
                    {
                        int zz =
base36ToDecimal(lineClean.Substring(4, 2));
                        string t =
lineClean.Substring(lineClean.LastIndexOf(':') +
1).TrimEnd().TrimStart();
                        chips[zz] = t;
                    }
                    else if (lineClean.StartsWith("#VOLUME"))
                    {
                        int zz =
base36ToDecimal(lineClean.Substring(7, 2));
                        string t =
lineClean.Substring(lineClean.LastIndexOf(':') +
1).TrimEnd().TrimStart();
                        volumes[zz] = int.Parse(t);
                    }
                }
            }
        }
    }

}
```

HandPresence.cs

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

public class HandPresence : MonoBehaviour
{
    public InputDeviceCharacteristics
controllerCharacteristics;
    public List<GameObject> controllerPrefabs;

    private InputDevice targetDevice;
```

```
    private GameObject spawnedController;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(GetDevices(1.0f));
    }

    IEnumerator GetDevices(float delayTime)
    {
        yield return new WaitForSeconds(delayTime);
        List<InputDevice> devices = new List<InputDevice>();
        InputDevices.GetDevices(devices);

        Debug.Log("Enumerating XR Devices...");
        foreach (var item in devices)
        {
            Debug.Log(item.name + item.characteristics);
        }
        if (devices.Count > 0)
        {
            targetDevice = devices[0];
            GameObject prefab =
controllerPrefabs.Find(controller => controller.name ==
targetDevice.name);
            if (prefab)
            {
                spawnedController = Instantiate(prefab,
transform);
            }
            else
            {
                Debug.LogError("Error");
                spawnedController =
Instantiate(controllerPrefabs[0], transform);
            }
```

```
        }
    }
    // Update is called once per frame
    void Update()
    {
        if (targetDevice.isValid)
        {
            spawnedController.SetActive(true);
        }
    }
}
```

Instrument.cs

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

public class Instrument : MonoBehaviour
{
    public GameObject channel;
    // Start is called before the first frame update
    void Start()
    {

    }

    public void Play(){
        channel.GetComponent<Channel>().handleInput();
    }
}
```

Nota.cs

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

public class Nota : MonoBehaviour
```

```
{
    public int objectNumber = 0;
    public int objectChannel = 0;
    public DTXConverter DTXConverter;
    public GameObject FootIcon;
    public float speed = 0;
    public Channel channel;
    public MeshRenderer m;
    public bool visible = true;

    // Start is called before the first frame update
    void Start()
    {
        channel =
gameObject.transform.parent.transform.parent.GetComponent<Chan
nel>();
        m.enabled = visible;
    }

    // Update is called once per frame
    void Update()
    {
        if (visible) m.enabled =
(transform.parent.transform.parent.localPosition.y +
transform.localPosition.y) < 80;
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == channel.judgement){
            if (objectChannel == DTXConverter.eBPMChannel){
                DTXConverter.changeBPM(objectNumber);
                Destroy(gameObject);
            }else if (objectChannel ==
DTXConverter.eBGMChannel){
                DTXConverter.playChip(objectNumber);
                Destroy(gameObject);
            }else if (DTXConverter.auto) {
```

```
                DTXConverter.playChip(objectNumber);
                DTXConverter.PerfectNote();
                Destroy(gameObject);

channel.StartCoroutine(channel.TurnLightsOn(1));
            }

        }else if (other.gameObject == channel.death){
            DTXConverter.MissedNote();
            channel.StartCoroutine(channel.TurnLightsOn(3));

            Debug.Log("Missed note: objectNumber " +
objectNumber + "   objectChannel " + objectChannel);
            Destroy(gameObject);
        }
    }
}
```

PedalsController.cs

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

public class PedalsController : MonoBehaviour
{

    public DTXConverter converter;
    public Channel bassDrum;
    public GameObject HiHatClosed;
    public GameObject HiHatOpen;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Alpha5))
        {
            bassDrum.handleInput();
```

```
        }

HiHatClosed.SetActive(!Input.GetKey(KeyCode.LeftShift));
        HiHatOpen.SetActive(Input.GetKey(KeyCode.LeftShift));
    }
}
```

Singleton.cs

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

public class Singleton : MonoBehaviour
{

    private static Singleton inst;

    public string path;
    public GameObject UI;
    public GameObject UIMainMenu;
    public GameObject UIPickSong;
    public GameObject UISettings;
    public GameObject UIScoreBoard;
    public GameObject MapPrefab;
    public DTXConverter converter;
    public int lastScore = 0;
    public int lastTotalNotes = 0;
    public int lastPerfectNotes = 0;
    public int lastGoodNotes = 0;
    public int lastMissedNotes = 0;
    public TMP_Text TotalScoreLabel;
    public TMP_Text AccLabel;
    public TMP_Text PerfectNotesLabel;
    public TMP_Text GoodNotesLabel;
```

```
    public TMP_Text MissedNotesLabel;
    public GameObject LeftRay;
    public GameObject LeftHand;
    public GameObject RightRay;
    public GameObject RightHand;
    public GameObject BaquetaPrefab;
    public GameObject Baqueta1;
    public Transform Baqueta1Pos;
    public bool Baqueta1Grabbed;
    public GameObject Baqueta2;
    public Transform Baqueta2Pos;
    public bool Baqueta2Grabbed;
    private void Awake()
    {
        if(inst == null)
        {
            inst = this;

            DontDestroyOnLoad(gameObject);
        }
    }

    // Start is called before the first frame update
    void Start()
    {
        FileBrowser.SetFilters(true, new
FileBrowser.Filter("DTX", ".dtx", ".DTX"));
        FileBrowser.SetDefaultFilter(".dtx");

        FileBrowser.AddQuickLink("Users", "C:\\Users", null);
    }

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

    }
```

```
    public void PickSong(){
        UIMainMenu.active = false;
        UIPickSong.active = true;
        StartCoroutine(ShowLoadDialogCoroutine());
    }

    public void CloseGameAndShowScore(){
        Destroy(converter.gameObject);
        UIScoreBoard.SetActive(true);
        TotalScoreLabel.text = lastScore.ToString();
        AccLabel.text = ((lastScore / (lastTotalNotes * 300))
* 100).ToString() + "%";
        PerfectNotesLabel.text = lastPerfectNotes.ToString();
        GoodNotesLabel.text = lastGoodNotes.ToString();
        MissedNotesLabel.text = lastMissedNotes.ToString();
        Destroy(Baqueta1);
        Destroy(Baqueta2);

LeftHand.GetComponent<UnityEngine.XR.Interaction.Toolkit.Actio
nBasedController>().enableInputActions = true;

RightHand.GetComponent<UnityEngine.XR.Interaction.Toolkit.Acti
onBasedController>().enableInputActions = true;

    }

    public void BackToMenu(){
        UIScoreBoard.SetActive(false);
        UIMainMenu.SetActive(true);
    }

    public IEnumerator ShowLoadDialogCoroutine()
    {
        yield return
FileBrowser.WaitForLoadDialog(FileBrowser.PickMode.FilesAndFol
ders, true, null, null, "Load Files and Folders", "Load");
```

```
        if (FileBrowser.Success)
        {
            path = FileBrowser.Result[0];
            Baqueta1 = Instantiate(BaquetaPrefab,
Baqueta1Pos);
            Baqueta1.name = "Baqueta1";
            Baqueta2 = Instantiate(BaquetaPrefab,
Baqueta2Pos);
            Baqueta2.name = "Baqueta2";
            StartGame();
        }

    }

    public void StartGame(){
        var newObj = Instantiate(MapPrefab, transform);
        converter =
newObj.transform.Find("Map").GetComponent<DTXConverter>();
        converter.path = path;
        lastScore = 0;
        lastPerfectNotes = 0;
        lastGoodNotes = 0;
        lastMissedNotes = 0;
        LeftRay.SetActive(false);
        RightRay.SetActive(false);
        Baqueta1.SetActive(true);
        Baqueta2.SetActive(true);
    }

    public void checkStart(){
        if (Baqueta1Grabbed && Baqueta2Grabbed){
            converter.startGame();
        }
    }

    public static Singleton GetInstance()
```

```
    {
        return inst;
    }
}
```

TravelingLight.cs

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

public class TravelingLight : MonoBehaviour
{
    [ColorUsage(true, true)]
    public Color Perfect;
    [ColorUsage(true, true)]
    public Color Good;
    [ColorUsage(true, true)]
    public Color Miss;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
            transform.Translate(Vector3.Scale(new Vector3(0,
0, -40), (transform.parent.transform.localScale) ) *
Time.deltaTime);
            if (transform.localPosition.y < -80)
Destroy(gameObject);
    }
}
```

ButtonVR.cs

```
/*****************************************************
```

```
/*Copyright : Copyright (c) RealaryVR. All rights reserved.
Description: Script for VR Button functionality.
*****************************************************/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class ButtonVR : MonoBehaviour
{
    public GameObject button;
    public UnityEvent onPress;
    public UnityEvent onRelease;
    AudioSource sound;
    public GameObject cage;
    bool isPressed1;
    bool isPressed2;

    void Start()
    {
        sound = GetComponent<AudioSource>();
        isPressed1 = false;
        isPressed2 = false;
    }

    private void OnTriggerEnter(Collider other)
    {
        if (!isPressed1 && other.name.Equals("Baqueta1") &&
other.gameObject.GetComponent<Rigidbody>().velocity.y < -0.1)
        {
            onPress.Invoke();
            if (sound != null) sound.Play();
            isPressed1 = true;
        }
        if (!isPressed2 && other.name.Equals("Baqueta2") &&
other.gameObject.GetComponent<Rigidbody>().velocity.y < -0.1)
```

```
        {
            onPress.Invoke();
            if (sound != null) sound.Play();
            isPressed2 = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.name.Equals("Baqueta1"))
        {
            onRelease.Invoke();
            isPressed1 = false;
        }
        if (other.name.Equals("Baqueta2"))
        {
            onRelease.Invoke();
            isPressed2 = false;
        }
    }
}
```

12. Manual d'usuari

12.1 Requisits previs

Cal tenir instal·lat el software d'Oculus per a poder utilitzar les ulleres Quest 2. Aquestes han d'estar connectades al PC mitjançant un cable USB-C, preferiblement un d'almenys 3 metres de longitud, i han d'estar executant l'aplicació "Oculus Link". Cal tenir també uns pedals USB connectats al mateix PC que les ulleres. Aquests s'han de configurar de la següent manera:

- Pedal esquerre: Continuous trigger, tecla Shift esquerre
- Pedal dret: Single Trigger, tecla 5

12.2 Manual d'instal·lació

El projecte només funciona a ordinadors amb Windows. Per a poder jugar, cal

seguir els següents passos:

1. Descomprimir l'arxiu Zip "TFG_JoanTiscar.zip"
2. Accedir al directori TFG
3. Executar el fitxer TFG.exe

12.3 Manual d'usuari

Dintre del menú principal, el jugador ha d'apuntar amb els raigs que surten de les mans per a interactuar amb la interfície. Un cop seleccionada l'opció Play, cal navegar fins a la carpeta on es troba el fitxer DTX que es vol jugar.
Un cop triat, cal agafar les dues baquetes i es pot començar a jugar.
Cal tocar els instruments quan les notes hi siguin justament al damunt.
Un cop s'acaba la cançó, es carregarà la pantalla de puntuacions. Si es vol continuar jugant, cal prémer el botó de tornar al menú principal.

12.4 On obtenir fitxers .dtx

No existeix una pàgina centralitzada on es pugen els fitxers .dtx, però hi ha diferents pàgines on diferents persones pugen els fitxers que creen. La millor forma de trobar aquestes és utilitzant Google. A continuació indiquem una sèrie de cerques que ens han donat bons resultats:

- dtxmania simfiles
- dtxmania songs
- dtxmania maps
- dtxmania charts