Simulació de personatges autònoms en entorns de realitat virtual

Treball de fi de grau realitzat a la

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) – BarcelonaTech

per

Daniel Mateu Tudela

Grau d'Enginyeria Informàtica

Computació

Directora: Nuria Pelechano Gomez - Ciències de la Computació

Codirector: Alejandro Ríos Jerez - Ciències de la Computació

Barcelona, Juny 2017

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Agraïments

M'agradaria agrair en primer lloc als professors Alejandro Ríos i Nuria Pelechano per tot el seu suport constant, ajuda i coneixements que m'han ajudat tant en la realització del projecte com moralment.

Agrair també a Jordi Moyés per la seva assistència tècnica que necessitava i m'ha proporcionat.

Finalment, agrair als meus pares per donar-me l'oportunitat de realitzar aquest estudis, ja que sense ells no podria haver arribat fins aquí.

Abstract

This project aims to create a 3D virtual reality environment set in a train station to carry out studies on the reaction of users to a situation of alarm without notice. The user will get familiar to this virtual environment by performing some tasks that will have been assigned previously.

For the development of this project we have studied the features offered by Unity to create virtual environments. We have also studied all the features needed for programming the functions that are involved in the animation and behaviour of the characters and the user's avatar. In addition, the tools that Unity provides have been used to improve the visualization of the environment.

We have also enhanced and extended the basic features offered by Unity in order to further improve the movement of the characters, or the performance among others.

Resum

En aquest projecte es proposa crear un entorn 3D de realitat virtual ambientat en una estació de tren per poder realitzar estudis sobre la reacció dels usuaris davant d'una situació d'alarma sense previ avís. L'usuari s'adaptarà a aquest entorn virtual mitjançant la realització d'unes tasques que se li hauran assignat prèviament.

Per al desenvolupament d'aquest projecte s'han estudiat els mecanismes que disposa Unity per realitzar entorns virtuals. També hem estudiat totes les característiques necessàries per a la programació de les funcionalitats implicades en l'animació i el comportament dels personatges autònoms i del propi usuari. S'han aprofitat, a més, les eines que proporciona Unity per a la millora de la visualització de l'entorn.

També hem millorat i ampliat les característiques bàsiques que ofereix Unity per tal de millorar el moviment i el rendiment dels personatges, entre d'altres.

Resumen

En este proyecto se propone crear un entorno 3D de realidad virtual ambientado en una estación de tren para poder realizar estudios sobre la reacción de los usuarios delante de una situación de alarma sin previo aviso. El usuario se adaptará a este entorno virtual mediante la realización de unas tareas que se le habrán asignado previamente.

Para el desarrollo de este proyecto se han estudiado los mecanismos que dispone Unity para realizar entornos virtuales. También hemos estudiado todas las características necesarias para la programación de las funcionalidades implicados en la animación y el comportamiento de los personajes autónomos y del propio usuario. Se han aprovechado, además, las herramientas que proporciona Unity para la mejora de la visualización del entorno.

También hemos mejorado y ampliado las características básicas que ofrece Unity con tal de mejorar el movimiento y rendimiento de los personajes, entre otros.

Índex de contingut

Índex de figures

Índex de taules

1. Context

1.1. Introducció

Avui en dia, la simulació és un procés molt utilitzat en qualsevol àmbit degut a la gran complexitat dels problemes i la gran quantitat de variables que hi poden haver. Per exemple, en l'àmbit de les ciències químiques és utilitzat per saber el comportament i la reacció que hi ha entre diferents molècules, o últimament també s'ha estat utilitzant, aprofitant els nous sistemes de realitat virtual, per saber la reacció que tenim les persones davant de diferents situacions.

En aquest projecte es desenvoluparà un entorn 3D de realitat virtual ambientat en una estació de tren i diferents models que intentaran simular de forma realista el comportament de persones en aquest entorn. Un cop efectuat, s'introduirà una seqüència per simular l'evacuació de l'estació i veure el comportament de l'usuari immers en aquest entorn.

Cal destacar que es parteix d'aquest projecte amb una base, és a dir, un model senzill sense texturitzar d'una estació de tren, un model sense animació que representen els trens i diversos models que representen les persones que es troben en l'estació amb una IA senzilla de moviment (moure's cap a un punt determinat).

A més, pel desenvolupament utilitzarem Unity [1], un motor gràfic molt utilitzat a dia d'avui i que es coneix com a una de les plataformes més utilitzades per la creació de videojocs multiplataforma tant en 2D com en 3D, majoritàriament en el darrer.

Unity treballa principalment amb els llenguatges C# i javascript, tot i que ens centrarem amb el primer d'ells degut a què el coneixement previ d'aquest és més alt.

1.2. Actors implicats

En primer lloc, tenim el tutor/a o tutors/es del projecte. En el meu cas, la tutora del projecte és la Nuria Pelechano i el co-director serà l'Àlex Ríos, que guiaran a través d'aquest projecte amb l'ajuda que necessiti la persona que s'encarrega de desenvolupar-lo.

En segon lloc, tenim el desenvolupador o desenvolupadors del projecte, que en aquest cas seré només jo al tractar-se d'un projecte individual.

Per finalitzar, tenim a l'usuari o usuaris que tinguin l'oportunitat de provar el sistema de realitat virtual implementada en el projecte i puguin opinar sobre aquest.

1.3. Tecnologies utilitzades

A continuació, s'explicaran a grans trets les tecnologies utilitzades per a la realització del projecte.

1.3.1. Unity

Unity serà la plataforma principal per al desenvolupament de tot el projecte i és la que ens permetrà la creació de tot l'entorn 3D, de l'animació dels personatges i de l'usuari i de la creació de totes les funcionalitats necessàries per a què el sistema funcioni. [1]

Per aquells que no han utilitzat mai Unity, la seva estructura és la següent: La zona o zones on es treballa i es pot visualitzar tot el que es realitza s'anomenen escenes, i en ella es troben tots els objectes que incloure'm en l'entorn, com les estructures o els personatges. Tot objecte dins d'una escena s'anomena GameObject, i aquest està format per components. Les components són les propietats de l'objecte, com per

exemple la Transform, que ens indica la posició, rotació i escala en local space de l'objecte, o els scripts que serviran per descriure el comportament dels objectes.

L'eina que utilitza Unity per a l'animació s'anomena Mecanim. Aquesta eina proporciona un sistema d'animacions sofisticat que permet a l'usuari tenir un control quasi total dels moviments i els canvis que s'apliquen sobre els objectes animats. Aquest sistema s'haurà d'estudiar per poder aplicar-lo en el projecte i aconseguir els objectius que esmentarem en l'apartat 3.

Dins de Unity també podem trobar un sistema ja creat de navegació i pathfinding. Degut a la falta de temps i de coneixements de Unity, no es podrà crear un altre algorisme i s'estudiaran i s'utilitzaran les funcionalitats d'aquest per a moure els personatges a través de l'escena.

1.3.2. C#

El llenguatge C# és el que farem servir per a la creació dels scripts en Unity. Es tracta d'un llenguatge orientat a objectes on la seva sintaxis és semblant a llenguatges com C++ o Java. Com que C# és un llenguatge dissenyat pel framework .NET, s'utilitzaran llibreries com per exemple les següents:

- System.Collections: Conté classes útils com les cues o les ArrayList.
- System.IO: Conté classes que farem servir per l'emmagatzematge de dades.
- System.Runtime.Serialization: Conté classes que permeten transformar objectes o estructures a una línia de bytes que després es pot emmagatzemar i permeten fer també el pas invers.

També s'utilitzaran les llibreries pròpies de Unity que ens serviran per utilitzar les funcionalitats de l'entorn, com per exemple:

- UnityEngine.UI: Conté classes per treballar amb els layouts.
- UnityEngine.AI: Conté classes per utilitzar el sistema de navegació propi de Unity, la NavigationMesh.

2. Estat de l'art

2.1. Estudis anteriors

A continuació, es veuran més detalladament tots els conceptes principals que formen el nostre projecte i que han sigut objecte d'estudi en projectes i tesis anteriors.

2.1.1. Simulació de multituds

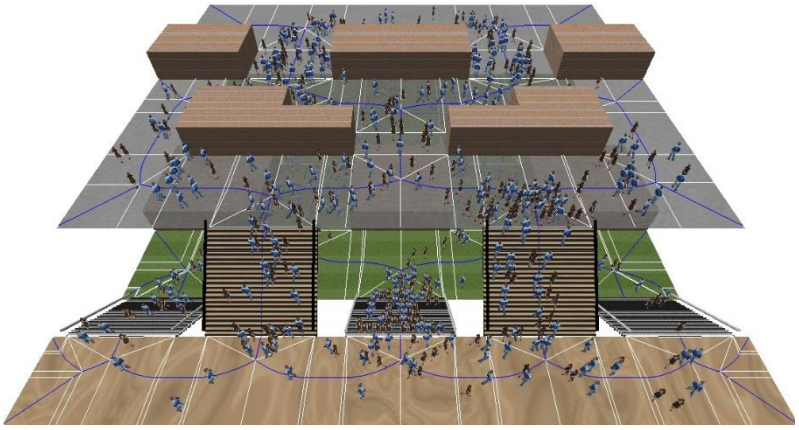


Figura 1: Crowd Simulation en un escenari 3D

La simulació de multituds, o també conegut com *Crowd Simulation*, ha estat un dels temes més estudiats en l'àmbit dels videojocs i també és molt habitual trobar-s'ho en escenes virtuals de pel·lícules. És el procés de simular el moviment o el comportament d'un gran nombre d'entitats en un entorn virtual [2].

Degut a què el nombre d'entitats en una multitud pot variar, es poden distingir principalment dos tipus de models en la simulació de multituds. D'una banda tenim els models que intenten simular el comportament individual de cada individu. Aquest tipus de models s'anomenen models microscòpics. D'altra banda tenim els models que intenten simular el comportament d'un grup d'entitats com a un conjunt sencer, coneguts com els models macroscòpics [3]. El model que s'adaptarà a la realització del projecte a l'hora de la implementació de la simulació dels models de les persones serà un model microscòpic, ja que ens interessa que cada persona es comporti de manera diferent en qualsevol moment donat i d'aquesta manera ens aporti una major sensació de realisme.

Com ja hem comentat anteriorment, s'utilitzarà Unity com a la eina principal de desenvolupament del projecte. Per a la simulació de multituds, en Unity es farà ús d'uns components anomenats "colliders" [4], que ens ajudaran per la detecció de col·lisions i forces externes que entrin en contacte amb l'objecte que es trobi a dins d'aquest component. Aquest component és molt útil per la simulació de trajectòries, un dels temes més importants en la simulació de multituds.

En la simulació de trajectòries, però, apareixen diversos problemes que es tractaran en el projecte com poden ser el "foot sliding", que es produeix quan la velocitat de l'animació no correspon a la distància que avança el model, o el salt abrupte entre animacions, provocant que els moviments no siguin naturals [5].

2.1.2. Realitat virtual

Quan parlem de realitat virtual ens referim a la tecnologia que utilitza un software i un hardware específics per generar imatges, sons i diverses sensacions per intentar simular un ambient real i donar a l'usuari una sensació d'immersió en l'entorn més profunda que quan ens trobem davant d'una pantalla [6].

La realitat virtual, com en la simulació de multituds, s'ha desenvolupat últimament bastant en l'àmbit dels videojocs. D'aquesta manera, Unity ha facilitat als usuaris l'adaptació dels seus projectes a la realitat virtual aportant nous mètodes de navegació i renderització ja inclosos en l'editor. Tanmateix, ha donat suport al desenvolupament dels drivers necessaris per a la importació i exportació dels projectes als diferents dispositius que permeten la reproducció dels projectes realitzats en realitat virtual.

L'avantatge principal de la realitat virtual és que permet la realització d'experiments sense posar en risc la vida de l'usuari, si es dona el cas, i permet veure com interacciona l'usuari amb l'entorn i els objectes que té al seu voltant.

2.2. Conclusions

En general, la majoria de treballs de l'actualitat que estudien el comportament de les persones immerses en multituds virtuals es centren principalment en els problemes de moviment local, com per exemple el fet de realitzar trajectòries entre dos punts intentant evitar xocar amb la gent. Hi ha altres treballs que es centren en l'estudi del comportament de masses, com per exemple l'evacuació d'un edifici en funció de quanta gent coneixia l'edifici, quanta tendia a seguir a altres persones i quanta es dedicava a donar voltes intentant trobar la sortida [7]. Però mai s'ha fet un estudi de si realment això és el que passa quan ens veiem immersos en aquesta situació, per la qual cosa ens centrarem en la creació de l'entorn i els personatges autònoms de manera que puguem dirigir el seu comportament per preparar experiments en realitat virtual que ens permetin estudiar això.

3. Formulació del problema

3.1. Objectius

Degut a què el desenvolupament d'un entorn 3D i la simulació de models requereix d'una feina considerable, caldrà doncs organitzar-la i determinar quins són els objectius principals a seguir per poder assolir-ho.

Els principals objectius seran els següents:

1. Aconseguir que l'entorn sigui el més realista possible per augmentar la immersió de l'usuari.
2. Aconseguir una bona sensació de presència de l'usuari millorant aspectes que influeixin directament amb ell.

Capturar la major quantitat d'informació per a què els experiments que es facin a posteriori amb aquest entorn siguin més precisos i clars.

3.2. Assignatures relacionades de l'especialitat

Mirant a fons la naturalesa del treball, en primer lloc una de les assignatures que més es relaciona i ha aportat al treball és Gràfics, ja que m'ha ajudat a entendre molt millor els conceptes bàsics del comportament dels objectes en les tres dimensions, com per exemple el moviment, les rotacions o les animacions, en les que es treballa amb Unity. Tanmateix, ha resultat molt útil gràcies a la teoria sobre il·luminacions i textures, on també es veu aplicada en aquest motor gràfic.

D'altra banda, les assignatures d'Algorismia i Intel·ligència Artificial també han sigut una aportació, ja que m'han ajudat a comprendre les estructures de dades que calien utilitzar en diferents casos, com per exemple a l'hora d'aplicar el sistema de trens o de cues en les màquines de bitllets de l'escena i, d'altra banda, a la millora del local avoidance per al sistema de pathfinding de Unity.

Per últim, crec que l'assignatura de Teoria de la Computació ha sigut útil a l'hora de realitzar les màquines d'estat del comportament de les persones que es troben dins l'entorn.

3.3. Relació del projecte amb les CT i el nivell d'assoliment

Les competències tècniques juntament amb el seu nivell són les següents:

- **CCO1.1**: Avaluar la complexitat computacional d'un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d'acord amb els requisits establerts. [En profunditat]
- **CCO1.3**: Definir, avaluar i seleccionar plataformes de desenvolupament i producció hardware i software per al desenvolupament d'aplicacions i serveis informàtics de diversa complexitat. [En profunditat]
- **CCO2.6**: Dissenyar i implementar aplicacions gràfiques, de realitat virtual, de realitat augmentada i videojocs. [En profunditat]
- **CCO3.1**: Implementar codi crític seguint criteris de temps d'execució, eficiència i seguretat. [Una mica]

En primer lloc, degut a què s'ha treballat un algorisme de local avoidance sobre el pathfinding de Unity, també es necessitarà conèixer les seves propietats, i això inclou el seu rendiment. Aquest punt és important ja que s'ha implementat un codi de local avoidance que millora la col·lisió entre agents i s'havia de vigilar la seva eficiència per a què no afectés el rendiment de l'aplicació i, d'aquesta manera, disminuir la immersió de l'usuari. (CCO1.1)

Per a la realització del projecte, s'han estudiat les possibilitats de realitzar-lo mitjançant diferents motors gràfics en funció de les propietats que requeria, però finalment s'ha decidit realitzar-lo en Unity principalment degut als coneixements previs i les seves funcionalitats, entre d'altres. (CCO1.3)

La següent competència és imprescindible, ja que una de les característiques principals del projecte serà exportar-lo a la realitat virtual i es tracta bàsicament de la realització d'una aplicació gràfica. (CCO2.6)

Una part del codi, en menor mesura, serà crític, i aquest està relacionat amb l'activació i desactivació de components a l'escena en tems d'execució, ja que un mal ús d'aquests pot provocar que l'aplicació deixi de funcionar si un script necessita d'un component concret i aquest es troba desactivat. (CCO3.1)

3.4. Justificació a l'especialitat de Computació

La primera relació del projecte amb l'especialitat de computació, i la més obvia segurament, és el fet de què es tracta principalment d'una aplicació gràfica i es treballen temes relacionats amb l'àmbit de gràfics, com la il·luminació i els comportaments en entorns 3D, essent aquest un dels temes tractats en l'especialitat.

A més a més, dins de l'entorn 3D es crearan models de persones que simularan, en part, la intel·ligència humana, és a dir, la capacitat de triar diverses accions tals com caminar, anar a comprar, marxar de l'estació o pujar a un tren. Tot i això, cal dir que no s'implementarà cap sistema perquè aquests puguin aprendre o raonar degut a la seva complexitat que ens porta a un excés de temps.

Per acabar de reforçar els lligams, tot el codi implementat es farà seguint unes estructures de dades sofisticades i s'avaluaran altres alternatives ja que ens interessa de cara a la sensació de presència en la realitat virtual.

4. Abast

4.1. Abast del projecte

Per tal de veure l'abast possible del projecte, es farà una descripció més detallada de tots els processos que es duran a terme per tal d'assolir la fita final.

En primer lloc ens centrarem en la millora de l'entorn 3D en el que es trobarà l'usuari perquè es trobi en un ambient més realista i que es vegi més immers en aquest. Per tal d'aconseguir-ho, es triaran uns models d'estructures adequats juntament amb unes textures realistes. També es procurarà que hi hagi una quantitat adequada d'elements decoratius en l'entorn perquè no hi hagi la sensació d'una escena buida. Es tindran en compte els detalls típics d'una estació, com els avisos d'arribada i de sortida dels trens, els sons de l'exterior, els sons dels propis trens, el soroll de la gent, etc. A més, s'estudiaran les propietats que ens proporciona l'eina de Unity per els càlculs de la il·luminació i se li aplicaran els canvis que facin falta per millorar el realisme de l'escena.

Un cop efectuada la millora en l'escena, s'utilitzaran diverses eines per aplicar noves animacions als models que representaran els personatges autònoms de l'estació. Degut a la manca de temps, no serà possible la realització d'animacions pròpies ja que es tracta d'un tema d'estudi molt ampli i requereix d'uns coneixements i d'una pràctica prèvia majors al temps que hi ha disponible. Llavors, el que es durà a terme serà l'aplicació d'animacions ja creades per altres usuaris disponibles en la xarxa. Una de les eines que ens permet afegir animacions a models humanoides ja creats és Mixamo [8], que s'encarrega del rigging del model (el procés en el qual se li assignen pesos als vèrtexs de la malla de l'humanoide respecte als ossos de l'esquelet als que està associat per poder efectuar deformacions suaus d'aquesta i poder aplicar-li animacions al personatge) i ens dona un ampli ventall d'animacions per afegir-les als models.

Quan s'hagin afegit noves animacions als models, s'exportaran a Unity i des d'allà aprofitarem les eines que ens donen per afegir les transicions entre les animacions. En

aquest cas, s'utilitzarà una eina que incorpora anomenada Mecanim, que ens permet, a més d'afegir transicions entre les animacions i activar els estats corresponents a partir de codi, afegir estats que representen un "blend tree". Un blend tree ens permet barrejar dues o més animacions (o fins i tot barrejar animacions amb altres blend trees formats per altres animacions) per tal de què no hi hagi salts sobtats entre aquestes efectuant un procés d'interpolació lineal [9]. S'estudiarà en profunditat les característiques que ens proporcionen aquest tipus d'estats per tal d'augmentar la fluïdesa entre les animacions. Després d'aplicar les animacions, es crearà un diagrama d'estats per aquest personatges i es programaran tots els estats per tal de què efectuïn diverses accions i no es vegin estàtics, d'aquesta manera augmentant el realisme.

Un cop modificades les animacions i el comportament dels personatges, s'afegiran animacions als trens i a certs elements de l'entorn aprofitant una altra de les característiques que ens proporciona Mecanim per crear animacions als objectes. Aquesta eina permet afegir un estat inicial (amb posició i rotació inicials) i un estat final, i efectua una interpolació d'aquestes dues posicions, calculant totes les posicions intermèdies i creant d'aquesta manera un efecte de moviment en l'objecte. A més, per la navegació dels models dels personatges a través de l'estació s'utilitzarà una eina anomenada Navigation Mesh, que permet moure una entitat anomenada Navigation Mesh Agent a través de la zona creada amb aquesta eina. El principal problema es troba quan hi ha una gran quantitat de NMAs, ja que provoca diferents errors. Per aquest motiu, s'estudiarà a fons aquest tema i, degut a què no es pot modificar directament el codi d'aquest sistema, es crearà un sistema extern que permeti calibrar i intentar, en la mesura del possible, arreglar els errors que provoca el sistema original.

Un cop solucionat el moviment dels objectes de l'entorn i el dels models dels personatges, s'afegirà un sistema d'evacuació per tal d'evacuar l'estació de tots els personatges autònoms i parar tot el sistema dels trens.

Després d'afegir el sistema d'evacuació, es crearà un sistema de moviment per l'usuari previ a la ampliació de la realitat virtual per tal de poder enregistrar les seves accions i moviments i carregar-les en un visualitzador 2D amb diverses opcions.

Finalment, s'exportarà tot el projecte a la realitat virtual. Es modificaran tots els paràmetres i es faran els canvis necessaris per tal de què el projecte s'exporti correctament.

4.2. Possibles obstacles

Els principals obstacles que ens podrem trobar a mesura que anem avançant en el projecte són diversos. En primer lloc, i el més obvi, són els errors que es puguin tenir en el codi, tant en l'estructura com en l'eficiència. Per tal de minimitzar aquests errors, es faran tests cada cop que s'implementa alguna funcionalitat nova.

A més de l'eficiència del codi, tenim l'eficiència en la pròpia escena per la quantitat d'objectes que es trobin en aquell moment. Per això, s'hauran d'anar fent proves cada cop que es faci un canvi en aquesta i es comprovarà que tot funciona amb la fluïdesa adequada.

Degut a què el tema de la realitat virtual és un tema recent i no s'ha desenvolupat al màxim, caldrà anar fent proves per veure que tot funciona correctament i els resultats siguin els esperats.

Per últim s'ha de tenir en compte que Unity està orientat principalment cap a la creació de videojocs, i això pot suposar un obstacle degut a què les funcionalitats estan pensades i optimitzades per aquest àmbit i no per a la simulació de multituds, com per exemple el local avoidance que porta incorporat en el pathfinding.

5. Metodologia i rigor

Per tal d'assolir els objectius esmentats del projecte en el temps limitat de què es disposa, en primer lloc es mantindrà un seguiment força actiu d'aquest i es realitzarà una reunió amb els directors del projecte cada dues setmanes per tal de facilitar la comunicació sobre els temes treballats, en procés i a realitzar. De totes formes, per millorar el feedback sempre hi haurà la comunicació via correu electrònic.

Per les tasques del projecte, s'utilitzarà una eina anomenada Trello [10], que permet organitzar la feina per "sprints", és a dir, tasques a realitzar a curt termini, i permet un control actiu a distància del projecte.

Les eines a utilitzar i ja esmentades anteriorment seran Unity per el desenvolupament principal del projecte, Blender [11] o Sketchup [12] en cas que es necessiti modificar algun model de l'escena i Mixamo per les animacions dels models dels personatges autònoms. En Unity, el codi es farà en el llenguatge C#, un llenguatge orientat a objectes i funcional, amb característiques semblants a C++ i Java, juntament amb les extensions que ens proporciona el propi editor Unity. S'ha decidit triar Unity degut a que es un dels motors gràfics més utilitzats en l'àmbit acadèmic i en el món laboral.

Per tant, la metodologia que s'adapta millor a les propietats esmentades anteriorment i que seguirem en aquest projecte és la metodologia Scrum [13].

Finalment, per comprovar si l'entorn creat efectua correctament la seva funcionalitat, el més adequat seria provar-ho amb diverses persones i rebre el seu "feedback" per tal de veure que l'entorn crear és l'adequat, o requereix d'algun element més per acabar d'adquirir una bona sensació de presència dins de l'entorn.

D'altra banda, per comprovar que tot funciona correctament internament, s'hauria de fer una simulació completa i veure pas per pas cada sistema afegit dins del programa, com per exemple una comprovació del sistema dels trens, veient el comportament d'aquests davant de certes situacions (si hi ha una persona esperant, o si n'hi ha més de la capacitat total que pot suportar el tren, per exemple).

Després de treballar amb Trello durant les primeres setmanes, s'ha decidit canviar la plataforma per realitzar els sprints a ScrumMe. Les dues plataformes tenen la mateixa finalitat, però hem trobat que la segona ens proporciona funcionalitats per facilitar-nos la creació de sprints, per aquest motiu hem passat tota la informació creada de Trello a ScrumMe.

Respecte a la metodologia proposada, es segueixen realitzant les reunions cada dues setmanes com es va indicar al principi i s'ha incrementat el feedback a través de correu electrònic.

6. Planificació

6.1. Tasques

A continuació, es definiran les tasques a nivell general que es seguiran durant tot el desenvolupament del projecte. Cal remarcar que , degut a la naturalesa del projecte i les condicions prèvies de les tasques, l'ordre en què s'implementen algunes d'elles no importa. En el cas de les tasques 4, 5 i 6, es podrien realitzar en qualsevol ordre ja que no hi ha dependències entre elles gracies a la manera en què es treballa amb Unity, però es faran en aquest ordre ja que d'aquesta manera es segueix un ordre lògic i intuïtiu de les tasques. En el cas de la tasca 4, es tracta d'una tasca que es realitzarà durant tot el projecte segons es vagi necessitant degut a què es necessita una gran quantitat de material i es requereix en varies parts de les demés tasques.

Com que la metodologia que seguim en aquest projecte és la Scrum i requereix un seguiment actiu amb reunions cada dues setmanes, pot ser que apareguin diverses desviacions en els objectius establerts en aquest període. Tot i això, no afecta a les tasques a nivell general.

1. Planificació del projecte.
2. Preparació de l'entorn on treballarem i de les eines a utilitzar (Unity, Mixamo)
3. Estudiar les característiques i les funcionalitats que ens proporciona l'eina principal per les animacions dins de Unity, Mecanim.
4. Cerca de material necessari per al disseny de l'entorn i la creació d'animacions.
5. Implementació del codi i les estructures del comportament dels agents.
6. Implementació del codi per a la millora del local avoidance del pathfinding.
7. Assignació de tasques i navegació de l'usuari.
8. Estudiar les característiques de les eines per l'exportació a la realitat virtual.
9. Redacció de la memòria final.

6.2. Temps

El temps del qual disposem per la realització del projecte és de 4 mesos i tres setmanes, incloent setmana santa. És a dir, del 15 de febrer al 29 de juny.

Degut a les possibles desviacions, la duració del projecte és una estimació real. Les hores que ocuparan cada tasca es veuen reflectides en el següent quadre:

Tasca a realitzar	*Duració (en hores)*
Planificació del projecte	90 h
Preparació del entorn i eines	10 h
Estudi de l'eina Mecanim	30 h
Cerca de material	40 h
Implementació comportament agents	100 h
Implementació codi millora pathfinding	65 h
Assignació de tasques i navegació de l'usuari	80 h
Estudi eines exportació VR	80 h
Redacció memòria final	35 h
	530 h

Taula 1: Temps de cada tasca general

Amb aquesta distribució de temps a les tasques s'assoleixen un total de 530 hores.

6.3. Recursos

En el projecte s'han utilitzat diferents eines tant de software com de hardware especificades a continuació.

Per al software, les eines que utilitzarem seran: Unity, Mixamo, Trello, Blender, Sketchup, Adobe Character Generator, MS Office, Windows 10.

Per al hardware, les eines seran les següents: PC, Cascos VR, PC Controller (a determinar).

6.4. Diagrama de Gantt previ

	Nombre de la tarea	Inicio	Finalizar	Duración
1	Planificació del projecte	20/02/17	27/03/17	36d
2	Definició de l'abast i contextualitzaci	20/02/17	28/02/17	9d
3	Planificació temporal	01/03/17	06/03/17	6d
4	Gestió econòmica i sostenibilitat	07/03/17	13/03/17	7d
5	Presentació preliminar	14/03/17	20/03/17	7d
6	Bloc d'especialitats	21/03/17	27/03/17	7d
7	Document final	21/03/17	27/03/17	7d
8	Powerpoint presentació final	21/03/17	27/03/17	7d
9	Preparació de l'entorn i eines	12/02/17	13/02/17	2d
10	Unity	12/02/17	12/02/17	1d
11	Mixamo	12/02/17	12/02/17	1d
12	Manual Unity	13/02/17	13/02/17	1d
13	Manual C#	13/02/17	13/02/17	1d
14	Estudi de l'eina Mecanim	14/02/17	21/02/17	8d
15	Cerca d'informació	14/02/17	17/02/17	4d
16	Anàlisis de la informació	18/02/17	21/02/17	4d
17	Cerca de material	22/02/17	22/06/17	121d
18	Anàlisis	22/02/17	22/06/17	121d
19	Filtre	22/02/17	22/06/17	121d
20	Tests	22/02/17	22/06/17	121d
21	Implementació comportament agen	23/02/17	31/03/17	37d
22	Estudi i Disseny	23/02/17	24/02/17	2d
23	Implementació	25/02/17	28/03/17	32d
24	Tests	29/03/17	31/03/17	3d
25	Implementació codi millora pathfinc	01/04/17	25/04/17	25d
26	Estudi i Disseny	01/04/17	02/04/17	2d
27	Implementació	03/04/17	22/04/17	20d
28	Tests	23/04/17	25/04/17	3d
29	Assignació de tasques i navegació	26/04/17	24/05/17	29d
30	Estudi i Disseny	26/04/17	27/04/17	2d
31	Implementació	28/04/17	21/05/17	24d
32	Tests	22/05/17	24/05/17	3d
33	Estudi eines exportació VR	25/05/17	22/06/17	29d
34	Cerca d'informació	25/05/17	26/05/17	2d
35	Anàlisis de la informació	27/05/17	29/05/17	3d
36	Exportació del projecte	30/05/17	22/06/17	24d
37	Redacció memòria final	22/06/17	28/06/17	7d

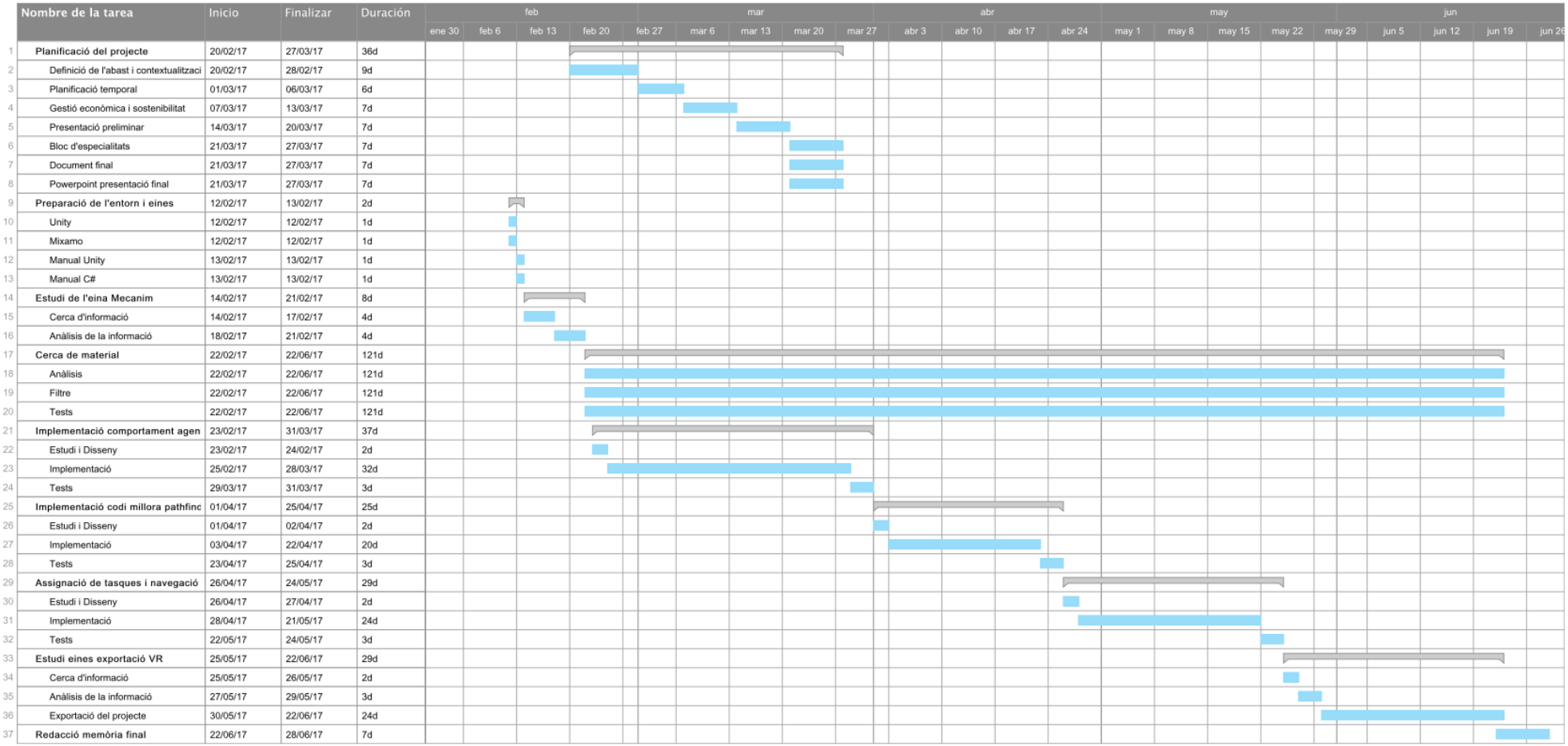


Figura 2: Diagrama de Gantt previ

6.5. Diagrama de Gantt final

	Nombre de la tarea	Inicio	Finalizar	Duración
1	Planificació del projecte	20/02/17	27/03/17	36d
2	Definició de l'abast i contextualitzaci	20/02/17	28/02/17	9d
3	Planificació temporal	01/03/17	06/03/17	6d
4	Gestió econòmica i sostenibilitat	07/03/17	13/03/17	7d
5	Presentació preliminar	14/03/17	20/03/17	7d
6	Bloc d'especialitats	21/03/17	27/03/17	7d
7	Document final	21/03/17	27/03/17	7d
8	Powerpoint presentació final	21/03/17	27/03/17	7d
9	Preparació de l'entorn i eines	12/02/17	13/02/17	2d
10	Unity	12/02/17	12/02/17	1d
11	Mixamo	12/02/17	12/02/17	1d
12	Manual Unity	13/02/17	13/02/17	1d
13	Manual C#	13/02/17	13/02/17	1d
14	Estudi de l'eina Mecanim	14/02/17	21/02/17	8d
15	Cerca d'informació	14/02/17	17/02/17	4d
16	Anàlisis de la informació	18/02/17	21/02/17	4d
17	Cerca de material	22/02/17	22/06/17	121d
18	Anàlisis	22/02/17	22/06/17	121d
19	Filtre	22/02/17	22/06/17	121d
20	Tests	22/02/17	22/06/17	121d
21	Implementació comportament agen	23/02/17	31/03/17	37d
22	Estudi i Disseny	23/02/17	08/03/17	14d
23	Implementació	09/03/17	25/03/17	17d
24	Tests	26/03/17	31/03/17	6d
25	Implementació codi millora pathfinc	01/04/17	25/04/17	25d
26	Estudi i Disseny	01/04/17	08/04/17	8d
27	Implementació	09/04/17	19/04/17	11d
28	Tests	20/04/17	25/04/17	6d
29	Assignació de tasques i navegació	26/04/17	24/05/17	29d
30	Estudi i Disseny	26/04/17	05/05/17	10d
31	Implementació	06/05/17	19/05/17	14d
32	Tests	20/05/17	24/05/17	5d
33	Estudi eines exportació VR	25/05/17	22/06/17	29d
34	Cerca d'informació	25/05/17	28/05/17	4d
35	Anàlisis de la informació	29/05/17	04/06/17	7d
36	Exportació del projecte	05/06/17	22/06/17	18d
37	Redacció memòria final	22/06/17	28/06/17	7d

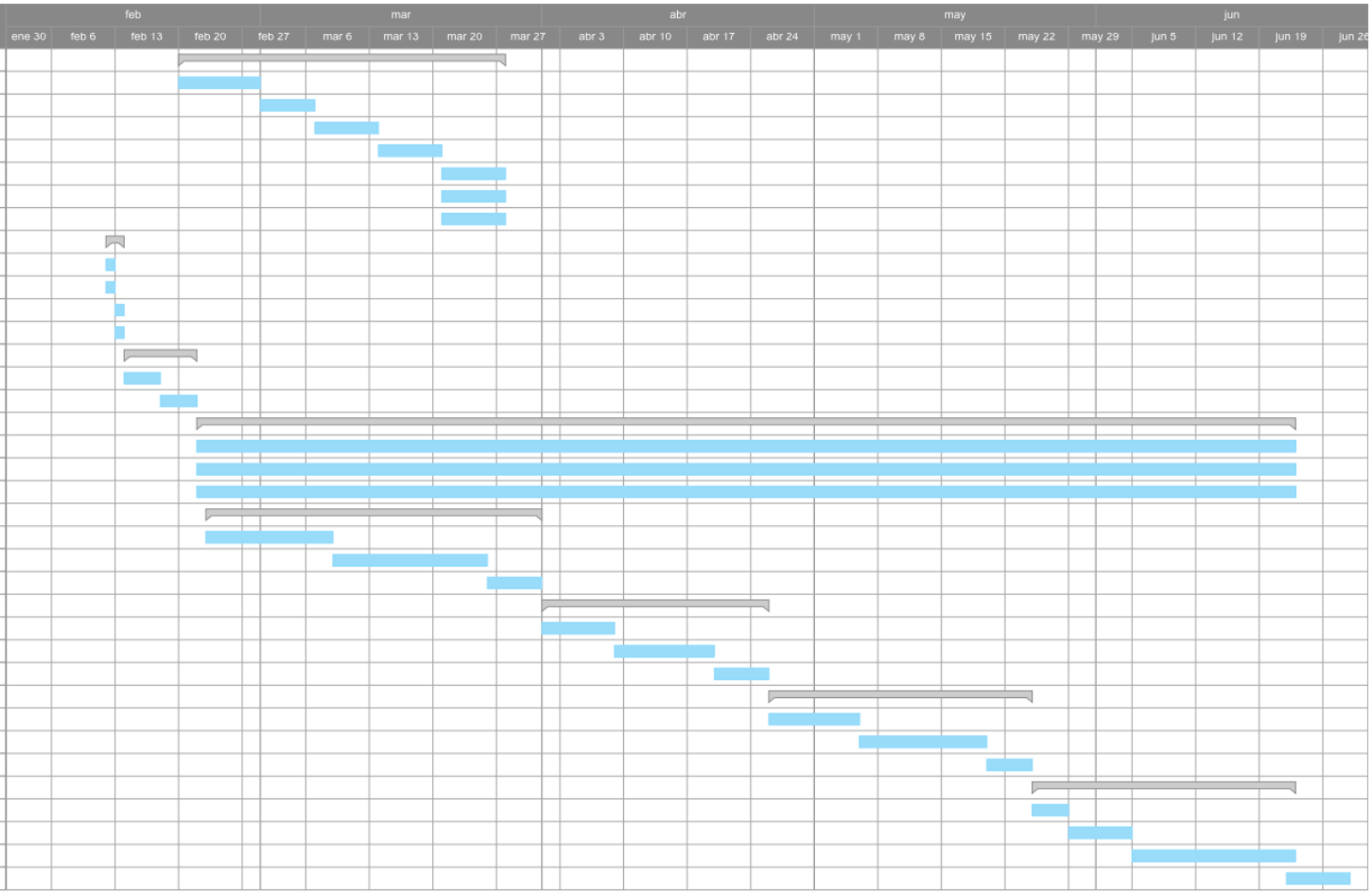


Figura 3: Diagrama de Gantt final

7. Valoració d'alternatives i pla d'acció

7.1. Possibles desviacions i solucions

En la realització del projecte, com ja hem comentat anteriorment, apareixen algunes desviacions eventuals que afecten la duració d'aquest, com poden ser augments en temps de cerca degut a la manca d'informació o augments en temps d'implementació i de proves degut als errors. Tot i això, com s'ha establert una metodologia Scrum, s'han realitzat reunions sovint i d'aquesta manera s'ha evitat un augment de temps total i s'han pogut gestionar els problemes amb poc temps.

Com que el temps que hi ha disponible és molt limitat i les duracions per a cada tasca poden variar, si no es pot acabar una tasca al complet per la manca de temps, s'ha assegurat que almenys tenim una versió bàsica i funcional d'aquesta per a què es pugui dur a terme la finalitat del projecte. Com a mínim, tindrem una versió funcional bàsica per a cada tasca indicada i l'exportació del projecte a la realitat virtual.

Si tenim que el projecte en total té una duració de 530 hores i el temps disponible és de 4 mesos i 3 setmanes (equivalent a unes 19 setmanes), això ens dona a unes 28 hores setmanals, cosa que és totalment assolible si tenim en compte els 7 dies de la setmana.

7.2. Canvis respecte la planificació inicial

Com es pot observar en els diagrames de Gantt dels apartats anteriors, no hi ha hagut cap canvi en la duració de les tasques generals, però sí en la duració de les tasques que deriven d'aquestes. En concret, en les tasques d'implementació del comportament dels agents, de la implementació del codi de millora del local avoidance del pathfinding i de l'assignació de les tasques i la navegació de l'usuari s'han vist modificades les tres parts de cada una d'elles.

Degut al poc coneixement previ de les funcionalitats de Unity, s'ha augmentat el temps en l'estudi i disseny i els tests d'aquestes. Per evitar que la duració total del projecte no

variés en gran mesura, s'ha ajustat aquest canvi i s'ha disminuït les hores dedicades a la part d'Implementació de les tres tasques principals.

També s'ha vist afectada i augmentada, en menor mesura, la quantitat d'hores dedicades a la cerca d'informació i el anàlisis de la informació dedicades a l'estudi d'eines de l'exportació a la realitat virtual. D'aquesta manera, també s'ha ajustat el temps total i s'ha reduït el nombre d'hores dedicades a l'exportació del projecte a la realitat virtual.

Aquest canvis efectuats no ha afectat per res ni en els objectius ni en el desenvolupament del projecte, ja que s'ha pogut desenvolupar fins ara el que s'havia previst en la planificació inicial.

Els costos del projecte s'han adaptat en la secció de recursos humans, ja que les hores dedicades a cada rol han canviat i, degut a què el preu per hora de cada rol són diferents, s'ha hagut d'adaptar el preu final a la nova planificació. Després de fer els càlculs corresponents, s'ha comprovat que el preu total en recursos humans ha augmentat en total 200€.

8. Identificació dels costos

Per la secció de la identificació dels costos del projecte, es tindran en compte totes les dades obtingudes dels anteriors lliurables. Els costos es divideixen en dos tipus: costos directes i indirectes. Per els costos directes, en el nostre cas tindrem costos per recursos humans, per hardware i per software. D'altra banda, tindrem uns costos indirectes que seran representats per despeses generals. Les amortitzacions aniran representades juntament amb els elements que el seu cost es regeix per la seva vida útil.

Degut a què el projecte encara està en desenvolupament, les dades a continuació són una previsió del pressupost total. Per tal que el pressupost final no sigui superior a la previsió, cada cop que es realitzi una de les tasques en el diagrama de Gantt s'actualitzaran les dades.

En el cas de l'estimació dels costos de recursos humans, es fa a nivell de les tasques indicades prèviament al diagrama de Gantt (ja que depèn directament de la duració de les tasques).

8.1. Recursos humans

Pel que fa als costos de recursos humans, aquests es divideixen segons els diferents rols que es troben presents en el projecte. Aquests són: Cap de projecte, analista o dissenyador, programador i beta tester. Com que aquest projecte només es duu a terme per una sola persona, aquesta s'encarregarà d'ocupar els 4 rols presents. A continuació, en l'estimació dels costos hi ha una descripció més detallada del temps consumit per cada un dels rols respecte al temps de cada una de les tasques generals de la planificació.

8.2. Hardware

En el cas del hardware, es tindran en compte els elements indicats en apartats anteriors. Aquests seran: l'ordinador principal on es desenvoluparà tot el projecte i els cascos de realitat virtual que s'utilitzaran un cop el projecte hagi estat exportat a VR juntament amb el controlador que permetrà moure's a l'usuari.

8.3. Software

Pel software, també es tindran en compte els elements indicats que són: Unity, Mixamo, Trello, Blender, Sketchup, Adobe Character Generator, MS Office i Windows 10.

8.4. Despeses generals

En el nostre cas, tindrem unes despeses generals que seran: la llum, paper i tinta que serviran per la utilització de l'eina principal de treball per el projecte i tota la documentació necessària que es vagi fent a mesura que s'avança.

9. Estimació dels costos

En aquest apartat s'indicarà amb taules detallades tots els costos per separat de cada un dels tipus de costos descrits anteriorment i, finalment, una taula amb una previsió dels costos totals del projecte.

9.1. Recursos humans

Rol	Preu per hora	Temps	Cost
Cap de projecte	55€/h	90h	4.950,0€
Analista/Dissenyador	35€/h	170h	5.950,0€
Programador	30€/h	180h	5.400,0€
Beta tester	25€/h	90h	2.250,0€
Total		530h	18.550,0€

Taula 2: Cost de recursos humans

El preu per hora s'ha obtingut a través de la xarxa i d'altres treballs.

El cap de projecte s'encarrega només de la planificació inicial del projecte.

L'analista o dissenyador s'encarregarà tant de la preparació de l'entorn i eines, com els estudis de l'eina de Mecanim i de l'exportació a VR. També aportarà a la cerca de material i les fases d'implementació.

El programador s'encarregarà principalment de les fases d'implementació i també intervindrà, si és necessari, en la cerca de material.

La funció principal del beta tester seran les comprovacions d'errors en les fases d'implementació i la redacció de la memòria final. Degut a què fer comprovacions a mesura que el programador va implementant, també intervindrà en part a les fases d'implementació, tot i que en menor mesura.

9.2. Hardware

Producte	Preu	Vida útil	Amortització
Ordinador (PC)	1250€	4 anys	82,8125€
Cascos VR HTC Vive	900€	3 anys	79,5€
Controlador PC	30€	3 anys	2,65€
Total	2.180€		164,9625€

Taula 3: Cost de hardware

L'amortització tant dels elements de hardware com de software s'ha calculat a partir de les dades d'aquest any (mitjana dies laborables per any i mitjana d'hores actives per dia laborable) obtingudes a través de la xarxa.

9.3. Software

Producte	Preu	Vida útil	Amortització
Unity	0€	-	0,0€
Mixamo	0€	-	0,0€
Trello	0€	-	0,0€
Blender	0€	-	0,0€
Sketchup	0€	-	0,0€
Adobe Character Generator	0€	-	0,0€
MS Office	69,00€/any	3 anys	6,095€
Windows 10	149,0€	3 anys	13,1617€
Total	218€		19,2567€

Taula 4: Cost de software

Els productes que es compren per subscripció, en aquest cas es considera subscripció única d'un any, per tant el càlcul total del preu es fa considerant que és de compra única.

9.4. Despeses generals

Producte	Preu	Quantitat	Cost
Llum	0,09839€/kWh	212kWh	20,8587€
Paper	10,27€/500 fulls	1 pack	10,27€
Tinta	24,97€	1	24,97€
Total	1.818€		56,0987€

Taula 5: Cost de despeses generals

Els preus de les despeses generals s'han extret de la xarxa. La quantitat ha estat calculada, en el cas de la llum, a partir de les hores totals que durarà el projecte i la quantitat mitjana de potència per hora que consumeix l'ordinador que utilitzarem per la realització del projecte. En el cas de la tinta i el paper, només són necessaris per la redacció de la fita inicial i la memòria final.

9.5. Costos totals

A partir dels costos obtinguts en les taules anteriors, podem calcular el cost total del projecte:

Tipus	Cost
Recursos humans	18.550,0€
Hardware	164,9625€
Software	19,2567€
Despeses generals	56,0987€
Total	18.790,3179€

Taula 6: Cost total

10. Control de gestió

Tot i que hi ha una planificació indicada, poden sorgir diverses desviacions en el projecte. Respecte als recursos de hardware, es pot assegurar que des d'un principi no es necessitaran més recursos dels que s'indiquen. Pel que fa al software, potser es requereix d'algun altre de més, però s'assegura que no afectarà al cost ja que si es necessita s'agafarà algun software d'ús gratuït. L'únic cost que es pot veure afectat és en els recursos humans, més específicament, a la part de implementació del codi.

Degut a què la implementació és una tasca que intervenen diferents rols, en la major part el programador, si sorgeix algun desviament es pot tornar a distribuir la feina per tal que no superi la previsió dels costos en recursos humans. Per això mateix, per evitar un augment en els costos, es farà servir el diagrama de Gantt per fer els ajustos necessaris a les tasques afectades i tornar a calcular les hores de cada rol.

11. Sostenibilitat

11.1. Econòmica

Tal i com hem pogut comprovar en els apartats anteriors, s'han pogut avaluar els costos tant directes com indirectes i a més, s'ha comprovat que el cost pels ajustaments en el projecte es pot evitar. Aquest projecte no requereix d'actualitzacions periòdiques amb noves funcionalitats, degut a què la seva finalitat és única. Tot i això, no treu que es pugui millorar en versions futures, com per exemple amb millores de rendiment, cosa que s'hauria d'afegir un cost addicional.

Degut a què el temps per realitzar el projecte és molt curt, és difícil que es pugui reduir el nivell de recursos i temps utilitzats, reduint d'aquesta manera el cost.

El temps dedicat a cada tasca està dividida segons la seva importància, en aquest cas, ens centrem més en el propi desenvolupament del codi ja que és el més important.

11.2. Social

La finalitat del projecte es centra principalment en l'estudi del comportament de les persones davant de certes situacions, en el nostre cas l'evacuació d'una estació. Actualment, les investigacions que es realitzen per estudiar aquest comportament esta augmentant gràcies a l'avanç que està tenint la realitat virtual, sobretot en els països desenvolupats on són capaços de permetre's aquesta tecnologia. Degut a la falta de sistemes per detectar aquest comportament, aquest projecte pot arribar a millorar la qualitat de vida no dels usuaris que utilitzin aquest projecte, ja que esta dedicat a l'àmbit d'investigació, sinó als nous sistemes de seguretat i d'evacuacions, per exemple, que es poden implementar després de saber com actuaran les persones davant d'aquestes situacions.

Com a aportació personal, la realització d'aquest projecte m'ha servit per assolir coneixements en nous àmbits i afrontar situacions que em puc trobar en un futur de cara a la realització d'altres projectes en l'àmbit laboral.

A més, actualment la immersió a la realitat virtual és una tecnologia en desenvolupament, i com s'ha pogut comprovar una exposició contínua i de duració

elevada pot afectar directament a la immersió de l'usuari provocant mareigs. És per això que cal tenir en compte aquest risc.

11.3. Ambiental

L'impacte d'aquest projecte sobre la part ambiental de la sostenibilitat és molt baixa, ja que els únics recursos que afecten al medi ambient són les despeses generals (els que més afecten són la llum i el paper), i la majoria de recursos necessaris són de software, cosa que no afecten en aquesta àrea. La quantitat de folis usats són gairebé insignificants, i la quantitat de llum utilitzada és menor a la que consumeix una persona normal.

En aquest cas, degut a què l'impacte ambiental del projecte és mínim, no hi ha cap risc a tenir en compte en la taula de sostenibilitat.

La matriu de sostenibilitat resultant és la següent:

Sostenible?	*PPP*	*Vida útil*	*Riscs*
Econòmica	Factura	Pla de viabilitat	Riscs econòmics
	8	8	0
Social	Impacte personal	Impacte social	Riscs socials
	9	8	-5
Ambiental	Consum del disseny	Petjada ecològica	Riscs ambientals
	9	9	0
Rang sostenibilitat	26	25	-5
	46		

Taula 7: Matriu de sostenibilitat

12. Desenvolupament del projecte

12.1. Introducció

En aquest apartat, s'explicaran tots els temes tractats per a la realització del projecte de forma organitzada i detallada, començant primer per la creació de l'entorn 3D i tot el que comporta, seguidament la incorporació dels personatges a l'entorn i com es comporten i naveguen a través de l'entorn, a continuació la navegació de l'usuari en l'entorn de realitat virtual juntament amb les dades que enregistrarem d'aquest i finalment la visualització de les dades enregistrades de l'usuari. En resum, aquest projecte es divideix en 3 fases principals que són:

1. Configuració del Unity per a una correcta visualització, on es té en compte la configuració de la importació i modificació de models, de les animacions, de la il·luminació i dels personatges en l'escena.
2. Disseny, implementació i solució d'errors de la màquina d'estats del comportament i la navegació i local avoidance dels personatges en l'entorn.
3. Navegació de l'usuari a través de l'entorn, és a dir, la interacció i el moviment previs a l'extensió de la realitat virtual, l'emmagatzematge i la visualització de les dades i l'extensió de la navegació a la realitat virtual.

Degut a què cal tenir coneixements en Unity sobre moltes seccions, gran part del temps s'ha dedicat a la familiarització d'aquestes com s'ha pogut comprovar en els canvis de la planificació de temps.

12.2. Creació de l'Entorn 3D

En la creació de l'entorn s'explicarà des de la introducció dels models a l'entorn, la modificació de les seves propietats, tot el sistema d'il·luminació utilitzat i la seva configuració per a assolir el nostre objectiu, els elements auxiliars que s'han utilitzat en l'escena per a algunes funcionalitats, els problemes que han sorgit i com s'han abordat. També s'explicarà les animacions utilitzades en els elements de l'entorn, en aquest cas les portes i els trens.

12.2.1 Disseny de l'entorn

Com s'ha comentat en la introducció del context del treball, aquest projecte parteix d'una base. Aquesta incorporava un model de l'estació, un model pels trens i diversos models de botigues i màquines de bitllets i personatges amb l'animació per defecte de Unity.

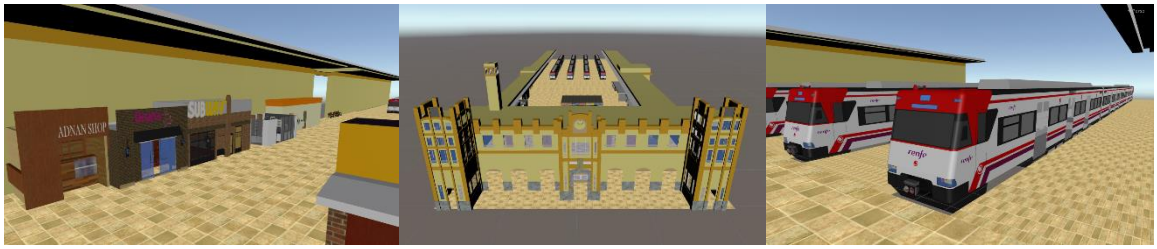
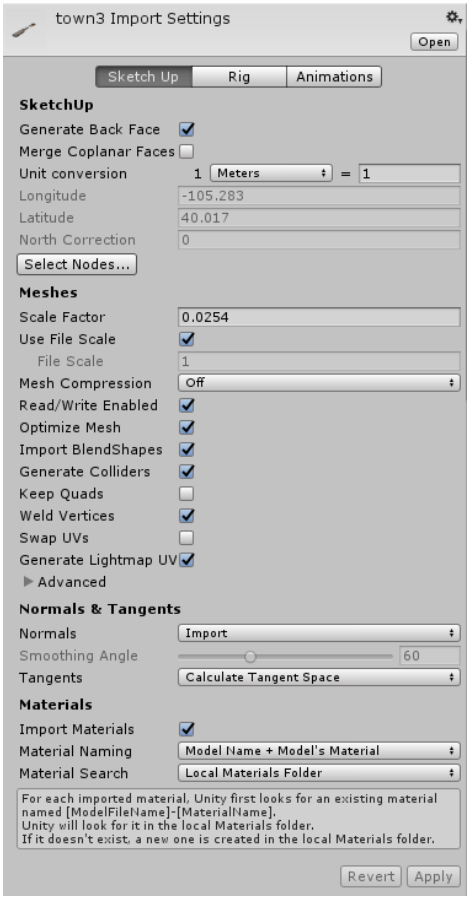


Figura 4: Elements inicials de l'escena

Per tal de millorar el realisme en l'escena, el primer que s'ha fet ha sigut buscar més models que omplin una mica l'entorn i l'usuari no trobi a faltar elements, tant exteriors com interiors, per això s'ha remodelat l'estació.

Degut a la falta de temps i de coneixements i experiència en modelatge, els models s'han obtingut d'una pàgina externa anomenada 3dwarehouse [14], que proporciona models gratuïts. Com que els models no són propis, s'ha requerit la modificació d'algun d'ells. Per la seva modificació, s'ha utilitzat el programa Sketchup gràcies a la seva facilitat d'ús.

Un cop aconseguit els models necessaris, s'han incorporat en l'escena. A l'hora d'incorporar models en l'escena, es poden seleccionar certes opcions, entre elles les que hem de tenir en compte segons les nostres necessitats són les següents:



- Generate colliders: Ens permet generar automàticament els colliders a partir del mesh del model. Els colliders son components que permeten col·lisió amb objectes físics (Un objecte es considera físic si té un component rigidbody i un collider).
- Generate Lightmap UVs: Genera de forma automàtica els Lightmap que Unity utilitza per als càlculs d'il·luminació. Això s'explicarà en el següent apartat 12.2.2.
- Generate Back Face: En cas que el model tingui plans (en Unity el plans només són visibles per la cara frontal) genera també la cara del darrera i la fa visible.

Figura 5: Configuració importació models

- Weld Vertices: Permet a Unity ajuntar els vèrtexs que es troben molt a prop entre ells en un de sol per millorar l'eficiència en les crides al renderitzar.

Després d'haver importat els models en l'escena, s'han modificat les seves textures per augmentar el seu realisme. Aquí s'introdueix una nova component que són els materials. Els materials són necessaris per a què els objectes es vegin visualment en l'escena. Aquests són els encarregats de decidir quin shader s'utilitza, els colors i les textures que es faran servir a l'hora de renderitzar l'objecte, i formen part de la component Mesh Renderer. Aquesta component permet dir-li a Unity que el Gameobject es renderitzi amb el material assignat.

Quan importem els models, s'importen automàticament els materials que venien amb ells. El shader que s'utilitzarà serà el Standard, i ens limitarem a modificar els paràmetres d'aquest per tal de millorar la visualització. Els paràmetres que s'han tingut en compte a l'hora de cambiar les textures són els següents:

- Rendering Mode: Ens permet triar si l'objecte tindrà transparència o no i el tipus de transparència que volem. En el nostre cas, la majoria d'objectes són opacs, a excepció d'alguns elements com les "barreres invisibles" que impedeixen al jugador sortir de l'àrea delimitada, o els vidres de les botigues i les portes. En aquest cas, s'ha decidit utilitzar l'opció de Transparent, ja que és la única que ens permet tenir l'objecte semi-transparent i conserva les reflexions.
- Albedo: En aquesta opció s'indica el color RGB i la transparència que volem que tingui l'objecte. Aquí s'ha de triar la textura corresponent a cada objecte.
- Metallic: Aquest paràmetre permet ajustar l'aparença de l'objecte des de un color mate fins a un color completament metàl·lic. En aquest, també es pot configurar la propietat de smoothness, que afecta també a la visualització de les reflexions de l'objecte.
- Normal Map: En cas que la textura obtinguda tingui un normal map, el podem associar al material per tal d'augmentar la qualitat d'aquest. Degut a què la creació de textures amb normal map és més sofisticada, les úniques textures que s'han trobat amb un normal map han sigut les de la vorera del carrer.
- Emission: Aquesta propietat permet ajustar el color i la intensitat de llum que emet el material.
- Tiling i Offset: Permet ajustar les coordenades d'inici i final i la posició de la textura renderitzada. Amb aquests paràmetres, podem ajustar les vegades que es repeteix la textura en un material, i d'aquesta manera millorar la seva qualitat en la visualització.

Per augmentar el rendiment, també es poden desactivar les opcions de Specular highlights i Reflections, d'aquesta manera el material no tindria reflexos ni el punt lluminós quan una llum l'està enfocant directament. Això és recomanable desactivar-ho en cas de veure disminuït el rendiment en l'escena.

Tots els paràmetres explicats anteriorment s'han estudiat prèviament i s'han configurat en els materials utilitzats per tal de millorar el realisme en l'escena.

De cara a la interacció de l'usuari amb l'entorn, s'ha utilitzat el sistema de tagging que ofereix Unity per poder identificar els objectes en runtime. Aquesta propietat la tenen tots els gameobjects de l'escena, i funciona com un identificador per poder classificar-los. A més, permet que des de codi sigui més fàcil accedir a un objecte en concret.

Per a les parets invisibles que limiten la zona de l'usuari, per tal que no afectin a altres elements s'ha modificat la matriu de col·lisions.

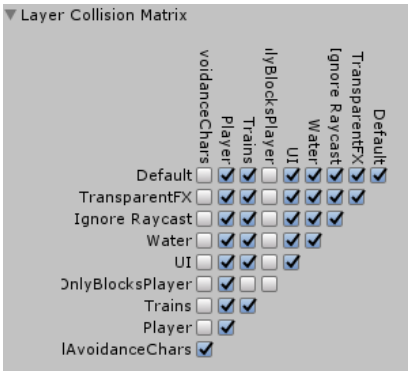


Figura 6: Matriu de col·lisions

A cada gameobject també se li pot assignar una layer o capa. Aquesta layer té diverses funcionalitats, i una d'elles es fa mitjançant la matriu de col·lisions de layers, que permet triar quins objectes afecten físicament a altres segons la layer a la que estiguin associats.

Per tal de disminuir el nombre d'estructures fora de l'estació, s'ha utilitzat una textura que envolta tota l'escena donant la sensació de què l'estació es troba en una ciutat. Aquesta textura s'anomena cubemap, i s'ha creat a partir de 6 textures quadrades que serien cada una de les cares d'un cub. La textura s'ha obtingut a través de la xarxa i s'ha modificat amb Gimp [15] amb l'objectiu de millorar el realisme.

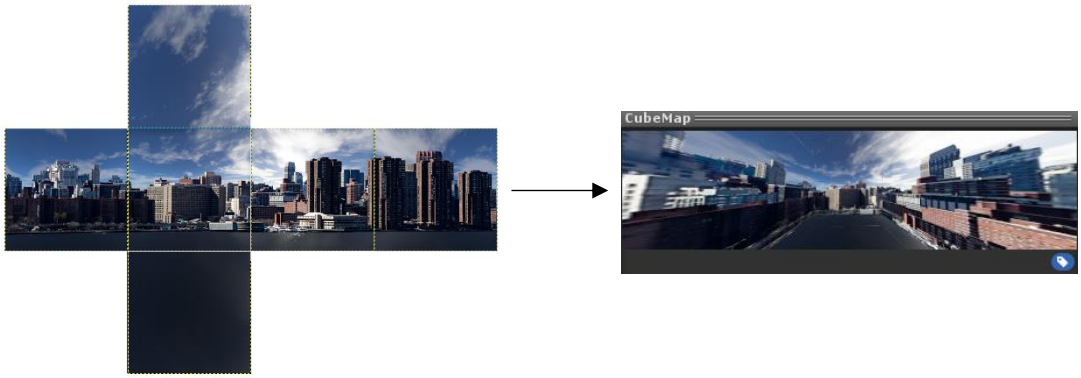


Figura 7: Cubemap de l'escena

Degut a què els models estaven molt mal optimitzats, el rendiment era molt baix, inferior als 30fps. Aproximadament, entre un 84-88% del temps que es trigava en processar un frame era degut a les crides de pintar l'escena. El renderitzat trigava entre 38-40 ms per frame.

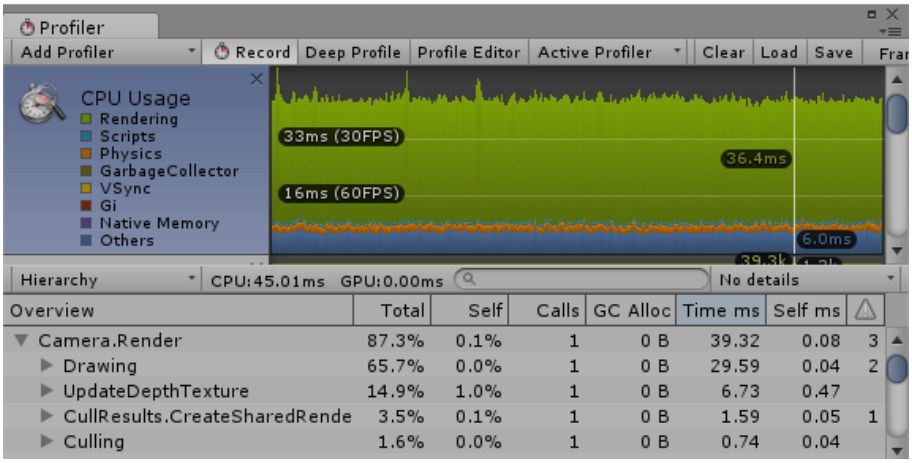


Figura 8: Dades profiler escena sense optimitzar amb una càmera

Aquestes dades han estat obtingudes gràcies a una eina que proporciona Unity per fer un tracking dels elements que es tenen en compte per al rendiment de l'aplicació.

Com es pot observar, aquest rendiment no és acceptable a l'hora de tenir en compte que s'ha d'aplicar la realitat virtual. A més, les dades obtingudes són sense els personatges autònoms en l'escena, i tot i així el rendiment és baix.

El principal problema estava en què els models importats es subdividien en centenars d'objectes. El que s'ha fet per solucionar això és, amb l'ajuda del Sketchup, s'ha agafat model per model i s'han agrupat els objectes en diferents mesh. Això el que fa és que Unity passi de fer centenars de crides a renderitzar a només unes desenes o menys. A l'hora d'agrupar els elements, s'ha tingut en compte d'agrupar-los segons les necessitats d'interacció de l'usuari, per exemple, les parets i el terra s'han posat a part dels elements de la botiga, i aquests s'han agrupat segons el tipus d'objecte que eren.

Un cop fet tot això, s'ha tornar a realitzar una prova de rendiment, i el resultat és el següent:

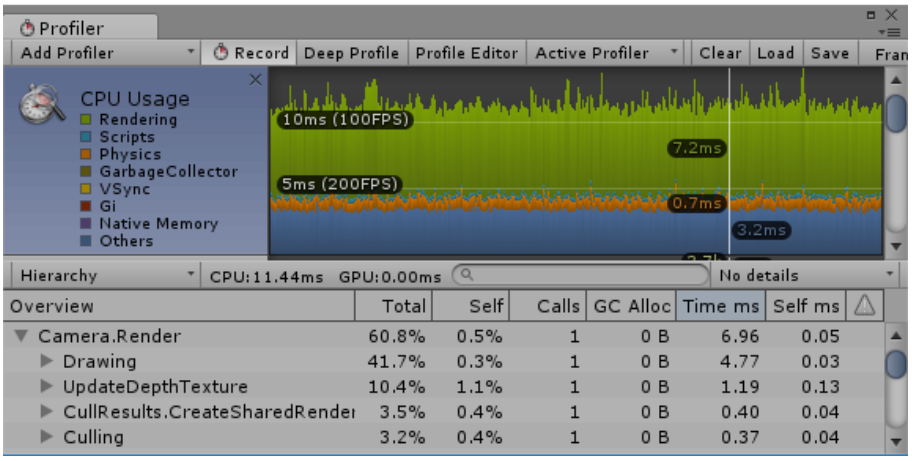


Figura 9: Dades profiler escena optimitzada amb una càmera

Com es pot observar, el nombre de fps ha augmentat considerablement a estar al voltant d'uns 80-90. Les crides a renderitzar encara ocupen la major part del temps per frame, però aquest temps s'ha disminuït fins al voltant d'uns 7-8 ms, cosa que es una millora general considerable.

Una possible millora per al futur de cara al rendiment seria optimitzar els models al màxim utilitzant eines com el Sketchup o altres modeladors 3D. Degut a la falta de temps i de coneixements d'aquests programes, aquesta millora és la màxima que s'ha pogut aconseguir.

12.2.2 Il·luminació

La il·luminació és un factor important a tenir en compte a l'hora de millorar la visualització de l'usuari dins de l'entorn i de la mateixa manera augmentar el realisme i la qualitat dels objectes. És per això que s'han estudiat a fons les funcionalitats que ens ofereix Unity per poder maximitzar el realisme en l'escena.

Unity té disponibles diferents tipus de llum. Aquí s'explicaran les llums que s'han utilitzat en l'escena. La primera llum que s'ha utilitzat ha sigut la Directional Light com a llum global, és a dir, la llum solar. És la més adequada ja que no importa la seva posició en l'escena, només la seva direcció, i la seva intensitat és la mateixa en tots els punts. Per les llums interiors s'han agafat els Point Lights. Aquestes llums si que tenen una posició i emeten llum en totes direccions en un radi forma d'esfera.

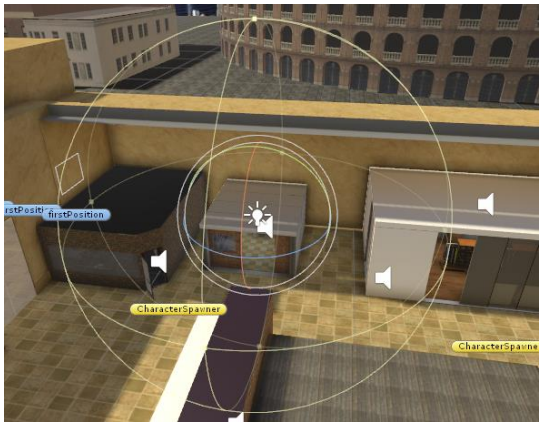


Figura 10: Point Light

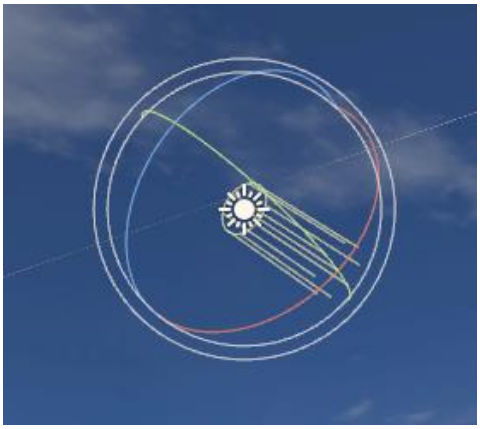


Figura 11: Directional Light

En cada tipus de llum es pot configurar la seva intensitat, el mode, el color i les ombres que produeixen. Aquest paràmetres s'han ajustat adequadament per tal de millorar la qualitat en general de l'escena. En el cas de les llums interiors, s'han eliminat les ombres ja que s'ha triat un mode de llum precalculada, o també anomenada baked, que explicarem a continuació.

Un dels problemes que s'han trobat un cop col·locades les llums ha sigut el rendiment en executar l'escena. Degut a la quantitat de llums que s'executen a l'hora i la quantitat d'elements, el temps de renderització de l'escena es molt elevat, ja que per cada llum, es calcula la il·luminació en cada element de l'escena per frame. Per tal de disminuir-lo, el que s'ha fet ha sigut posar les llums interiors de realtime a baked i efectuar el procés de càlcul de la il·luminació de tota l'escena que proporciona Unity, ja que d'aquesta manera la il·luminació està precalculada i emmagatzemada en els lightmaps dels objectes i s'aconsegueix una millora considerable de rendiment.

Els lightmaps són estructures que guarden la informació de la llum i per això és necessari generar-los quan s'importa un model en l'escena (lightmap UV). A més, per tal que un objecte es vegi afectat per la il·luminació precalculada, cal marcar-lo com a static en l'escena, d'aquesta manera Unity el té en compte pel càlcul. Cal tenir en compte que tant si un punt de llum està marcat com a realtime o baked, si no es fa un pre-càlcul de la il·luminació (bake) en l'escena només es té en compte la llum directa, és a dir, els rebots de llum no es calculen, i en les zones on teòricament hi arriba llum gràcies al rebot (cosa que passa en la vida real) estarien fosques totalment. A més, si no s'efectua el càlcul, totes les llums es consideren en realtime. Per això, sempre s'ha d'efectuar el bake de l'escena per a calcular, com a mínim, el rebot de la llum en zones on no hi arriba directament.

Cal tenir en compte també que la directional light s'ha posat en mode "mixed". Aquest mode és una barreja entre el realtime i el baked. La diferència és que aquesta llum també es té en compte per al càlcul de la il·luminació en els objectes estàtics, és a dir, actua com a una llum en mode bake. A més, però, pels objectes dinàmics com els personatges de l'escena, també es té en compte aquesta llum com si estigués activada en realtime, és a dir, permet calcular les ombres dels objectes dinàmics.

A l'hora de fer el bake de la il·luminació, s'han ajustat certs paràmetres a la configuració, entre ells, els més importants són els següents:

- Indirect resolution i Lightmap resolution: Permet ajustar la resolució final dels càlculs realitzats per la llum indirecta i els lightmaps. Aquests s'han ajustat de manera que no es dispari la memòria que s'utilitza per generar la informació ni el temps de càlcul.
- Compress Lightmaps: Aquesta opció permet comprimir els lightmaps per a disminuir l'espai en memòria, però poden aparèixer errors en alguns models degut a la compressió. Per tal de disminuir els errors d'il·luminació, aquesta opció ha estat desactivada.

- Fog: Aquesta opció permet afegir boira a l'escena. S'ha activat i s'han ajustat els paràmetres de color i densitat per tal de crear un ambient de ciutat.

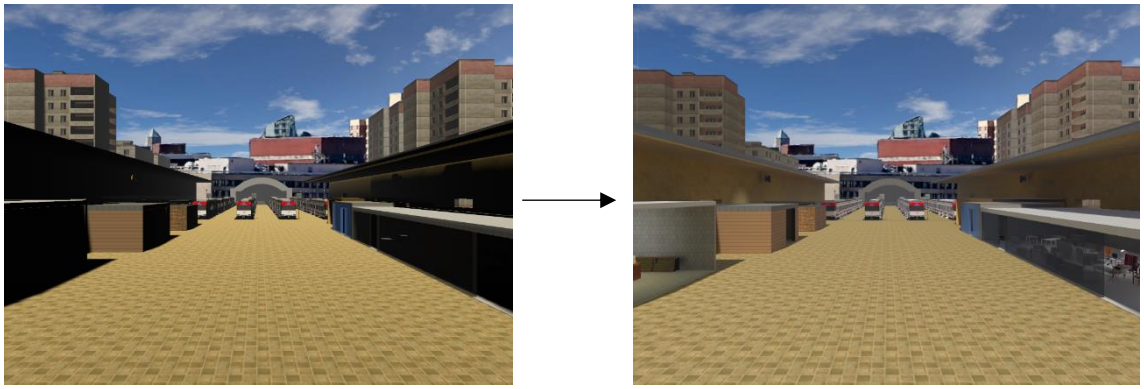


Figura 12: Escena abans i després del bake d'il·luminació respectivament

Després d'haver realitzat diverses proves amb la il·luminació, s'ha observat un problema amb el bake en les zones on hi havia objectes modulars. El problema sorgeix en què els càlculs de la il·luminació no es realitzen correctament en les arestes dels elements dels objectes.

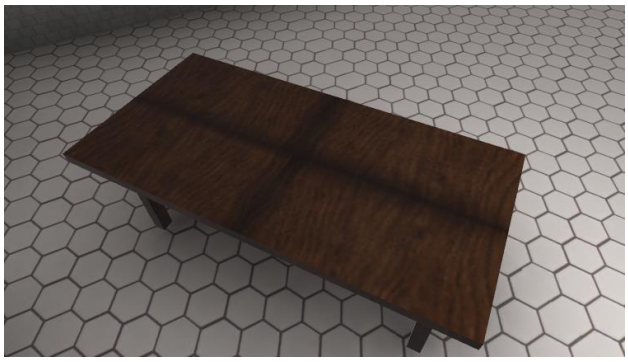


Figura 13: Zones fosques en element modular

La solució que s'ha aplicat ha estat, per a cada element de l'escena, modificar-lo a través del Sketchup i eliminar les arestes innecessàries per tal que l'objecte estigui unificat. D'aquesta manera s'ha aconseguit un resultat molt més acceptable.

Figura 14: Element modular unificat sense zones fosques

Un altre problema que ha sorgit al efectuar el bake de l'escena ha sigut la visualització dels elements dinàmics, en aquest cas els personatges, en entrar a les botigues. Degut a què les llums interiors només s'han tingut en compte a l'hora de fer el càlcul d'il·luminació, els personatges no es veien afectats per aquestes. La solució aplicada ha sigut la utilització d'un nou element anomenat Light Probe Group.

Aquest element permet col·locar esferes anomenades light probes en diferents punts de l'escena abans que s'efectuï el bake d'il·luminació. Aquestes light probes, un cop col·locades permeten guardar la informació de la llum calculada en el punt on s'han col·locat. Llavors, si un objecte es troba dins d'alguna de les àrees delimitades per les light probes més properes a ell, permet calcular la seva il·luminació aproximada a partir d'una interpolació dels valors calculats en les light probes més properes.

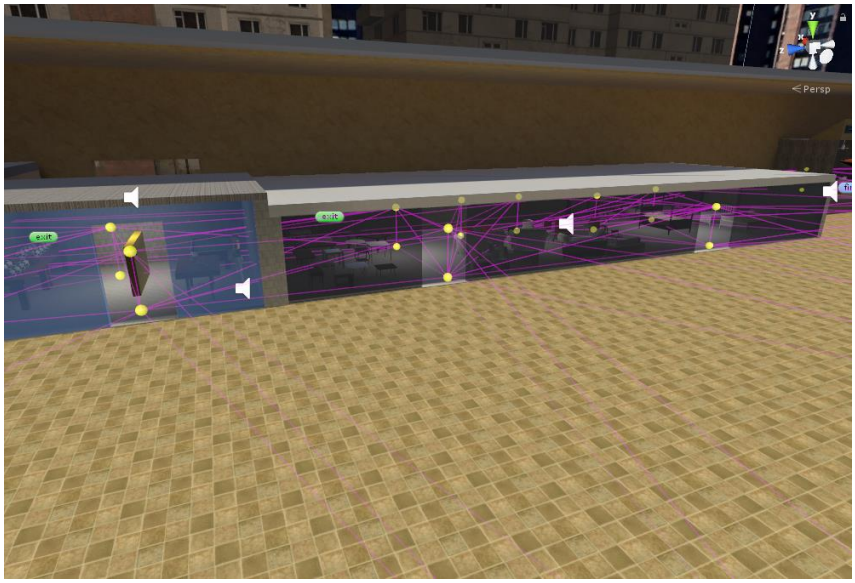


Figura 15: Light probes de l'escena

12.2.3 Animacions

En l'escena és important que, a més dels personatges autònoms que es troben en moviment, hi hagi certs elements en l'entorn que tinguin animacions per augmentar la sensació de presència. És per això que s'han afegit animacions a certs objectes que són els trens i les portes.

Per tal d'animar a un objecte, és necessari associar-li una component anomenada Animator. A aquesta component se li ha d'associar un controlador, i a dins de cada controlador es troben representades les transicions entre les animacions que conté aquest.

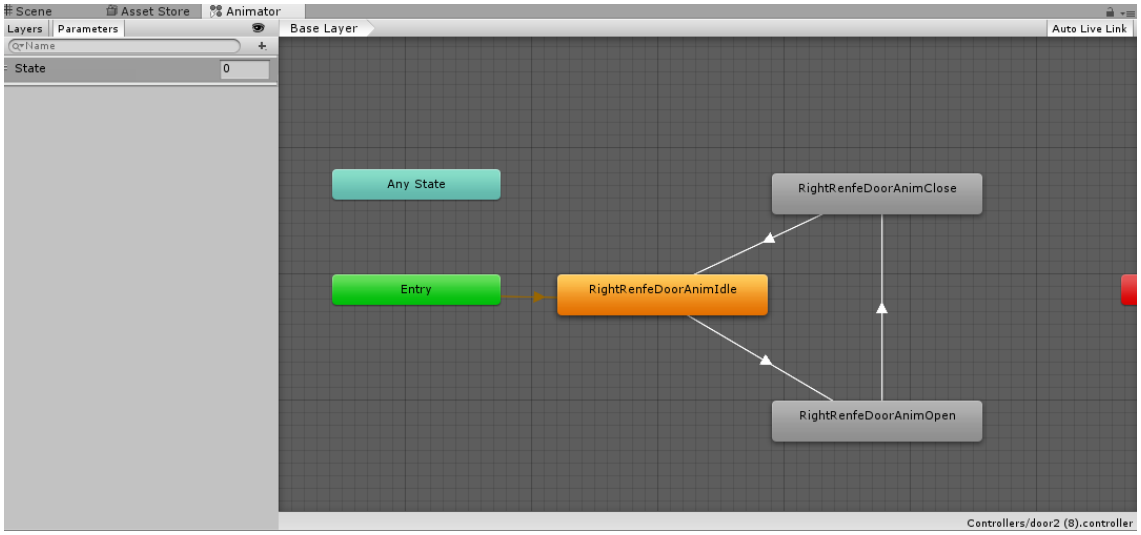


Figura 16: Controlador d'una porta dels trens

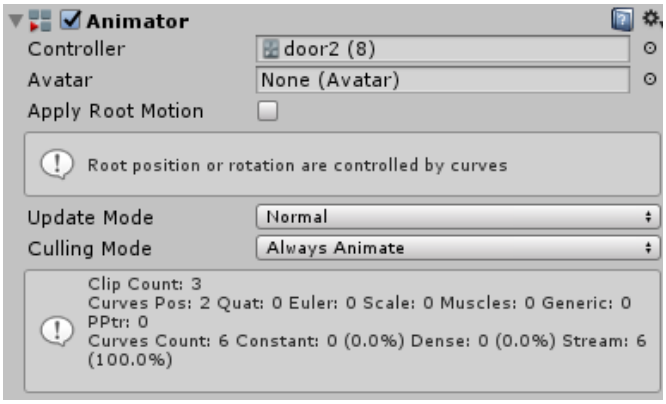


Figura 17: Component Animator

Les animacions comencen amb un estat inicial i, a partir de les transicions, van cambiant d'animació. A cada animació se li pot ajustar la velocitat a la que es reprodueix i si es vol que es repeteixi contínuament (loop). Per les transicions, es pot ajustar les condicions per les quals l'animació pot canviar cap a una altra. Per poder indicar una condició, primer s'han de crear els paràmetres necessaris a la secció de paràmetres. En el cas de les portes, només és necessari un paràmetre, que indicarem com a estat i serà un enter. Els paràmetres poden ser integers, floats, bools i triggers.

Un cop creats els paràmetres necessaris, hem de crear les condicions necessàries. Per les portes hem creat 3 animacions: Idle, on les portes no es mouen, Close, on les portes fan l'animació de tancar, i Open, on les portes fan l'animació d'obrir.

Cal indicar un paràmetre de les transicions important, i aquest és el "Has exit time". Aquest paràmetre permet que l'animació inicial no entri a la transició fins què aquesta acabi. En el cas de les portes en general s'ha activat per evitar que es puguin obrir i tancar sense parar.

Els paràmetres s'han de controlar a través dels scripts, i és per això que el funcionament del control de totes les animacions s'explicarà en l'apartat 12.3.2.

Fins aquí, el procés de creació de les components Animator i les transicions és molt semblant a la que s'ha utilitzat per als personatges autònoms, les diferències s'explicaran degudament en l'apartat 12.3.2. La creació de les animacions en els objectes és el que es diferència en les animacions dels personatges. En aquest cas, les animacions de totes les portes i dels trens s'han creat a partir de Mecanim.

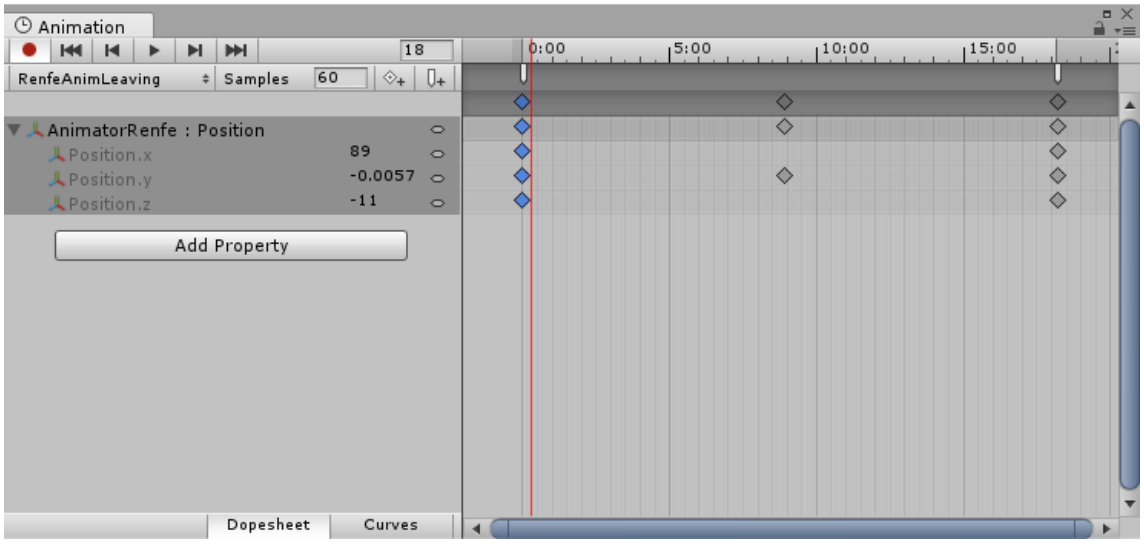


Figura 18: Finestra Animation per la creació d'animacions

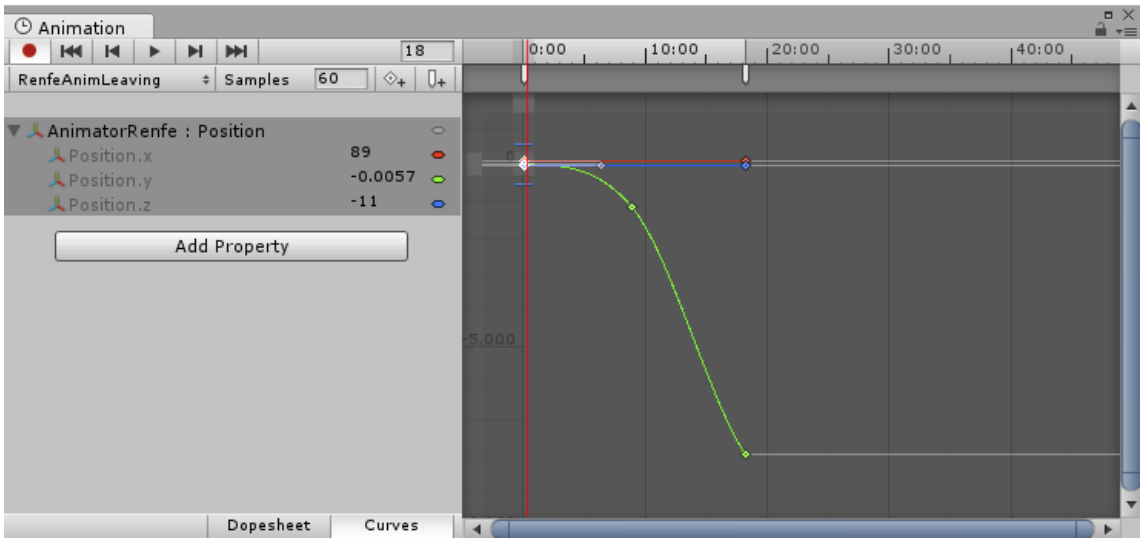


Figura 19: Finestra Animation configuració de les corbes de velocitat de transició

En la finestra Animation, es pot configurar la posició i rotació dels objectes en funció del temps que es troba de l'animació. Mecanim s'encarrega de, un cop indicats al menys un estat inicial i un final, fer tota la transició d'estats intermedis. A més, també es pot configurar la velocitat en la que canvia d'un punt a un altre. Per exemple, si volem crear un efecte d'acceleració i es vol anar des d'un punt fins a un altre en un segon, es pot ajustar la corba per tal que en mig segon només hagi avançat una quarta part del recorregut total i la resta avanci fins a la posició final (sempre amb una transició suau, perquè no es vegi que al cap de mig segon la seva velocitat canvia sobtadament). Aquestes corbes s'han ajustat tant a les portes com als trens per tal de crear un efecte més realista.

Cal indicar també que en les animacions, tant en les dels objectes de l'entorn com en les dels personatges, es poden crear "events" en cert moment de l'animació. Aquest events actuen com a triggers, i serveixen per activar funcions que es trobin en algun dels scripts associats a l'objecte que té l'animació. Aquesta funcionalitat també s'ha utilitzat i s'explicarà perquè s'ha utilitzat en l'apartat 12.3.2.

12.2.4 Sons

Els sons són una part fonamental a l'hora d'intentar assolir la màxima sensació de presència. És per això que també s'hi ha afegit a l'escena diferents sons com són el so

ambiental de l'estació, sons a algunes botigues al entrar, sons a les passes de la gent i sons als objectes, com els trens i els lavabos.

Per afegir sons, cal afegir un component a un gameobject anomenat Audio Source. Aquesta component té diversos paràmetres, i els que s'han modificat per a les nostres necessitats han sigut els següents:

- AudioClip: És el clip d'àudio associat al component que es reproduirà.
- Play On Awake: Si aquesta opció es troba activada, es reproduirà el so només començar l'escena.
- Loop: Si s'activa l'opció el so entrarà en loop.
- Spatial Blend: Serveix per reproduir el so en 2D o 3D. Si el so es reprodueix en 2D, el so que escolti l'usuari no es veurà afectat per la distancia entre aquest i el gameobject que té la component que reprodueix el so. En canvi, si el so es reprodueix en 3D, la distància sí que influeix, i la seva intensitat es pot configurar a través de la corba distancia/volum.

Tots els sons s'han obtingut a través de la xarxa, i per els sons dels altaveus, s'ha utilitzat una pàgina anomenada AcapelaBox [16], que permet reproduir amb veu el text introduït.

Els sons que s'han posat fixes en reproducció automàtica són els sons ambient i els de les botigues. Per a la resta de sons, s'activaran segons es compleixin determinades condicions.

En primer lloc tenim els sons dels trens. Per als sons dels trens intervenen dos sons per tren. Un és el so de la megafonia per anunciar que el tren s'està preparant per marxar, i l'altre és el so del mateix tren quan entra i surt de l'estació. El so del mateix tren el pot reproduir el script associat al tren en funció de l'estat en el que es trobi, per exemple, si

el tren esta marxant aquest activarà el so i el reproduirà un cop, i si està arribant efectuarà el mateix. El mateix passa amb els sons que estan associats i es volen reproduir en el mateix objecte, que serien els lavabos, tot i que la diferència d'aquests últims és que el seu so s'activa quan l'usuari interacciona amb ells, explicat en les seccions posteriors.

En canvi, per al so de la megafonia, internament el script associat al tren s'hauria de comunicar amb la component dels altaveus i informar-los d'activar el so en cert moment. Per poder assolir la comunicació entre els diferents scripts que estan associats als gameobjects de l'escena, no només amb els trens, es necessari d'un gameobject que controli en tot moment tot el que passa en l'escena. Per això, s'ha creat un objecte amb un script associat anomenat GameManager. Aquest, per tal de ser accessible des de qualsevol altre script, es crea una única instancia estàtica de tota la classe. Això s'anomena un singleton, i d'aquesta manera s'aconsegueix que les variables es puguin associar directament des de l'inspector de Unity, cosa que no es podria si les variables s'haguessin de declarar estàtiques.

Un cop introduït el GameManager, ja es pot efectuar la comunicació entre el tren i els altaveus. Els trens disposen d'un timer intern que permet saber en quin moment del seu estat es troben, i d'aquesta manera poden avisar al GameManager per tal que es reprodueixi el so de sortida del tren. En el cas dels altaveus, es disposa d'una cua interna on, si hi ha dos o més trens que marxen, la reproducció del so de sortida s'emmagatzema a la cua i es van reproduint un a un.

Per al so dels altaveus, s'han aplicat uns efectes per tal de fer-ho més realista. Per això, s'ha utilitzat el Audio Mixer de Unity, que permet crear un objecte Mixer i a aquest associar-li diferents efectes, en aquest cas, perquè s'assembli al so d'una megafonia. Un cop s'ha creat el mixer, aquest s'associa al paràmetre Output del Audio Source del gameobject que es vol que se li apliqui l'efecte.

Per últim, tenim el so de les passes de la gent i de l'usuari. Aquest també s'activen a través de l'objecte, però funcionen d'una manera diferent. En un principi es va intentar reproduir el so de les passes cada cert temps quan l'usuari o el personatge es trobava en l'animació de caminar, però això donava errors, ja que en les animacions s'utilitza un blend tree, i aquest fa interpolacions entre les animacions (explicat en l'apartat 12.3.2), cosa que no es troba en una animació determinada, sinó que retorna un valor entre 0 i 1. Això provocava que el so de les passes no estiguessin alineats amb els peus i el so es distorsionava.

Per tal de solucionar això, s'han utilitzat les corbes de l'animació.

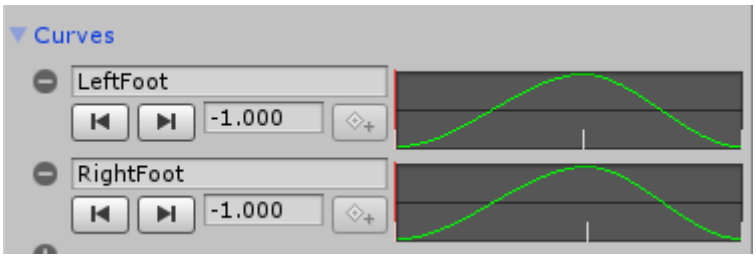


Figura 20: Corbes d'animació

Aquestes corbes permeten canviar el valor d'un paràmetre de l'Animator en funció del moment en què es trobi l'animació. Perquè les passes es facin quan es camina o es corre, les corbes només s'han creat per aquestes dues animacions (caminar i córrer), d'aquesta manera eliminem el problema de distorsió anterior.

La barra indica el moment de l'animació en el que es troba, i la corba indica el valor que agafaran les variables indicades, en aquest cas, LeftFoot i RightFoot. Llavors, el que es fa és detectar a través del script associat el valor de les variables en el frame actual i l'anterior. En el cas del peu esquerra, si el valor de LeftFoot actual és major o igual a 0 i l'anterior és inferior a 0, significa que hi ha hagut un canvi en la corba i s'executa el so de la passa. El mateix procés es fa per l'altre peu. La corba, per tal que el so de la passa soni quan toca amb el peu a terra, s'ajusta a l'animació en el canvi indicat de negatiu a positiu amb el frame en què el peu esquerra està tocant a terra, i el mateix per a l'altre peu.

Un cop ja hem acabat d'optimitzar l'escena i afegir-li tots els elements explicats anteriorment, el resultat ha sigut el següent:

Figura 21: Vistes d'escena final

12.3. Incorporació dels personatges

En aquesta secció s'explicarà tot el que es relaciona amb els personatges autònoms de l'escena començant des de la seva creació i importació, seguit del sistema d'animacions que s'ha creat perquè es puguin moure en l'escena i efectuar diverses accions, a continuació una explicació detallada del diagrama d'estats del comportament dels agents, tot el sistema de l'evacuació de l'estació i finalment l'anàlisi del pathfinding i la creació de l'algorisme de local avoidance per als agents. Tot això s'explicarà juntament amb les diferents possibilitats que s'han trobat per assolir els problemes i quines decisions s'han acabat prenent.

12.3.1 Importació dels personatges

A l'hora de crear els personatges, l'única opció que estava a l'abast ha sigut obtenir-los d'una pàgina externa, ja que la creació d'aquests té com a requisits coneixements previs avançats en modelatge i rigging, i aquesta opció s'ha descartat al instant. Els personatges han estat creats amb l'ajuda de la pàgina Adobe Character Generator ja mencionada anteriorment. Aquesta pàgina permet la creació de models 3D humanoides i els permet descarregar en format FBX, un format que Unity accepta per als models.

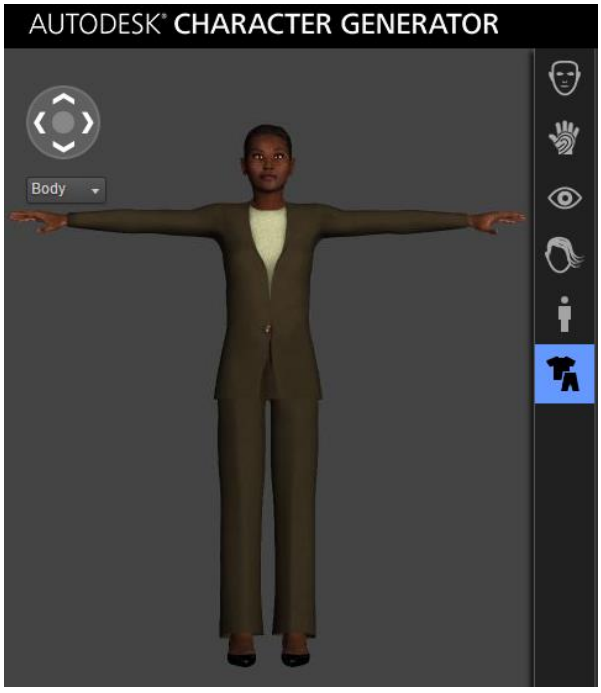


Figura 22: Model creat d'Adobe Character Generator

Abans d'importar el model, s'han de tenir en compte alguns dels paràmetres d'importació, com en els models d'objectes de l'entorn. Cal desactivar, llavors, la generació de colliders automàtica i els lightmap UV, ja que no es necessiten perquè aquests models seran dinàmics. La diferència principal amb els elements de l'entorn és que en els models de persones se li ha d'associar un rig. El rig és l'esquelet del model, està format per vèrtexs, que representarien les articulacions, i les unions, que representarien els ossos. El procés de rigging, ja explicat anteriorment, és un procés complicat, però Unity permet associar un rig per defecte (de tipus humanoide) en el cual es fa automàticament el procés de rigging a partir de la informació obtinguda del model. Per associar el rig, s'ha de canviar l'Animation Type a Humanoid, i la definició de l'avatar

(la informació necessària per el rigging) s'obté del mateix model. Fets aquests canvis, ja es poden importar a l'escena.

Un cop importats, per tal que puguin col·lisionar amb l'entorn, han de tenir un component Rigidbody i un Collider. En aquest cas, s'ha fet servir un Capsule Collider, ja que és el que més s'adapta a la forma del model. Un cop fet això, el model ja està preparat per a la importació a l'escena. A continuació s'explicarà el sistema d'animacions.

12.3.2 Animacions

El sistema d'animacions utilitzat per a la creació de transicions entre elles és el mateix que s'ha utilitzat per a les animacions dels elements de l'estació, el Animator. La diferència està en què les animacions per als models son molt més complexes, i la creació d'aquestes requereixen uns coneixements que no es tenen. Per això, s'ha mirat d'obtenir les animacions a través de la xarxa.

La primera possibilitat era l'obtenció d'animacions a través de la llibreria de la Universitat de Carnegie-Mellon. Aquestes animacions s'han trobat en el format adecuat .fbx, però el principal problema que hi havia a l'hora d'importar-les era que el rig de les animacions no coincidia amb el rig dels models importats a Unity. Això provocava un conflicte i Unity no podia reproduir les animacions en els models importats.

Degut a no poder importar les animacions de Carnegie-Mellon, s'ha buscat una altra possibilitat, obtenir les animacions a través de la pàgina de Mixamo.

En aquesta pàgina no hi ha problema en importar les animacions cap a Unity, ja que permet descarregar-les en un format fbx amb el rig que utilitza Unity. Tot i això, les

animacions que hi ha disponibles són molt limitades, i degut a això no s'ha pogut fer un acabat perfecte en quant a les accions que fan els personatges.

Perquè les animacions s'adaptin correctament als nostres models, s'ha d'importar un dels models creats en el Character Generator, ja que d'aquesta manera no hi haurà problemes després a l'hora d'importar les animacions a Unity un cop s'hagin associat al nostre model amb el rig correcte.

Les animacions que ens proporciona Mixamo tenen uns paràmetres ajustables, com per exemple, la separació que hi ha entre els braços i el cos, la velocitat a la que es produeix l'animació o, si es tracta d'accions, l'exageració d'aquestes. A més, aquestes animacions han estat creades a partir de Motion Capture [17] i porten incorporades les velocitats de rotació i translació respecte als moviments que efectuen. És per això que no tindrem el problema anomenat Foot Sliding, on el personatge animat no es mou sincronitzat amb la distància que realment avança, i visualment no queda realista.

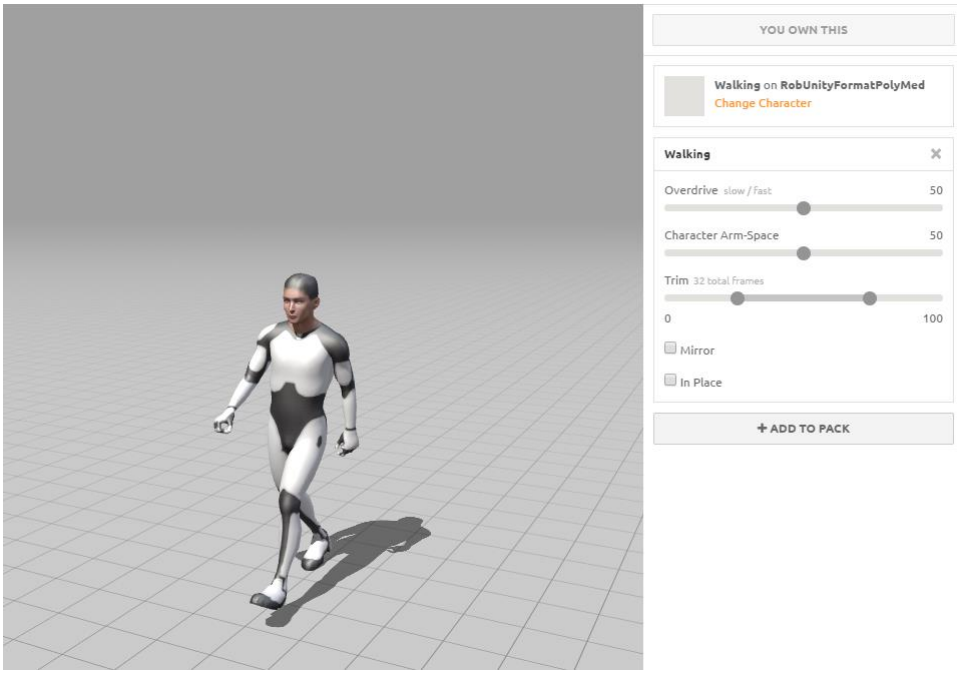


Figura 23: Exemple d'animació de mixamo

Un cop ajustats els paràmetres i importades les animacions a Unity, s'han d'ajustar certs paràmetres abans de crear el Controller de l'Animator. En primer lloc, cal indicar que el tipus d'animació és humanoide, i l'avatar que s'utilitzarà per el rig cal indicar un dels

avatar generats dels models sense importar quin, ja que en un principi els models que farem servir tenen característiques molt semblants, i d'aquesta manera no cal importar una mateixa animació per a cada un dels models. Un cop fet això, ens deixarà ajustar nous paràmetres a l'apartat Animations.

En aquí, es pot configurar si l'animació entrarà en loop o no i, en cas que tingui rotació i desplaçament incorporats, es pot decidir bloquejar-ho o no.

El loop ens interesa activar-lo, per exemple, en les animacions de caminar, girar o córrer, ja que serem nosaltres els que controlem a partir del codi en quin moment deixaran de fer-ho. En canvi, per les accions com pagar al sortir d'una botiga o al comprar un bitllet, ens interessa desactivar-la.

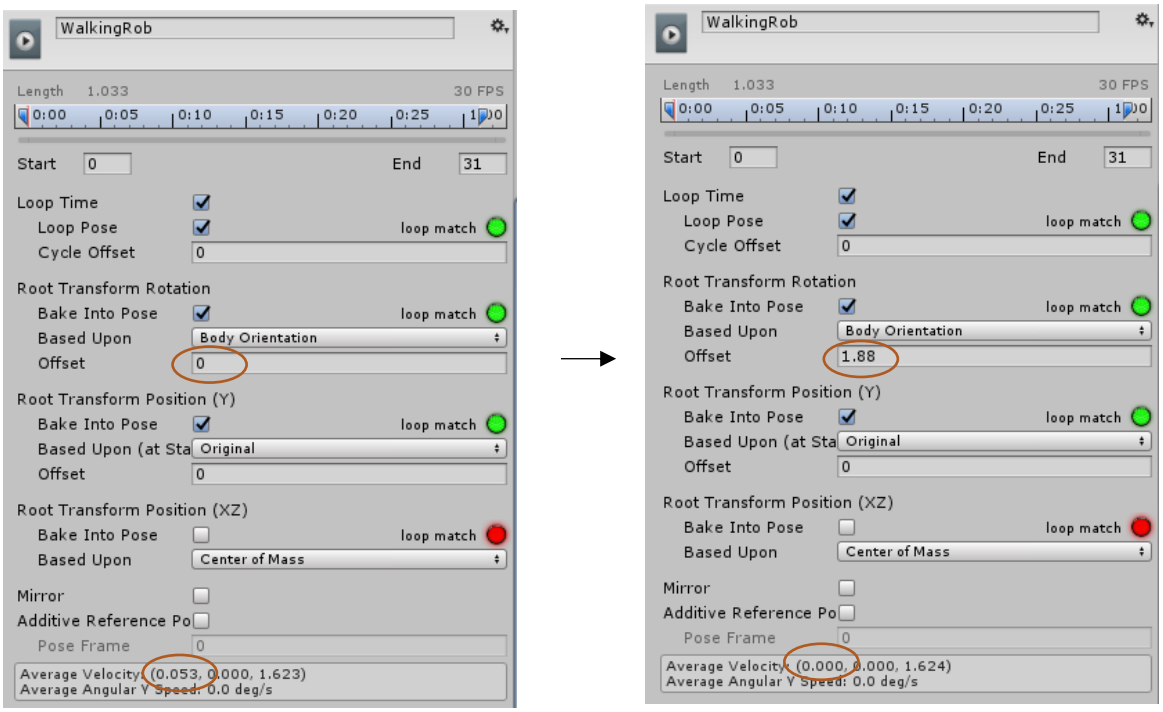


Figura 24: Configuració de paràmetres d'animacions

El bloqueig de les rotacions o les posicions serveix per eliminar els marges d'error de desplaçament o orientació de l'animació. Per exemple, en l'animació de caminar, si l'animació té un valor en la velocitat angular Y, quan activem l'opció del bloqueig de la rotació, aquesta velocitat s'elimina. Tot i això, quan s'activa aquesta opció altres valors

en la velocitat poden variar. És per això que s'ha d'acabar ajustant amb un offset fins aconseguir els resultats esperats.

Un cop finalitzats els ajusts per a totes les animacions, s'ha creat el següent controlador per als personatges autònoms:

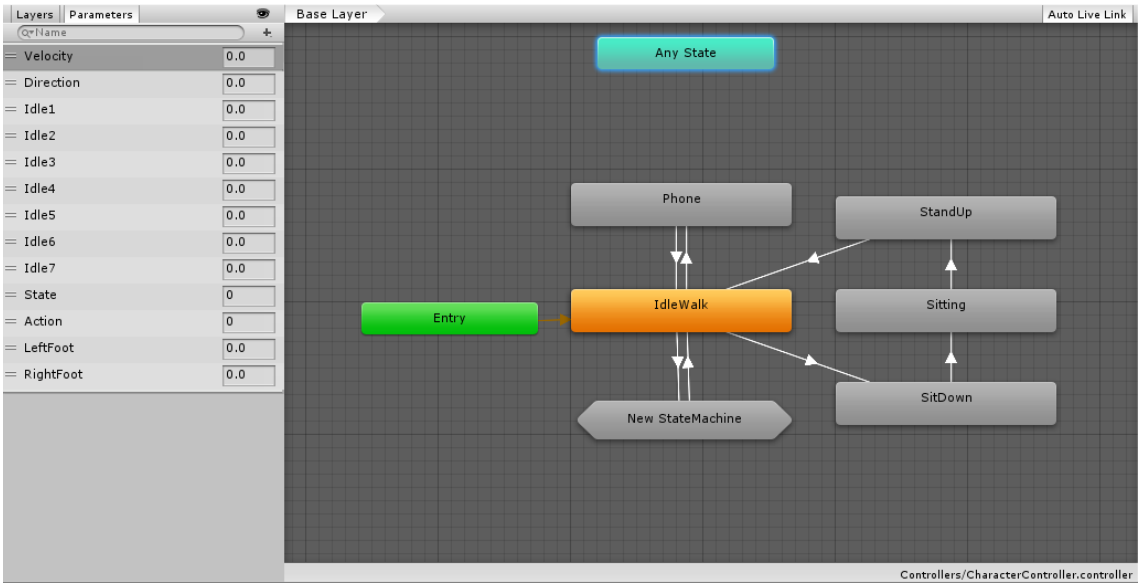


Figura 25: Controlador dels personatges autònoms

L'estat inicial és un blend-tree, on aquest té les animacions de moviment d'un personatge: caminar, córrer, estar-se quiet, girar cap als costats, caminar cap als costats, córrer cap als costats i anar endarrere. Un blend-tree permet barrejar dues o més animacions en funció de certs paràmetres, d'aquesta manera el que s'aconsegueix, per exemple, és una transició fluida entre les animacions de caminar i córrer, o estar-se quiet i començar a caminar i girar.

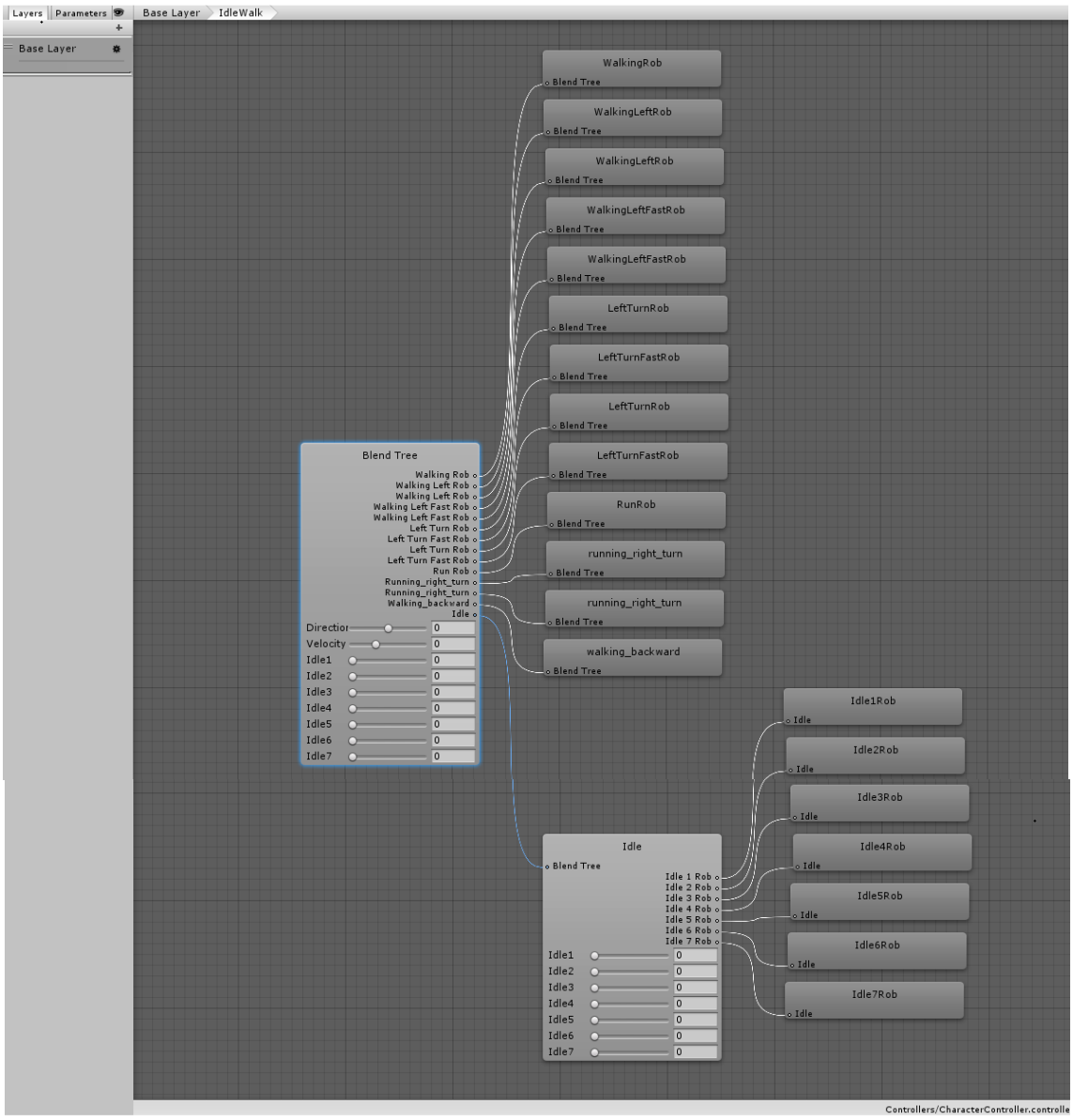
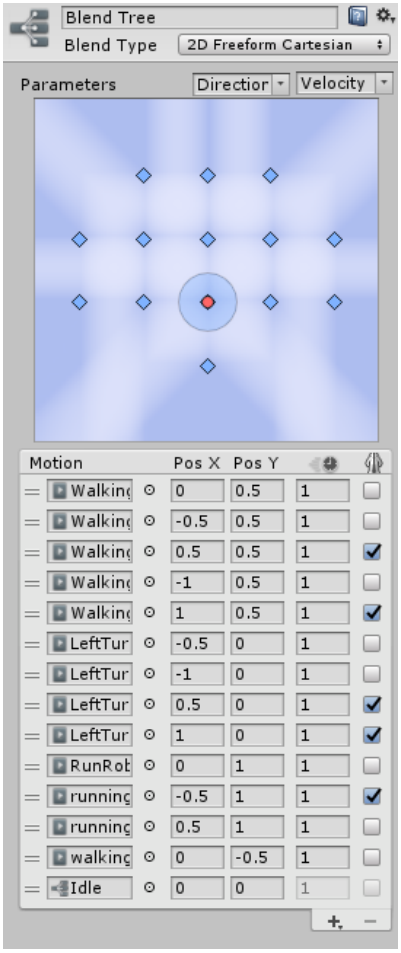


Figura 26: Estat inicial, blend tree amb animacions de moviment i Idle

En cada blend-tree es pot observar els paràmetres que intervenen per a la transició d'animacions. Per al primer, els paràmetres que influeixen són la velocitat i la direcció, i dins d'aquest s'ha creat un altre blend-tree perquè el personatge, quan es trobi quiet no efectuï sempre una mateixa animació de Idle i, d'aquesta manera, s'augmenta la percepció de realisme en els personatges.

Per a la configuració de quines animacions s'activen segons el valor dels paràmetres s'ha canviat el tipus de blend-tree.



En total hi ha 5 tipus de blend-tree. En primer lloc tenim el blend-tree en una dimensió, i aquest no s'ha triat ja que només ens permet efectuar les transicions entre les animacions de córrer, caminar, anar endarrere i estar quiet sense tenir en compte la segona dimensió en la que ens trobem, com córrer mentre s'està girant cap a un costat.

En segon lloc, tenim les opcions de 2D Simple Directional i 2D Freeform Directional. Aquestes opcions tampoc ens serveixen ja que estan orientades cap a animacions que representen direccions. En el nostre cas, les nostres animacions no representen direccions, ja que no utilitzem una animació de caminar cap als costats, per exemple.

Figura 27: Paràmetres blend-tree

En tercer lloc, tenim l'opció triada que és la 2D Freeform Cartesian. Aquesta està orientada cap a la transició d'animacions que no tenen direcció, i és la que millor s'adapta a les nostres necessitats, com hem explicat en el cas anterior.

Per últim, tenim l'opció Direct, que ens permet assignar-li variables directament a cada una de les animacions. Aquesta opció s'ha triat per a la realització de la transició d'estats Idle dels personatges, ja que d'aquesta manera permet creat un sistema de transicions d'estats aleatori.

El personatges sortirà de l'estat inicial en cas que necessiti efectuar alguna acció. Les accions s'han dividit segons els estats en què s'efectuen. D'una banda, tenim l'animació

de Phone, i aquesta s'efectua en cas de què la persona, en funció d'una probabilitat en el codi, entri en l'estat de Idle. Aquesta transició és semblant a les que ja hem vist en les animacions de les portes, per exemple, juntament amb les animacions de seure, estar sentat i aixecar-se (SitDown, Sitting i StandUp, respectivament). El personatge pot entrar en aquestes animacions en funció de la variable State, en aquest cas, efectuarà l'animació de Phone en cas que State sigui 1, o en l'animació de SitDown en cas que sigui 3, i sortirà d'elles si State torna a ser 0.

D'altra banda, tenim les accions que s'efectuen un sol cop, on s'han agrupat dins d'una màquina d'estats per separat. El personatge entrarà dins d'aquest sub-estat en cas de què State sigui 2. Un cop dins, es manté en Idle fins què el valor d'una segona variable Action li permet fer la transició cap a una de les animacions que representen les accions. Un cop finalitzada l'acció, per tal de no entrar un altre cop en la transició cap a la mateixa acció, el valor de Action es canviarà a 0 fent servir un Event al final de l'animació, que cridarà a una funció que canviarà el valor de la variable a 0. Aquest sortirà de la màquina d'estats quan el valor de State torni a ser 0.

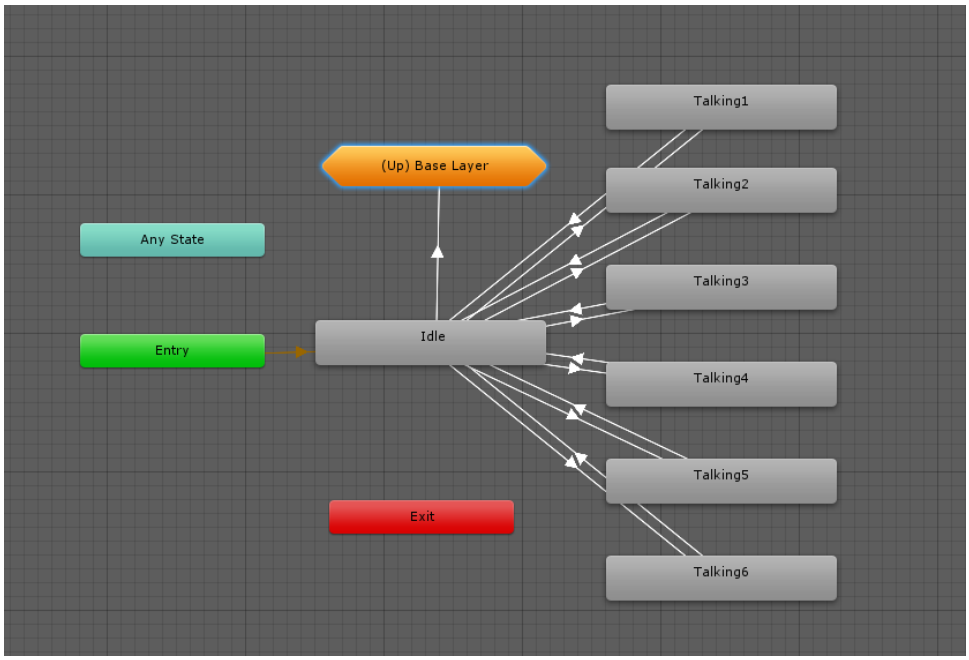


Figura 28: Màquina d'estats d'accions del personatge

12.3.3 Diagrama d'estats

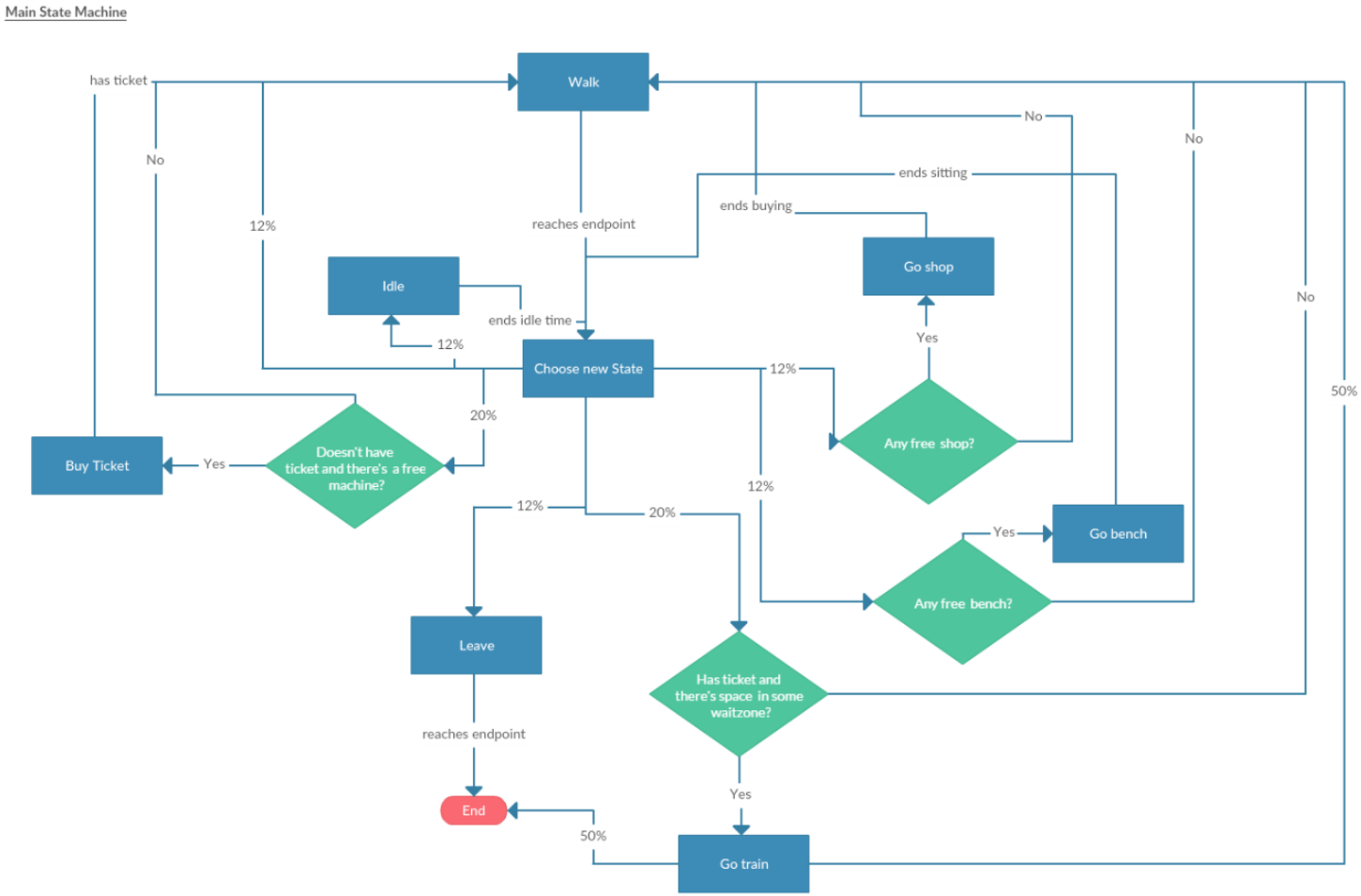


Figura 29: Diagrama d'estats del comportament dels personatges

Per la implementació del comportament dels personatges en l'entorn, s'ha dissenyat primer la màquina d'estats finita indicada anteriorment. Per a cada un dels estats, s'executen un seguit de fases que explicarem a continuació. A més, per l'estat d'anar cap al tren, també hi intervé la màquina d'estats dels trens, és per això que es dedicarà una secció a l'explicació del comportament d'aquests.

12.3.3.1. Generació dels personatges

En aquí es generen els personatges en diferents punts aleatoris de la malla de navegació i s'activa una funció que regula el nombre de personatges en l'escena, és a dir, si aquest disminueix, en genera de nous.

Precondicio: Cert.

Postcondicio: S'han instanciat els personatges en l'escena.

CharacterSpawner – Inicialització

```
m <- Màxim nombre de persones generades
n <- Nombre de persones actuals en l'escena
FOR i = 1 to m DO
      p <- Punt vàlid de la malla de navegació en l'escena;
      pers <- Nova instància de persona amb model aleatori en el
            punt p;
      n = n + 1;
END FOR
Generacio_automatica(); #Es tracta d'una corutina i s'efectua
                           repetidament cada cert temps en
                           l'execucio de l'escena.
```

Precondicio: Cert.

Postcondicio: S'ha instanciat un personatge que faltava en l'escena.

CharacterSpawner – Generació automàtica()

```
t <- Màxim nombre de persones en l'escena
n <- Nombre de persones actuals en l'escena
IF n < t THEN
      p <- Punt de sortida aleatori;
```

```
      pers <- Nova instància de persona amb model aleatori en el
            punt p;
      n = n + 1;
END IF
```

Abans d'explicar els estats dels personatges, primer explicarem com s'instancien en l'escena.

Per instanciar-los, el que s'ha fet ha sigut crear diversos gameobjects en l'escena que serviran de "spawners" per als personatges. Aquests tenen un script associat on s'indica el nombre de personatges que instanciarà un gameobject, el nombre de personatges totals en l'escena, una llista dels models que poden tenir els personatges i una llista amb els punts de sortida de l'estació.

La feina que realitzaran els generadors de personatges serà la de generar més personatges quan els que es troben en l'escena o bé marxen, o bé se'n van amb el tren, a excepció de quan comenci l'evacuació, llavors la generació s'aturarà.

Cal indicar que l'estat inicial de tots els personatges un cop s'hagin instanciat en l'escena serà el de Walk.

12.3.3.2. Estat Walk

En aquest estat, el personatge tria un punt aleatori vàlid dins de la malla de navegació, es calcula el camí a partir de l'algorisme de pathfinding de Unity i es dirigeix cap al punt.

Precondicio: El personatge no té cap estat nou assignat.
Postcondicio: El personatge té assignat l'estat walk.

$\underline{CharacterController - CalcularNouEstat()}$

```
k <- Random(1,7);
IF k == 1 THEN
      estat_actual <- walk;
```

```
      velocitat_agent <- Random(0.3f,0.5f);
      destinació_agent <- NovaDestinacio();
END IF
```

Precondicio: El personatge no té cap punt de destí assignat.

Postcondicio: El personatge té un punt dins de la malla de navegació assignat com a destí.

$\underline{CharacterController - NovaDestinacio()}$

```
te_desti <- false;
desti <- posició de l'agent;
WHILE !te_desti DO
      FOR i = 0 to 30 DO
            p <- Punt aleatori dins d'un rang esfèric des de la
                  posició de l'agent;
            IF p es troba en la malla de navegació THEN
                  te_desti = true;
            END IF
      END FOR
END WHILE
RETURN destí;
```

El mètode Update() s'executa un cop per frame.

Precondicio: Cert.

Postcondicio: S'ha actualitzat l'estat en el que es troba el personatge.

$\underline{CharacterController - Update()}$

```
SWITCH state
      CASE walk:
            IF distancia(posició_actual,posició_desti) < 0.2f THEN
                  CalcularNouEstat();
            END IF
      BREAK;
END SWITCH
```

En aquest estat, es selecciona una velocitat aleatòria de desplaçament per a l'agent. La velocitat indicada s'agafa com a entrada per al càlcul dels valors de les animacions Velocity i Direction, això s'explicarà en l'apartat 12.3.5. A continuació, es busca un punt aleatori que es trobi dins de la Navigation mesh, és a dir, un punt que tingui un camí vàlid des de la posició actual fins a la posició destí en un radi al voltant de l'agent.

Un cop ha trobat una posició vàlida, l'agent es dirigeix fins al destí i, al arribar, selecciona un altre estat aleatori.

12.3.3.3. Estat Idle

En aquest estat, el personatge es queda parat durant un temps fins que un temporitzador intern arriba a 0. Llavors, es calcula un nou estat.

Precondicio: El personatge no té cap estat nou assignat.

Postcondicio: El personatge té assignat l'estat idle.

$\overline{CharacterController - CalcularNouEstat()}$

```
k <- Random(1,7);
IF k == 2 THEN
      estat_actual <- idle;
      telefon <- (Random(0,1) <= probabilitat d'entrar a
                  l'animacio de telefon);
      IF telefon THEN
            estat_animacio <- 1;
            tempsEnIdle <- 49;      #en segons
      ELSE
            estat_animacio <- 0;
            tempsEnIdle <- Random(2.0,5.0);
      END IF
      velocitat_agent <- 0;
END IF
```

Precondicio: Cert.

Postcondicio: S'ha actualitzat l'estat en el que es troba el personatge.

CharacterController – Update()

```
SWITCH state
      CASE idle:
            tempsEnIdle <- tempsEnIdle – tempsFrame;
            IF tempsEnIdle <= 0 THEN
                  IF telefon THEN
                        estat_animacio <- 0;
                  END IF
                  CalcularNouEstat();
            END IF
      BREAK;
END SWITCH
```

Si l'agent entra en estat Idle, té una possibilitat d'entrar a l'animació de Phone. Si entra en aquest estat, s'executa l'animació sencera canviant el valor corresponent al State de l'Animator i al acabar s'escull un nou estat.

Si no entra en l'estat Phone, llavors el seu estat queda en Idle, i el valor que agafarà per les animacions d'Idle estan controlades per un altre script. En aquest script, es fa una interpolació dels valors cada cop que es canvia entre les animacions d'Idle en un temps determinat, en cas que l'agent estigui parat i en State a 0. Si l'agent es comença a moure, es canvia el valor d'Idle al per defecte, i es fa una comprovació dels anteriors valors perquè no es produeixin errors en les animacions.

12.3.3.4. Estat Buy Ticket

En aquest estat, el personatge es dirigeix cap a una de les màquines de bitllets i es posa a fer cua. Aquest va avançant fins que és el seu torn d'agafar el bitllet. Un cop té el bitllet, surt de la cua i entra en estat walk.

Precondicio: El personatge no té cap estat nou assignat.

Postcondicio: El personatge té assignat o bé l'estat buy ticket o bé l'estat walk.

CharacterController – CalcularNouEstat()

```
k <- Random(1,7);
IF k == 3 THEN
      IF !te_tiquet THEN
            IF hi ha una maquina de tiquets lliure THEN
                  estat_actual <- go_to_line;
                  destinacio_agent <- posició maquina tiquets;
            ELSE
                  estat_actual <- walk;
            END IF
      ELSE
            estat_actual <- walk;
      END IF
END IF
```

Precondicio: Cert.

Postcondicio: El personatge té tiquet i l'estat assignat a walk.

CharacterController – Update()

```
SWITCH state
      CASE go_to_line:
            IF distancia(posició_actual,posició_desti) < 1.5f THEN
                  destinació_agent <- posició seguent de la cua;
                  AfegirPersonaCua(); #Afegeix la persona a la cua
                                      interna de la màquina;
                  estat_actual <- queue;
            END IF
      BREAK;
      CASE queue:
            IF distancia(posició_actual,posició_desti) < 0.3f THEN
                  rotar personatge en direcció a la maquina;
                  estat_actual <- get_ticket;
```

```
            END IF
      BREAK;
      CASE get_ticket:
            IF primer de la cua THEN
                  efectuar acció d'afagar ticket;
                  IF ha acabat d'efectuar acció THEN
                        EliminarPersonaCua(); #Elimina la persona de la
                                              cua interna de la maquina
                                              i actualitza la posició de
                                              la resta de gent;
                        estat_actual <- walk;
                  END IF
            END IF
      BREAK;
END SWITCH
```

En la realització d'aquest estat, es va pensar en un principi que s'implementaria un sistema de cues, ja que vist des d'un punt realista, també es fan cues a l'hora de comprar bitllets. Per la qual cosa, aquesta va ser la solució única i definitiva que es va aplicar en aquest estat.

Per poder accedir en l'estat, l'agent fa una comprovació prèvia per veure si ja ha comprat el bitllet de tren anteriorment. Si no es així, i hi ha alguna màquina de bitllets lliure (cada màquina té una capacitat màxima de persones en espera) llavors el sistema s'encarrega d'agafar la posició i el ID de la màquina que estigui més lliure i l'afegeix a la cua de la màquina. Quan parlem de sistema, ens referim a la comunicació entre el GameManager i la resta de classes, ja que és ell qui s'encarrega d'obtenir i de donar la informació a aquestes.

Seguidament, l'agent es dirigeix cap a la màquina i un cop està a un rang determinat d'ella, es col·loca a la seva posició de la cua. L'agent va avançant a mesura que els agents

davant seu van comprant el bitllet, fins què arriba el seu torn. Una vegada li toca comprar el bitllet, efectua l'acció de comprar, al finalitzar s'elimina de la cua i entra automàticament en l'estat Walk.

12.3.3.5. Estat Leave

En aquest estat, el personatge tria una de les sortides disponibles de l'estació i es dirigeix cap a ella. Un cop ha arribat, s'elimina de l'escena.

```
Precondicio: El personatge no té cap estat nou assignat.
Postcondicio: El personatge té assignat l'estat leave.
```

$\overline{CharacterController - CalcularNouEstat()}$

```
k <- Random(1,7);
IF k == 4 THEN
      estat_actual <- leave;
      velocitat_agent <- Random(0.3,0.5);
      destinació_agent <- posició d'una de les sortides de l'estacio
                          triada aleatòriament;
END IF
```

```
Precondicio: Cert.
Postcondicio: S'ha eliminat de l'escena el personatge.
```

$\overline{CharacterController - Update()}$

```
SWITCH state
      CASE leave:
            IF distancia(posició_actual,posició_desti) < 1.0f THEN
                  Eliminar persona del nombre total de persones actuals
                  en l'escena;
                  Destroy(this.gameObject); #Elimina del sistema la
                                            instancia de l'objecte;
            END IF
      BREAK;
END SWITCH
```

Aquest és un dels dos estats on l'agent marxarà definitivament de l'estació. En aquest estat, l'agent busca la sortida més propera i es dirigeix cap a ella. Un cop arriba a la destinació, el seu gameObject és eliminat i desapareix de l'escena.

S'ha tingut en compte que fer desaparèixer un personatge influeix en la immersió de l'usuari, és per això que les sortides s'han col·locat en punts concrets per tal que l'usuari no pugui veure l'agent desaparèixer.

12.3.3.6. Màquina d'estats dels trens

L'estat dels trens va canviant en funció de l'estat actual. Inicialment es troben en l'estació, i s'esperen fins que un temporitzador intern s'activa. Un cop activat, esperen a que pugi la gent i se'n va de l'estació. Quan ja ha marxat, s'espera un altre cop a que acabi del temporitzador. Un cop finalitzat, tria un nombre de persones aleatori, els elimina de l'escena i torna cap a l'estació. Quan arriba a l'estació, es repeteix el cicle.

```
Precondicio: Cert.
Postcondicio: S'ha actualitzat l'estat en el que es troba el tren.
```

$\overline{Train - Update()}$

```
#estat <- Estat actual del tren
#timer <- temps d'espera del tren en l'estació i fora
#estat_animacio <- animacio activa del tren
#tren_en_cami <- indica si el tren es troba en camí o no
#tren_marxant <- indica si el tren esta marxant o no
p <- Nombre de persones que es troben dins el tren
perc <- Percentatge aproximat de persones que s'eliminaran de l'escena
        un cop el tren estigui fora
IF estat == coming AND animacio activa del tren no es obrir portes THEN
      ObrirPortes(); #Efectua l'animació d'obrir les portes
      estat <- opening;
END IF
IF estat == waiting AND timer <= 15 AND !speakersCalled THEN
      AfegirSoSpeakers(); #Afegeix a la cua de reproducció dels altaveus
```

```
                        de l'estacio el so de la línia en la que es
                        troba el tren
      speakersCalled <- true;
END IF
IF timer <= 0 THEN
      IF state == is_out
            estat_animacio <- 2;
            tren_en_cami <- true;
            FOR i=0 to p DO
                  s <- Random(1,100);
                  IF s <= perc THEN
                        p = p – 1;
                        Destroy(character);
                        i = i – 1;
                  END IF
            END FOR
            AfegirPersonesEscena(); #afegeix les persones que havien
                                    marxat al total de persones actuals
                                    en l'escena
            estat <- coming;
      ELSE IF state == waiting
            state <- ready_to_go;
      END IF
END IF
IF state == ready_to_go THEN
      IF no hi ha persones entrant THEN
            FOR i=0 to p DO
                  Associar l'agent 'i' al tren per tal que quan aquest
                  es mogui, la gent també ho faci amb ell.
            END FOR
            estat <- closing;
            TancarPortes(); #Efectua l'animacio de tancar les portes
      END IF
```

```
END IF

IF state == leaving AND !tren_marxant THEN
      EliminarPersonesEscena(); #elimina les persones que estan dins el
                                tren al total de persones actuals en
                                l'escena
      estat <- is_out;
      timer <- Random(mín_temps_espera,max_temps_espera);
      speakersCalled <- false;
END IF
IF state == opening THEN
      IF portes obertes THEN
            FOR i=0 to p DO
                  Desvincular l'agent 'i' del tren;
            END FOR
            p <- 0;
            estat <- waiting;
            timer <- Random(mín_temps_espera,max_temps_espera);
      END IF
END IF
IF state == closing THEN
      IF portes tancades THEN
            estat_animacio <- 1;
            tren_marxant <- true;
            estat <- leaving;
      END IF
END IF
```

Abans d'explicar l'estat Go Train dels personatges autònoms, s'explicarà a continuació el comportament que segueixen els trens. Els trens tenen una màquina d'estats independent de la màquina d'estats dels personatges que és la següent:

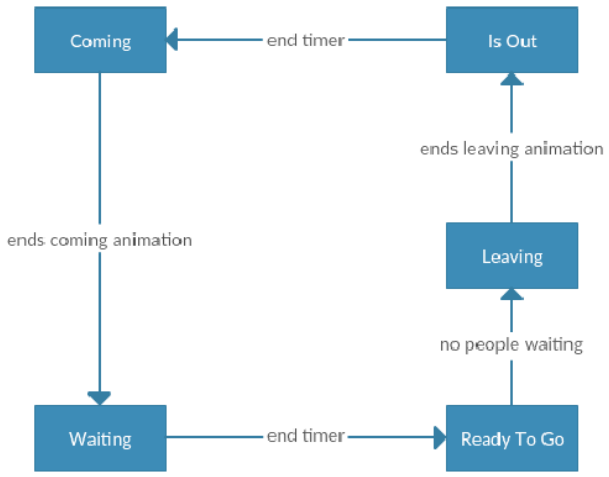


Figura 30: Diagrama d'estats dels trens

Els trens inicialment es troben en l'estat de Waiting. Cadascun d'ells té un temps d'espera aleatori d'entre 1 i 3 minuts. En aquest estat, els trens esperen fins què queden 15 segons per marxar, i llavors envien una senyal al sistema, afegint a la cua de reproducció dels altaveus el so de sortida de la via en la que es troba el tren. Quan el tren acaba el timer, passa a l'estat Ready To Go.

En aquest estat, els trens esperen a tota la gent que vol i pot entrar. Un cop la gent ha entrat, els trens tanquen les portes i les plataformes i passen al següent estat Leaving, reproduint el so del tren en moviment. Tant en l'anada com en la tornada, els trens, per passar de l'estat leaving a Is Out i del Coming al Waiting, s'han utilitzat els events de les animacions per, un cop finalitzada l'animació, es canviï d'estat automàticament.

Quan el tren es troba en l'estat Is Out, aquest elimina les persones que es troben dins del tren al recompte total de persones de l'escena. D'aquesta manera, els generadors de personatges generaran les persones que hagin marxat amb el tren. Llavors, el tren s'espera a què el timer d'espera finalitzi.

Un cop ha finalitzat, el tren entra en l'estat Coming. Aquest és el segon estat on els agents poden sortir definitivament de l'estació. Primer s'eliminen algunes de les

persones que hi ha a dins el tren en funció d'un percentatge, per tal que no s'ompli l'estació de molta gent, ja que els generadors han generat més personatges i no volem que baixi massa el rendiment de l'escena degut al nombre de personatges. Tot i així, hem decidit posar el percentatge ja que és més realista veure a algunes persones tornant amb els trens. Un cop s'han eliminat uns quants personatges, es suma al recompte total de persones en l'escena les persones que hagin tornat a l'estació amb els trens. En aquest estat també s'executa l'àudio dels trens d'arribada.

Un cop els trens arriben a l'estació, obren les portes i tornen a l'estat inicial Waiting, i repeteixen el cicle d'anada i tornada.

12.3.3.7. Estat Go Train

En aquest estat, les persones es dirigeixen cap a una de les zones d'espera de les andanes dels trens. Quan el tren es troba a l'estació, aquestes pujen i s'esperen a que el tren marxi. Un cop el tren ha efectuat la seva transició d'estats i han tornat a l'estació, les persones surten i es dirigeixen cap a la sortida de l'andana corresponent i entren en estat walk.

Precondicio: El personatge no té cap estat nou assignat.

Postcondicio: El personatge té assignat o bé l'estat go traint o bé l'estat walk.

$\underline{CharacterController - CalcularNouEstat()}$

```
k <- Random(1,7);
IF k == 5 THEN
      IF te tiquet THEN
            IF hi ha lloc en alguna andana THEN
                  estat_actual <- go_to_waitzone;
                  destinació_agent <- Punt d'una zona de l'andana lliure
            ELSE estat_actual <- walk;
            END IF
      ELSE estat actual <- walk;
      END IF
END IF
```

Precondicio: Cert.

Postcondicio: La persona ha estat eliminada de l'escena o bé ha tornat a l'estació i se li ha assignat l'estat walk.

CharacterController – Update()

```
SWITCH state
      CASE go_to_waitzone:
            IF distancia(posició_actual,posició_desti) < 0.4f THEN
                  estat_actual <- waiting_train;
            END IF
      BREAK;
      CASE waiting_train:
            IF tren en l'andana THEN
                  IF no hi ha ningú sortint AND hi ha lloc al tren THEN
                        AfegirPersonaEntrant(); #afegeix la persona al
                                                nombre de persones totals
                                                que estan entrant al tren
                        destinació_agent <- punt dins del tren
                        EliminarPersonaAndana(); #allibera espai a
                                                   l'andana
                        estat_actual <- entering_train;
                  END IF
            END IF
      BREAK;
      CASE entering_train:
            IF distancia(posició_actual,posició_desti) < mindist THEN
                  EliminarPersonaEntrant();
                  AfegirPersonaTren(); #afegeix persona al total de gent
                                       dins el tren
                  estat_actual <- inside_train;
            ELSE mindist <- mindist + 0.01f;
            END IF
      BREAK;
```

```
      CASE inside_train:
            IF el tren esta marxant THEN
                  estat_actual <- inside_moving_train;
            END IF
      BREAK;
      CASE inside_moving_train:
            IF el tren ha arribat a la parada THEN
                  destinació_agent <- punt de sortida de l'andana;
                  estat_actual <- leaving_train;
            END IF
      BREAK;
      CASE leaving_train:
            IF distancia(posició_actual,posició_desti) < 0.3 THEN
                  EliminarPersonaTren();
                  estat_actual <- walk;
            END IF
      BREAK;
END SWITCH
```

Aquest estat es va pensar en un principi d'utilitzar la mateixa tècnica que s'ha aplicat a l'estat de comprar bitllets. Un cop implementada, es va observar que el posicionament de les persones no era del tot natural, ja que en una estació real, les persones no estan a l'espera d'un tren en cap cua, sinó posicionades aleatòriament al llarg de la zona d'espera. És per això que finalment s'ha adoptat una altra mecànica més aleatòria que s'explicarà a continuació.

En primer lloc, el que es fa és comprovar si l'agent ja ha anat a comprar el bitllet. Si es dona el cas, llavors es comprova que alguna zona d'espera a costat de la parada de trens estigui lliure en funció de la seva capacitat màxima. Les zones d'espera en l'escena estan representades per cubs invisibles al costat de cada una de les entrades dels trens. Si troba alguna posició lliure, el que es fa és obtenir la zona menys ocupada i es selecciona

un punt de destí aleatori dins d'aquesta zona. S'ha realitzat d'aquesta manera ja que així s'evita que s'acumuli gent en un mateix punt i, d'aquesta manera, la gent estigui més distribuïda.

Un cop l'agent arriba al punt de destí, aquest canvia la seva màscara perquè pugui entrar dins el tren. La funció de la màscara s'explicarà en la secció 12.3.5.

L'agent es manté a l'espera fins què el tren arriba a l'estació i obre les seves portes. Llavors, es comprova si no hi ha gent sortint del tren i si hi ha espai disponible dins d'algun del seus vagons. En cas de no ser així, l'agent s'esperaria fins al proper tren. En cas contrari, l'agent es dirigeix cap a un dels destins de dins el vago més proper i allibera el seu espai en la zona d'espera. Els destins estan marcats com a gameobjects, i són els que guarden la capacitat de cada secció dels vagons.

Per tal d'evitar que els agents s'empenyin entre ells per arribar al mateix punt de destinació, el que s'ha fet és anar augmentant la distància requerida per arribar al punt de manera progressiva. D'aquesta manera, tots els agents es quedaran quiets en un moment determinat.

Un cop dins, l'agent queda en espera i entra en joc el funcionament dels trens. Després que els trens hagin efectuat el seu recorregut d'anada i tornada a l'estació, l'agent torna a tenir el control i segueix en l'estat Go Train. A continuació, s'elimina del sistema el tiquet de l'agent que surt del tren i procedeix a sortir d'aquest. A mesura que van sortint tots els agents, van deixant espai lliure als vagons i, un cop ja no queda ningú, les persones que estaven esperant procedeixen a entrar al tren. Un cop l'agent ha sortit del tren, es dirigeix cap al punt de sortida de les zones d'espera, on aquí ja es considera que l'agent torna a estar en l'estat Walk.

12.3.3.8. Estat Go Bench

En aquest estat, les persones trien un banc lliure aleatori, es dirigeixen cap a ell i s'assenten durant un temps. Un cop el temporitzador intern finalitza, les persones s'aixequen i calculen un nou estat.

Precondicio: El personatge no té cap estat nou assignat.

Postcondicio: El personatge té assignat o bé l'estat go bench o bé l'estat walk.

$\underline{CharacterController - CalcularNouEstat()}$

```
k <- Random(1,7);
IF k == 6 THEN
      IF hi ha algun banc lliure THEN
            estat_actual <- go_to_bench;
            destinació_agent <- punt per poder seure al banc;
            OcuparBanc(); #el lloc del banc queda ocupat per evitar que
                        un altre agent es posi al mateix lloc
      ELSE estat_actual <- walk;
      END IF
END IF
```

Precondicio: Cert.

Postcondicio: S'ha actualitzat l'estat en el que es troba el personatge.

$\underline{CharacterController - Update()}$

```
SWITCH state
      CASE go_to_bench:
            IF distancia(posició_actual,posició_desti) < 0.2f THEN
                  velocitat_agent <- 0;
                  rotar l'agent en direcció contraria al banc;
                  estat_animacio <- 3;    #animacio de seure
                  estat_actual <- sitting;
            END IF
      BREAK;
      CASE sitting:
```

```
            IF temps límit per estar sentat s'ha acabat THEN
                  DesocuparBanc();
                  estat_animacio <- 0;
                  estat_actual <- activate;
            END IF
      BREAK;
      CASE activate:
            IF l'agent ha acabat l'animacio d'aixecar-se THEN
                  CalcularNouEstat();
            END IF
      BREAK;
END SWITCH
```

En aquest estat, l'agent primer comprovarà a partir del sistema si hi ha algun banc lliure, sense importar la distància a la que es trobi. Si no es així, l'agent entrarà en l'estat Walk. En cas contrari, l'agent es dirigirà cap al lloc del banc on s'ha d'assentar, i aquella posició quedarà com ocupada per evitar que un altre agent pugui seure al mateix lloc mentre ja està ocupat.

Un cop l'agent arribi al punt de destí, aquest rotarà i, un cop estigui alineat amb el banc, començarà l'animació de assentar-se canviant el valor del State a 3. Menter l'agent es trobi assentat, s'executarà l'animació corresponent i estarà en espera fins què el timer corresponent s'activi i l'agent passi al següent estat. Degut a què els models dels personatges no són idèntics, l'animació d'estar assentat s'ha intentat ajustar al màxim des de Mixamo per tal que els braços recolzats a les cames no es vegin flotant. Tot i l'ajust, en alguns models encara hi ha petits defectes que només es podrien solucionar accedint directament i modificant les animacions, però no es tracta d'un problema greu, i la immersió es veurà afectada mínimament.

Quan el timer s'activi, el valor de State passarà a ser 0, i l'agent efectuarà l'animació d'aixecar-se. Un cop dret, es calcularà un nou estat per l'agent.

12.3.3.9. Estat Go Shop

En aquest estat, les persones trien una botiga lliure i es dirigeixen cap a un punt de compra aleatori d'aquella botiga. Un cop han arribat, efectuen el procés de compra i s'esperen a que el punt de pagament es trobi lliure. Quan aquest punt està lliure, es dirigeixen cap a ell, efectuen el procés de pagament i entren en estat walk.

Precondicio: El personatge no té cap estat nou assignat.

Postcondicio: El personatge té assignat o bé l'estat go shop o bé l'estat walk.

$\underline{CharacterController - CalcularNouEstat()}$

```
k <- Random(1,7);
IF k == 7 THEN
      IF hi ha alguna botiga lliure THEN
            estat_actual <- go_to_shop;
            destinació_agent <- punt de compra lliure;
            OcuparPuntDeCompra(); #per tal que un altre agent no vagi
                                    cap al mateix punt de compra
      ELSE estat_actual <- walk;
      END IF
END IF
```

Precondicio: Cert.

Postcondicio: El personatge ha finalitzat la compra i es troba en l'estat walk.

$\underline{CharacterController - Update()}$

```
SWITCH state
      CASE go_to_shop:
            IF distancia(posició_actual,posició_desti) < 0.2f THEN
                  rotar agent cap a l'objecte de compra;
                  estat_animacio <- 2; #estat de tria d'accions;
                  acció_animacio <- 2; #accio de compra
                  estat_actual <- buying;
            END IF
```

```
        BREAK;
        CASE buying:
              IF ha acabat de fer l'accio THEN
                    estat_animacio <- 0;
              END IF
              IF estat_animacio == 0 AND punt de pagament lliure THEN
                    destinació_agent <- Punt de pagament;
                    AlliberarPuntCompra();
                    OcuparPuntPagament();
                    estat_actual <- go_to_pay;
              END IF
        BREAK;
        CASE go_to_pay:
              IF distancia(posició_actual,posició_desti) < 0.2f THEN
                    Rotar l'agent cap al venedor;
                    estat_animacio <- 2; #estat de tria d'accions;
                    acció_animacio <- 2; #accio de compra
                    estat_actual <- paying;
              END IF
        BREAK;
        CASE paying:
              IF ha acabat de fer l'accio THEN
                    estat_animacio <- 0;
              END IF
              IF estat_animacio == 0 THEN
                    AlliberarPuntPagament();
                    estat_actual <- walk;
              END IF
        BREAK;
END SWITCH
```

El primer que es farà si l'agent vol anar a comprar és comprovar que hi ha alguna botiga lliure. En cada botiga s'han col·locat diversos punts de compra i un punt de pagament.

Aquests punts de compra estan orientats cap als objectes que hi ha a les botigues. Degut a què les animacions són limitades, no s'ha pogut fer que els personatges poguessin agafar un objecte i comprar-lo. És per això que el personatge es limita a realitzar una acció sense interactuar amb l'objecte.

Si l'agent troba una botiga qualsevol disponible, obtindrà un dels punts de compra lliures com a destinació i es dirigirà cap allà. Un cop hagi arribat al seu destí, l'agent s'orientarà cap a l'objecte i efectuarà una acció. Un cop finalitzada l'acció, l'agent comprovarà que no hi ha ningú al punt de pagament. Si hi ha algun altre agent, aquest s'esperarà a què finalitzi i llavors deixarà lliure la seva posició i anirà cap al punt.

Un cop arribat al punt, l'agent es girarà cap al venedor i efectuarà una acció per representar que està pagant per l'objecte obtingut. Finalment, quan hagi pagat, deixarà l'espai disponible i entrarà en l'estat Walk.

12.3.4. Sistema d'evacuació

Per a la realització del sistema d'evacuació s'han tingut en compte diversos factors que influeixen tant directament com indirectament en la immersió i reacció de l'usuari a l'hora d'efectuar-lo.

En primer lloc, cal indicar que la simulació no començarà en l'escena de l'estació directament, sinó que l'usuari primer veurà una escena buida, on estarà a l'espera d'instruccions. En aquesta escena, es triarà el percentatge de persones que evacuaran de l'estació en sonar l'alarma de l'estació. S'ha fet d'aquesta manera ja que així es poden provar diferents experiments en funció de les persones que veu l'usuari que marxen corrents de l'estació, i veure si aquest factor influeix d'alguna manera en la reacció i la presa de decisions que fa l'usuari al sentir l'alarma.

Aquest percentatge serà enviat a l'escena de l'estació un cop la simulació hagi començat, i es tindrà en compte quan l'evacuació comenci.

El sistema d'alarma no informarà a l'usuari de cap mena de perill, les úniques fonts d'informació que tindrà seran el so de l'alarma que s'activi en l'estació i la gent corrents. En funció d'això, els experiments que es realitzaran a posteriori veuran quin és el comportament de l'usuari davant d'una situació així, sense previ avís ni indicacions del que ha de fer.

Per tal que l'usuari s'adapti a l'entorn i augmentar l'efecte del factor sorpresa de l'alarma, se li indicaran una sèrie de tasques inicials per tal de poder interactuar amb els elements de l'entorn o anar a diversos llocs. Això serà explicat més detalladament en seccions posteriors.

Un altre factor que s'ha tingut en compte és la sortida que agafaran els personatges a l'hora de l'evacuació. Per veure el comportament de l'usuari davant d'un entorn desconegut, s'ha modificat l'escena per tal que les dues portes laterals siguin més petits i no visibles a simple vista. A més, s'han posicionat les estructures de les botigues de forma que una porta quedi molt més amagada que l'altra.

L'evacuació es durà a terme en funció de les accions de l'usuari. Degut a què l'entorn es realitza en realitat virtual, l'usuari no pot estar exposat massa estona amb els cascos de VR, ja que pot provocar marejos i malestar. És per això que, un cop l'usuari s'hagi acostumat a l'entorn i hagi efectuat la seva primera tasca, s'activarà un timer que, un cop finalitzat, activarà l'alarma. En cas que l'usuari hagi aconseguit efectuar més de 3 accions abans del temps màxim del timer, l'alarma s'activarà automàticament. L'última possibilitat d'activar l'alarma seria manualment a través d'un teclat, depenent de la decisió de la persona que executa l'experiment.

Un cop l'evacuació comenci, els personatges es dirigiran cap a la porta més amagada, i és aquí on es veurà si l'usuari decideix seguir a la resta de la gent, va cap a la porta més visible, es queda a l'estació o fa qualsevol altra cosa. La persona executant la simulació podrà parar-la en qualsevol moment, o fins què l'usuari intenti sortir de l'estació un cop l'alarma hagi estat activada.

La informació que s'emmagatzemarà de l'usuari serà guardada en uns arxius i podrà ser visualitzada a posteriori. Aquesta informació i el procés d'emmagatzematge s'explicarà en apartats posteriors.

12.3.5. Navegació i Local Avoidance

Per a la navegació de l'agent, s'ha aprofitat l'eina de navegació de Unity anomenada Navigation Mesh. Aquesta eina permet creat una estructura de dades que descriu les superfícies de l'escena per on poden caminar els objectes que tenen associada una component NavMeshAgent, i aquests són anomenats agents. L'estructura de dades està formada per polígons convexes, i l'algorisme que utilitza Unity per trobar camins a través d'aquestes superfícies és l'A*.

Els agents que naveguen a través d'aquesta estructura estan identificats com cilindres, i tant la malla de navegació com els agents es poden configurar a través d'uns paràmetres que s'explicaran a continuació. La malla de navegació s'ha de crear abans de l'execució de l'escena i si es vol que es generi en algun objecte geomètric de l'escena s'ha d'indicar prèviament que aquell objecte ha de ser estàtic. (Per defecte, no es pot crear la malla de navegació en un objecte dinàmic).

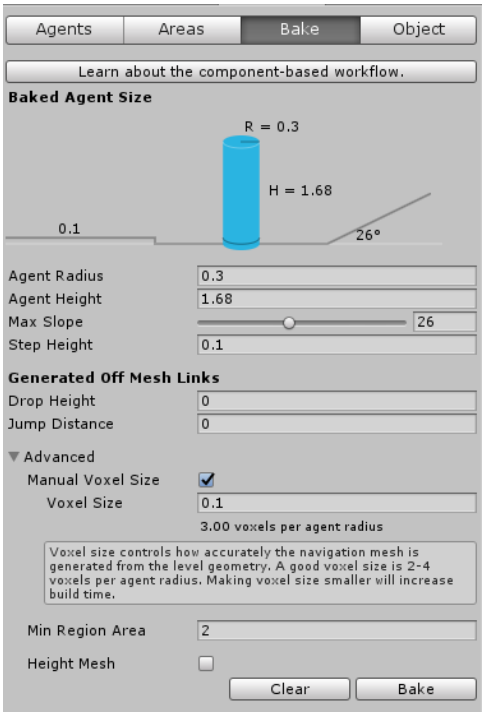


Figura 31: Propietats de la NavMesh

- Agent Radius: Indica el radi màxim de l'agent. A l'hora de crear la malla es té en compte aquest radi com la distància mínima que s'ha de deixar entre l'agent i les parets per evitar col·lisions.
- Agent height: Indica l'altura màxima de l'agent. Fa la mateixa funció que el paràmetre anterior però per la distància entre l'agent i els sostres.
- Max Slope: Indica l'angle màxim de les pendents per les quals l'agent pot caminar.
- Step Height: Indica el marge d'error que l'agent permet a l'hora de caminar a través d'obstacles en altures diferents. Si l'altura d'un punt a un altre supera aquest valor, no hi haurà connexió en la navigation mesh entre aquest dos punts.
- Voxel Size: indica la qualitat dels polígons creats en la navigation mesh. Si aquest paràmetre augmenta, la qualitat disminueix, i viceversa.
- Min Region Area: Indica l'area mínima perquè es pugui crear una navigation mesh.

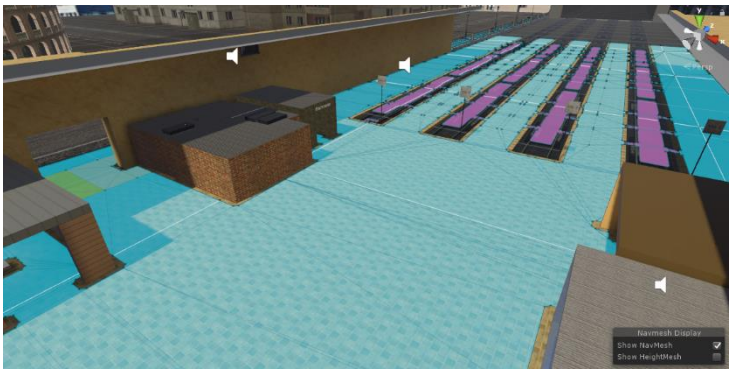


Figura 32: Malla de navegació creada

Com es pot observar en la figura anterior, la malla de navegació manté la distancia indicada a les parets i zones on no es pot caminar per el radi de l'agent. A més, es pot observar dues zones, una blava i una altra rosa. Aquestes zones representen dues àrees

diferents de la malla de navegació, i un dels seus usos és perquè certs agents puguin caminar en unes zones i altres no. Per exemple, en els trens ens interessa que les persones que no vulguin agafar el tren no puguin caminar per dins d'ell, és per això que s'han assignat una àrea per la zona de l'estació i una altra per la zona interior dels trens. Més endavant en aquesta secció s'explicarà un altre cas on s'ha utilitzat aquesta característica.

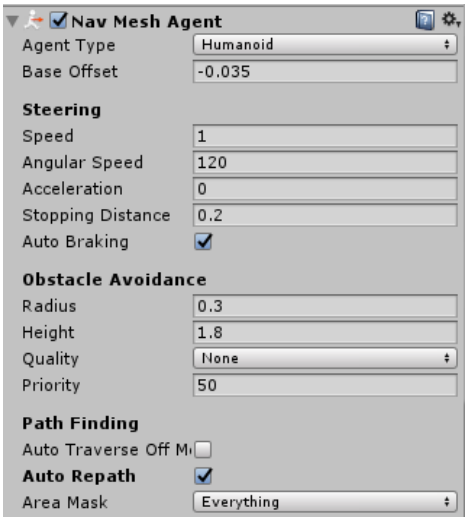


Figura 33: Paràmetres de l'agent

Un cop ja hem configurat i creat la malla de navegació, per tal que els personatges es puguin moure a través d'aquesta malla cal afegir una component anomenada Nav Mesh Agent. Aquesta component també té uns paràmetres de configuració, però els que ens interessen només és la velocitat a la que es desplaça l'agent, la velocitat angular, la distància a la que l'agent s'atura respecte el seu punt de destí, l'autorepath, que permet calcular automàticament un nou camí si l'agent detecta que el camí actual queda invàlid i la màscara de navegació. Aquest últim ens permet indicar a l'agent per quines àrees de la malla de navegació pot caminar i quines no, i aquesta màscara es canvia en funció del comportament de l'agent.

Un cop explicat el sistema de navegació de Unity, es passarà a explicar el script que s'ha creat per al moviment dels agents en l'escena.

Precondicio: Cert.

Postcondicio: S'han actualitzat els valors de velocitat i direcció en l'animació del personatge.

$\underline{ControllerMovementAI - Update()}$

```
IF component agent activa AND no ha arribat al desti THEN
```

```
        v <- velocitat_agent;
        v <- DetectarColisio(v); #calculem el nou vector en funció de
                                  la gent a esquivar, aquesta funció
                                  s'explica detalladament en aquesta
                                  secció.
        v <- InverseTransformDirection(v); #passem de world-space a local
        directionValue <- Atan2(v.x,v.z);
        velocityValue <- v.z;
        rotacioExtra(); #Apliquem rotació a la component transform per
                        ajudar a l'agent a girar degut a què estem
                        sobreescrivint el mètode OnAnimatorMove()
        velocitat_animacio <- velocityValue;
        direcció_animacio <- directionValue;
END IF
```

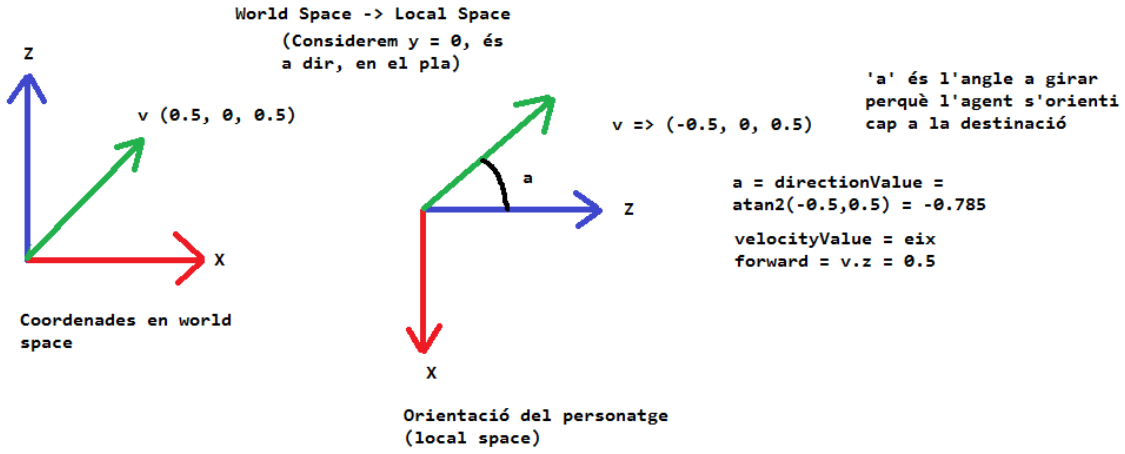


Figura 34: Exemple de càlcul dels valors de direcció i velocitat

Quan li assignem un punt de destinació de la malla de navegació, la component NavMeshAgent s'encarrega de calcular en world-space el vector velocitat que l'agent hauria de seguir. És per això que el primer el que fem és aconseguir la velocitat desitjada de l'agent.

A continuació es fa el càlcul del local avoidance per actualitzar el vector velocitat en cas que estigui a punt de xocar amb algun altre agent. Aquest càlcul s'explicarà més endavant en aquesta secció i també es comentarà per què s'ha fet l'elecció de crear-ne un en comptes d'utilitzar el local avoidance que Unity porta incorporat.

Un cop efectuat el càlcul de col·lisions, es transforma el vector velocitat de world-space a local-space per tal de calcular quina és la velocitat i direcció que ha d'adaptar l'agent. La seva direcció és l'angle resultant de calcular l'arctangent de la component x i la component z (eixos right i forward), i la seva velocitat esta representada per la component z (l'eix forward).

Després de trobar els valors de gir i velocitat de l'animator, hem d'aplicar una rotació manualment a la component transform en funció del directionValue. Això s'ha de fer ja que hem sobreescrit el mètode OnAnimatorMove() per tal d'actualitzar la velocitat del rigidbody amb la velocitat que té l'animació actual, i al sobreescriure el mètode nosaltres tenim el control tant de la rotació com de la translació de l'animator.

Finalment, canviem els paràmetres de l'animator de la Velocity i la Direction amb els valors trobars anteriorment.

Un cop vist el moviment dels agents, cal veure com es comporten quan col·lisionen entre ells. Unity té per defecte un sistema de local avoidance senzill però eficient que s'anomena Reciprocal Velocity Obstacles [18]. A més, en la component NavMeshAgent es poden configurar certs paràmetres com el radi de detecció, la qualitat o la prioritat que té cada agent. L'últim paràmetre senyala quins agents tenen prioritat sobre els altres, és a dir, si un agent té un valor de prioritat 1, aquest ignorarà els agents amb un valor de prioritat més elevat.

El principal problema que s'ha trobat és, degut a la gran quantitat d'obstacles i interseccions que hi ha en l'escena a causa dels edificis provoca que els agents es quedin o bé parats en zones molt estretes com la porta principal o les portes de les botigues, o bé amb problemes de flickering per les animacions. Això ha portat a pensar en la

implementació d'un sistema extern que ajudi a la mobilitat de les persones a través de l'escena.

La primera alternativa que s'ha provat ha sigut la implementació de Proxy Agents [19]. Hi ha tres tipus de proxies que es poden implementar:

- Aggresive: Aquests proxies formen part de l'agent, estan posicionats davant i quan s'apropa a agents que no tenen proxies aquest els repel·leix i li donen pas.

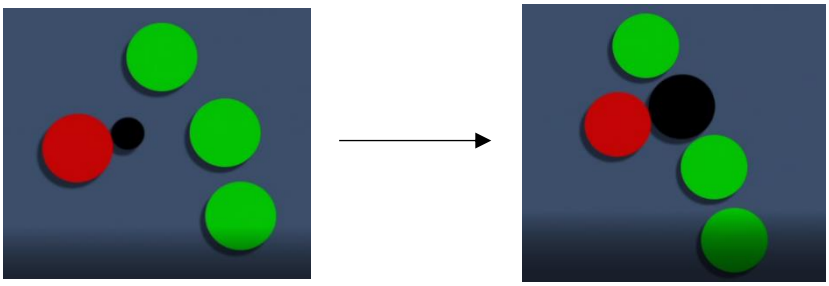


Figura 35: Aggresion proxies

- Priority: Aquests proxies estan col·locats de forma estratègica a les zones estretes, de forma que quan un agent s'apropa per una de les bandes, el proxie comença a créixer i evita que els agents de l'altra banda passin i formin un bloqueig.

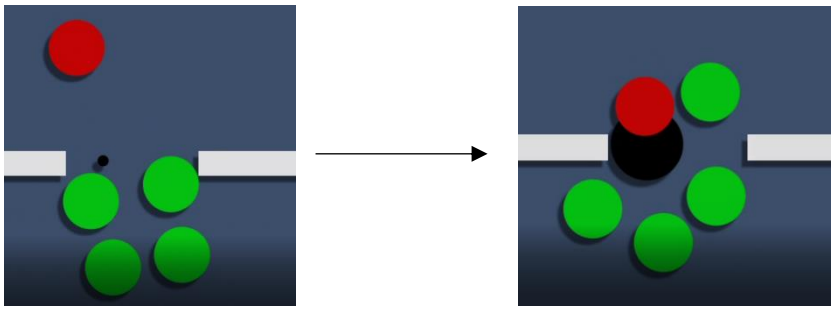


Figura 36: Priority proxies

- Trailblazer: Aquests proxies es creen a mesura que l'agent avança, i van desapareixent a mesura que passa el temps.

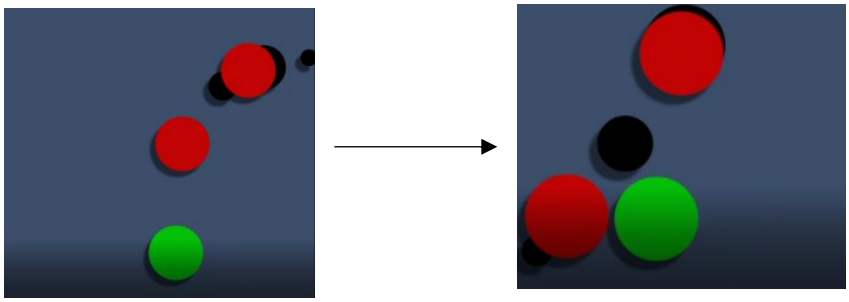


Figura 37: Trailblazer proxies

Observant la naturalesa del problema inicial, els proxies que més s'adapten per a la solució que busquem i que finalment hem implementat són els priority proxies. Aquest s'han implementat de la següent manera.

En primer lloc, s'han col·locat gameobjects amb colliders a fora i a dins de l'estació per identificar els agents que entren i que surten d'ella quan col·lisionen amb aquests, donant prioritat a la gent que surt. D'aquesta manera, si hi ha algun agent que surt de l'estació, els proxies s'activaran i creixeran, bloquejant als agents que volen entrar.

En segon lloc, s'han col·locat els proxies a les entrades com a gameobjects amb colliders esfèrics, juntament amb uns detectors que detecten si hi ha algun agent de dins l'estació apropant-se.

El funcionament d'aquests llavors és, si alguna persona col·lisiona amb el detector, aquest activa el proxie i el fa créixer a una velocitat constant, fins a un radi màxim. Si algun agent col·lisiona amb el proxie, la velocitat a la que va s'inverteix i d'aquesta manera va cap enrere, deixant pas a la gent que surt. El problema trobat a l'hora d'implementar aquest sistema ha sigut el següent:

Figura 38: Acumulació d'agents esperant per entrar amb Priority proxies

Degut a què hi havia un nombre constant de gent sortint de l'estació, els proxies estaven actius la majoria del temps. Això ha provocat que s'acumulin molts agents esperant fora i acaben formant un bloqueig a la gent que surt, provocant un efecte contrari a la solució buscada. És per això que aquest sistema ens ha portat a pensar en una altra alternativa, tot i que el codi i el sistema implementat s'ha deixat dins el codi font final de l'aplicació per si es volgués reutilitzar en algun moment.

La segona alternativa a evitar el col·lapse d'agents en les entrades ha sigut la creació de zones d'entrada i sortida en la malla de navegació. Aquest sistema és similar al de proxies en quant a la detecció de les persones dins i fora de l'estació. A partir d'aquí, el que es fa és crear una àrea d'entrada i una de sortida, similar a les regles socials que s'utilitzen a la vida real d'anar sempre per la dreta.

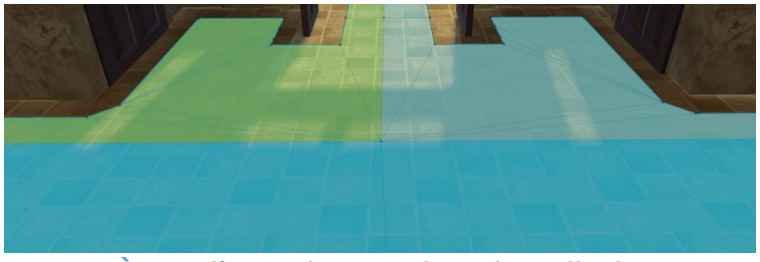


Figura 39: Àrees d'entrada i sortida en la malla de navegació

Com es pot observar, la zona més clara a la dreta és la zona de sortida o leaveZone, i la zona de color verd és la zona d'entrada a l'estació o enteringZone. El que es fa llavors és, mitjançant codi, canviar la màscara de navegació dels agents en funció de si estan dins de l'estació o fora, tenint en compte quan surten de la zona d'entrada o de sortida per evitar que l'agent es desplaci instantàneament cap a un punt vàlid de la malla. El script associat als colliders interiors és el següent.

Precondicio: Cert.

Postcondicio: S'ha actualitzat el valor de l'agent que indica si es troba dins de l'estació a true o bé s'ha afegit l'agent a la cua per actualitzar el valor posteriorment.

$\underline{BoxIn - Update()}$

```
n <- nombre de persones que estan entrant però falta actualitzar la
      màscara de navegació
IF n > 0 THEN
      FOR i=0 to n DO
            agent <- Component agent del gameobject charsToEnter[i];
            p <- existeix punt dins de la zona de navegació d'entrada
                  al voltant de l'agent en un radi.
            IF !p THEN
                  Actualitzar màscara agent;
                  Agent.setInside(true); #Indica que l'agent es troba
                                         dins
                  charsToEnter[i].Remove();
                  i <- i - 1;
            END IF
      END FOR
END IF
```

Precondicio: Detecta col·lisió amb la component collider d'un agent.

Postcondicio: S'ha actualitzat el valor de l'agent que indica si es troba dins de l'estació a true o bé s'ha afegit l'agent a la cua per actualitzar el valor posteriorment.

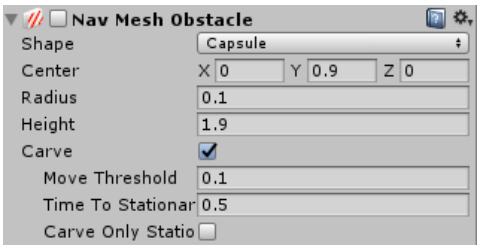
$\underline{BoxIn - OnTriggerEnter()}$

```
other <- Collider que col·lisiona
IF other és un agent AND other no es troba dins de l'estació THEN
      agent <- Component agent associat amb other
      p <- existeix punt dins de la zona de navegació d'entrada
                  al voltant de l'agent en un radi.
      IF !p THEN
            Actualitzar màscara agent;
            Agent.setInside(true);
      ELSE
            Afegir el gameobject associat amb other a la llista de
            persones esperant a actualitzar la seva màscara
      END IF
END IF
```

En el cas del script associat als colliders exteriors és similar als interiors però fent la comprovació i actualització de les màscares corresponents a la zona exterior.

Un cop implementada aquesta solució, ha resultat molt més efectiva que la implementació dels proxies, i s'ha eliminat completament l'acumulació de gent en les zones de les entrades. Tot i això, els problemes relacionats amb el flickering degut al local avoidance de Unity encara persisteixen, cosa que ens ha portat a pensar en l'alternativa de crear un sistema de local avoidance propi.

A més, cal esmentar també uns problemes relacionats amb una altra component relacionada amb la Navigation Mesh que ens ha portat a trobar la mateixa alternativa. Aquesta és la NavMeshObstacle que, un cop associat a un gameobject, actua com obstacles dins de la malla de navegació. Aquest se li pot activar la opció de Carve, on el seu funcionament és modificar la malla de navegació i crear un "forat" en la zona en la que es troba l'objecte amb aquesta opció activada.



Aquest component sembla útil en un principi, ja que es pot associar a l'usuari i, d'aquesta manera, els agents esquivaran automàticament a l'usuari degut a què s'ha creat un forat a la malla.

Figura 40: Paràmetres NavMeshObstacle

A més, també s'havia utilitzat per les persones quan entren en l'estat de seure a un banc, i així evitar col·lisionar amb la gent que passa al seu voltant.

Tot i això, aquesta opció s'ha descartat finalment, ja que s'ha observat que la opció Carve no funciona correctament. El problema principal és que, si l'usuari es movia per l'escena mentre tenia l'obstacle actiu o les persones es tornaven obstacles al seure, certa gent del voltant no recalculaven be el camí cap al seu destí, i es quedaven xocant contra els obstacles. No s'ha pogut saber si realment això era un problema de la pròpia funcionalitat de Unity, per la qual cosa ha portat a pensar definitivament en la implementació del local avoidance. D'aquesta manera la col·lisió entre jugador-agent es pot evitar, la gent que passa per costat de la gent asseguda també els esquivarà correctament i els problemes de flickering es solucionen. A continuació s'explicarà el procediment que s'ha seguit per a la implementació d'aquest.

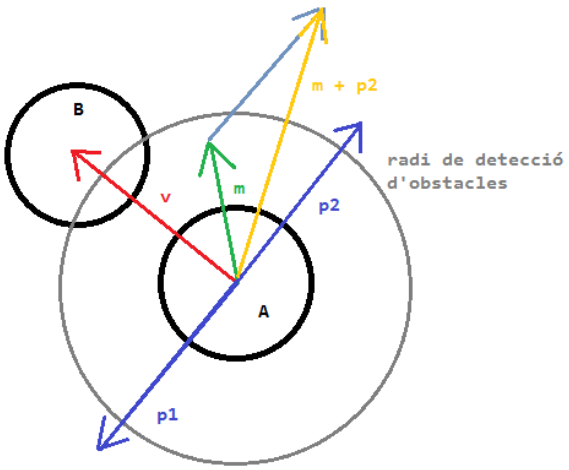


Figura 41: Primera fase de local avoidance

Primer es va plantejar el problema com una col·lisió entre agent i obstacle, és a dir, els agents tractaven a la resta d'agents com a obstacles quiets. D'aquesta forma, si tenim dos agents A i B, i A vol evadir a B el que es fa és calcular el vector V obtenint la posició de B, i es calculen els dos vectors perpendiculars a V, p1 i p2, i es normalitzen. Llavors es comprova quin dels dos és més proper a M, el vector de moviment de l'agent A. Un cop comprovat, es multiplica p2 per un valor aleatori entre 1 i 1.2 per tal que la direcció triada pugui variar respecte la direcció que calculi l'altre agent i es suma a M. Llavors, es normalitza el vector resultant i a partir d'aquest es calcula la velocitat i l'orientació de l'agent.

Fins aquí, el local avoidance es similar al de Unity, però encara hi havia el problema de flickering. S'ha comprovat que el flickering sorgeix en el cas que un agent vol anar cap a la dreta i un altre cap a l'esquerra, per exemple, en cantonades o encreuaments, com es pot veure en la següent imatge:

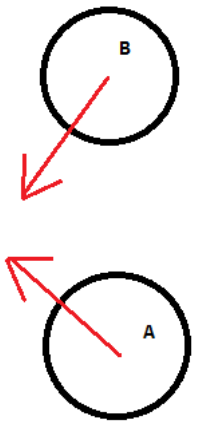


Figura 42: Cas d'encreuament d'agents

És per això que s'ha aplicat una tècnica semblant a la de les zones d'entrada i sortida. Si l'agent A es troba amb un agent B, en funció del seu vector de moviment el deixarà passar o l'intentarà esquivar.

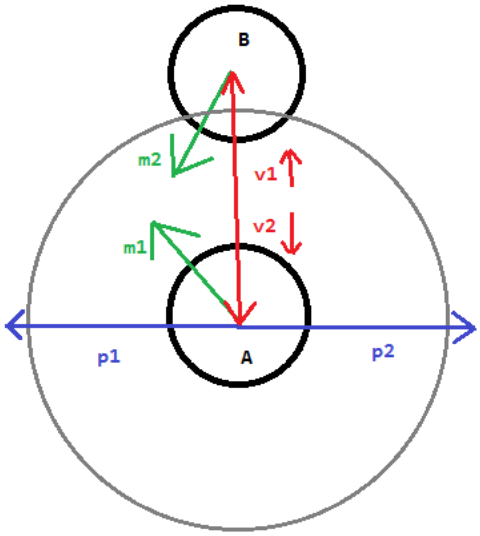


Figura 43: Segona fase del local avoidance

En aquest cas, l'agent A detecta l'agent B i calcula els vectors perpendiculars a v1, que són p1 i p2. Un cop calculats i normalitzats, abans de fer la comprovació del més proper, es calcula cap a on es dirigeix l'agent B respecte el seu propi vector v2 i els seus perpendiculars. Si l'agent B es dirigeix cap a la dreta com ho fa en la figura, l'agent A no calcularà el vector moviment, i el posarà a 0 fent que l'agent es quedi quiet fins què B acabi passant per davant seu. Si l'agent B es dirigeix cap a la dreta, llavors es calcularia el mateix que s'ha explicat anteriorment. Des del punt de vista de l'agent B, no es faria aquest càlcul, ja que si es dirigeix cap a la dreta sempre calcularà el vector moviment resultant. En resum, si un agent detecta un altre per la seva dreta, li deixarà pas, sinó s'evitaran normalment i continuaran el seu camí. Cal indicar que sempre es triarà per evadir l'agent més proper.

Per al local avoidance respecte a l'usuari, només es té en compte la primera part explicada, ja que l'usuari no es considera com a agent i el seu desplaçament no es pot preveure.

Amb aquest mètode es resol el problema de flickering en encreuaments i el resultat es acceptable tant en rendiment com visiblement. El procediment associat amb el local avoidance és el següent.

Precondicio: Es detecta un agent proper a punt de col·lisionar.

Postcondicio: S'ha actualitzat el vector de moviment per tal d'evadir l'agent proper.

ControllerMovementAI – DetectCollision()

```
move <- vector moviment de l'agent;
mg <- magnitud original del vector move;
col <- agent més proper;
IF col != null THEN
      IF col és un agent THEN
            v <- vector cap a l'agent normalitzat;
            angle <- angle de v respecte move;
            max_angle <- angle màxim per evadir als agents;
            IF angle < max_angle THEN
                  p1 <- perpendicular esquerra respecte v;
                  p2 <- perpendicular dret respecte v;
                  IF Angle(p1,move) < Angle(p2,move) THEN
                        dir <- direcció de l'agent proper;
                        IF dir == esquerra THEN
                              move <- move.normal + p1*Random(1f,1.2f);
                              move <- move.normal * mg;
                        ELSE move <- Vector3(0,0,0);
                        END IF
                  ELSE
                        move <- move.normal + p2*Random(1f,1.2f);
                        move <- move.normal * mg;
```

```
                END IF
            END IF
        ELSE IF col és l'usuari THEN
            v <- vector cap a l'agent normalitzat;
            angle <- angle de v respecte move;
            max_angle <- angle màxim per evadir als agents;
            IF angle < max_angle THEN
                p1 <- perpendicular esquerra respecte v;
                p2 <- perpendicular dret respecte v;
                IF Angle(p1,move) < Angle(p2,move) THEN
                    move <- move.normal + p1*Random(1f,1.2f);
                    move <- move.normal * mg;
                ELSE
                    move <- move.normal + p2*Random(1f,1.2f);
                    move <- move.normal * mg;
                END IF
            END IF
        END IF
END IF
```

12.4 Navegació de l'usuari

En aquesta secció s'explicarà tot el que ha comportat la creació dels elements necessaris per a què l'usuari es pugui veure immers en l'entorn, començant per la creació del moviment previ a l'extensió de la realitat virtual i les opcions que s'han implementat, com s'ha fet la interacció amb els elements de l'entorn i el procés seguit per al tracking i l'emmagatzematge de les dades de l'usuari en la simulació.

12.4.1. Càmeres i moviment

L'avatar que s'utilitza per a representar l'usuari en l'escena s'ha creat de la mateixa manera que els models dels personatges, així com les animacions que s'utilitzen per a moure l'avatar, també són les mateixes ja que d'aquesta manera s'aprofita el controller creat per a l'animator dels personatges.

Per tal de col·locar la càmera al cap del model, s'ha modificat la textura de la zona del cap per tal de fer-la invisible. Abans de la importació a la realitat virtual, s'ha implementat un sistema de moviment i interacció senzill explicat a continuació.

Precondicio: Cert.

Postcondicio: S'ha actualitzat el valor de velocitat de l'animació de l'usuari.

$\underline{MovementPlayer - Update()}$

```
#iVel <- valor d'interpolació per a la velocitat;
#iCam <- valor d'interpolació per a la càmera;
#velocitat <- valor de velocitat de l'avatar de l'usuari;
#posicioCam <- posició de la càmera final;
IF Input("W") THEN
      IF Input("SHIFT") THEN
            IF iVel < 1f THEN
                  iVel <- iVel + 2f * temps de frame;
            END IF
            IF iCam < 1f THEN
                  iCam <- iCam + 2f * temps de frame;
            END IF
      ELSE
            IF iVel < 0.5f THEN
                  iVel <- iVel + 2f * temps de frame;
            ELSE iVel <- 0.5f;
            END IF
            IF iCam > 0 THEN
                  iCam <- iCam – 1f * temps de frame;
            END IF
      END IF
ELSE
      IF iVel > 0.1f THEN
            iVel <- iVel – 2f * temps de frame;
```

```
      ELSE
            iVel <- 0;
      END IF
END IF
velocitat <- interpolar 0f amb velocitat màxima en funció de iVel; #iVel
      va de 0 a 1, si és més petit que 0 es considera 0, i si és més
      gran que 1 es considera 1;
posicioCam <- interpolar posició de la càmera inicial amb posició de la
               càmera inicial desplaçada cap endavant en funció de iCam;
velocitat_animacio <- velocitat;
```

El primer que s'efectua és comprovar si l'usuari està parat o en moviment, és a dir, augmentar o disminuir la velocitat de l'avatar en funció de la tecla W, i canviar entre córrer o caminar en funció de la tecla Shift. A l'hora, també s'efectua un càlcul per el valor d'interpolació de la càmera. Quan l'usuari comença a córrer, la càmera s'inclina cap endavant per augmentar la sensació de què es troba corrents, i a la inversa quan passa de córrer a caminar.

12.4.2 Interacció

La interacció bàsica creada en un principi es basa en l'ajuda del sistema tagging de Unity explicat anteriorment. Un cop els objectes amb els que es vol interactuar se li han assignat un tag, a partir d'una tècnica anomenada raycasting s'executa l'interacció amb l'objecte.

Aquesta tècnica permet crear un raig des d'un punt inicial en una direcció i amb una distància màxima si és necessari. Si el raig entra en contacte amb algun objecte que té un collider de l'escena, aquest retorna informació relacionada amb l'objecte i l'emmagatzema en una estructura anomenada RayCastHit, on d'aquest es pot obtenir paràmetres com el collider amb el que ha col·lisionat o la normal en el punt de col·lisió. Entre aquests paràmetres també es pot obtenir el tag de l'objecte col·lisionat. D'aquesta manera el que es fa és llançar el raig des de la càmera de l'usuari en línia recta i detectar si davant hi ha algun objecte amb el que es pot interactuar obtenint el seu tag.

Segons el seu tag, s'han dividit els objectes en 4 grups.

- Portes: Si s'interactua amb una porta, s'activa l'animació d'obrir o tancar en funció de si aquesta es troba oberta o tancada.
- Màquina de bitllets o diners: Si s'interactua amb una màquina es desactiva la rotació i el moviment de l'usuari, s'efectua una de les accions de l'animator i un cop acabada, es torna a activar la rotació i el moviment. Aquesta funcionalitat s'ha desactivat a posteriori degut a què l'extensió a la realitat virtual no és recomanable bloquejar completament la rotació i moviment quan l'usuari es troba immers.
- Lavabos: Aquests són tractats a part ja que al interactuar amb ells s'efectua un so concret.
- Resta: Tots els altres objectes, un cop s'interactua amb ells només es té en compte que ja s'ha interactuat amb ells dins del sistema, i s'activa un so discret per avisar a l'usuari que ha interaccionat correctament amb l'objecte.

Dins del sistema hi ha una llista amb el tag de tots els objectes amb els que es pot interactuar. D'aquesta manera no es pot interactuar amb un objecte més d'una vegada.

Més endavant, s'ha introduït una nova característica per tal que l'usuari pugui veure amb quins objectes pot interactuar i quins no, que és la de ressaltar l'objecte quan aquest el selecciona. Cal dir que aquesta funcionalitat s'ha creat un cop feta l'extensió a la realitat virtual.

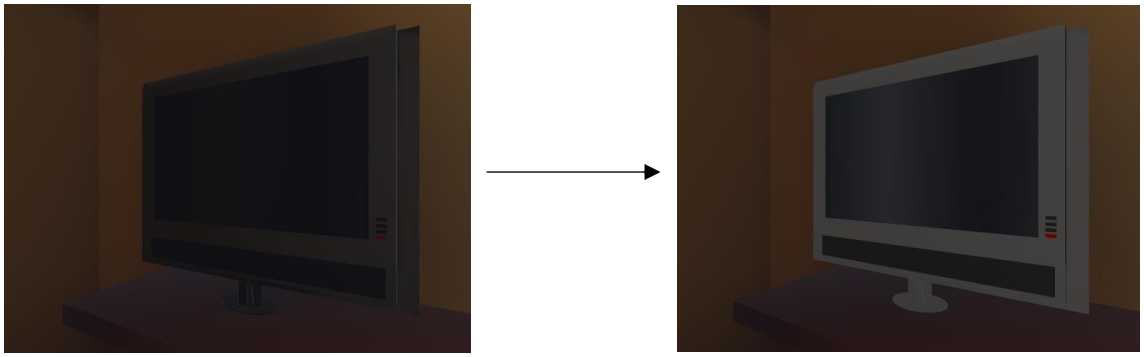


Figura 44: Canvi de color en objecte seleccionable

Per tal de ressaltar els objectes, s'ha utilitzat un shader extern de Unity en comptes de crear un shader personalitzat degut a la falta de coneixement del llenguatge que utilitza Unity per a la creació de shaders. La funció del shader és augmentar d'intensitat el color del material al que se li assigna el shader.

La única diferència en el codi és que, quan es detecta l'objecte amb el que es vol interactuar a partir de raycasting, per a cada un dels materials associats a l'objecte es canvia el shader al shader que resalta l'objecte. Si l'usuari deix d'apuntar l'objecte, es torna a canviar al shader per defecte de Unity.

12.4.3. Enregistrament i emmagatzematge de dades

Un cop la simulació hagi acabat, per tal d'analitzar el comportament de l'usuari a posteriori s'ha decidit emmagatzemar les dades relacionades amb la posició, orientació i accions de l'usuari, juntament amb la posició de les persones que es troben a l'estació.

La freqüència d'emmagatzematge és de 0.1, és a dir, cada 0.1 segons es capturaran les dades respecte a l'orientació i les posicions de l'usuari i les persones i es guardaran en memòria.

Per tal d'emmagatzemar les dades, s'ha creat una classe SerializableData, on aquesta conté els següents atributs:

```
Class SerializableData {
      List<Vector3> pos;
      List<float> time;
      Vector3 posPlayer;
      Float rotYPlayer;
      Float timePlayer;
}
```

Tenim una llista de vectors que representen les posicions de les persones, una llista de floats del temps en què s'han capturat cada una de les posicions, la posició del jugador, la rotació del jugador i el temps de captura de les dades del jugador.

Per guardar aquesta estructura, és necessari serialitzar prèviament les dades. És per això que s'ha utilitzat l'ajuda de la classe BinaryFormatter de .NET per tal de serialitzar les dades automàticament. El principal problema d'això és que Vector3 és una estructura pròpia de Unity, i no es pot serialitzar amb aquesta classe. Per tal que BinaryFormatter pugui serialitzar les dades s'ha utilitzat un asset on es proporciona una classe que utilitza la classe ISerializationSurrogate per tal que BinaryFormatter pugui serialitzar i deserialitzar Vector3.

Un cop emmagatzemades les dades en memòria, aquestes són serialitzades un cop finalitzada la simulació o, si la persona que executa la simulació vol emmagatzemar-les fins un moment concret clica la tecla G.

12.5. Visualitzador 2D

Per tal de visualitzar les dades, s'ha triat crear un visualitzador amb diverses opcions que esmentarem i explicarem a continuació. Aquesta elecció ha sigut la més adient degut a què es tracta de visualitzar dades de posicionament en un espai determinat, i és per això que s'ha creat un entorn a part on es pot veure el mapa de l'estació i la posició de les persones i de l'usuari en tot moment, tenint en compte també l'orientació de l'usuari.

12.5.1. Disseny

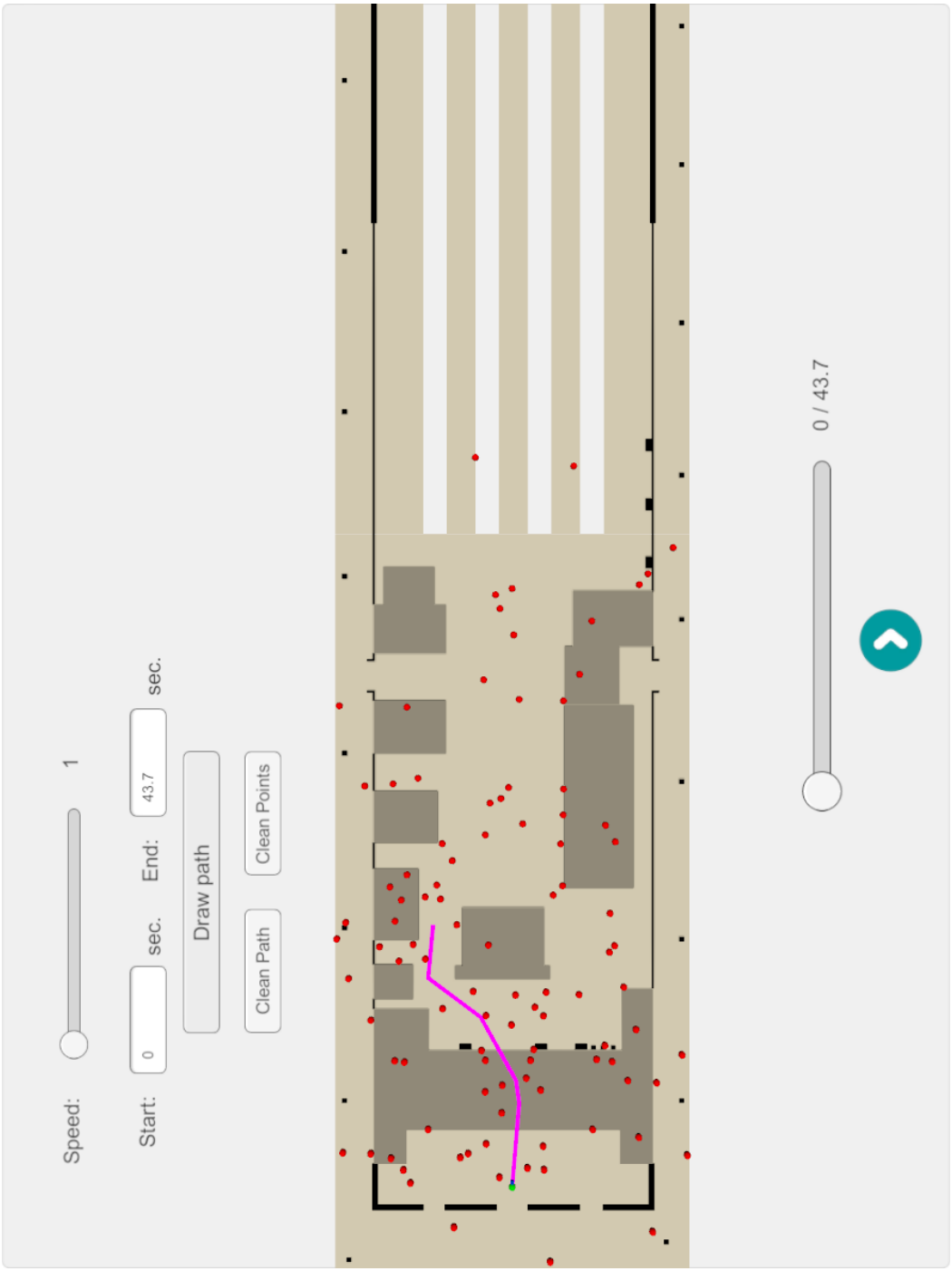


Figura 45: Disseny del visualitzador de dades 2D

Les opcions disponibles en el visualitzador són les següents:

- Speed: Permet ajustar la velocitat a la que s'executa la reproducció de les dades de posició.
- Start/End: Permet indicar el moment inicial i finals del camí que ha fet l'usuari en el tram indicat en segons.

- Draw Path: Dibuixa el camí fet per l'usuari segons els dos paràmetres anteriors, start i end.
- Clean path: Permet eliminar el camí del visualitzador.
- Clean points: Permet eliminar els punts que representen les posicions.

A sota del visualitzador tenim la barra de reproducció, amb el temps total de la simulació en segons, juntament amb un botó per reproduir o pausar la visualització. Com es pot observar, els punts vermells representen les persones, mentre que el punt verd representa l'usuari, juntament amb una barra blava que indica la seva orientació. Quan comença la reproducció, la posició de l'usuari i de les persones es va actualitzant i en qualsevol moment es pot pausar.

També es pot visualitzar el camí recorregut per l'usuari en un tram de la simulació. Aquest camí apareix marcat al mapa de color rosa, i s'han implementat les opcions per esborrar-lo o esborrar els punts per tal de visualitzar-lo millor.

12.5.2. Sistema de càrrega de les dades

Les dades s'emmagatzemen a la carpeta per defecte d'AppData de l'usuari en el cas de Windows. Per carregar-les també s'utilitza la deserialització i es carreguen en una instància de la mateixa classe SerializableData utilitzada anteriorment per emmagatzemar-les en memòria.

Un cop carregades, els punts es dibuixen en funció de la freqüència d'emmagatzematge en què s'han guardat les dades, i aquesta també es fa servir per indicar el temps de reproducció actual.

Per veure els punts en pantalla el que es fa inicialment és crear tantes esferes com nombre de posicions guardades en un moment determinat de la simulació. Aquesta quantitat d'esferes no es destrueix i es crea totalment a cada moment, sinó que es va actualitzant en funció del nombre de posicions en cada moment de la simulació. La posició del jugador també es representa amb una esfera però s'identifica a part de les esferes de les persones.

12.6. Ampliació a la realitat virtual

En aquesta secció s'introduirà l'entorn amb el que s'ha treballat a l'hora de l'extensió a la realitat virtual, els assets que s'han utilitzat i els canvis realitzats en l'usuari per tal d'adaptar-lo a la realitat virtual.

12.6.1. GoogleVR

En un principi, no es disposava del hardware necessari per efectuar l'extensió cap a la realitat virtual, per la qual cosa es va utilitzar un asset de Unity que permetia la implementació dels canvis sense necessitat d'utilitzar el hardware. Per això, primer es va utilitzar els assets de GoogleVR.

L'asset de GoogleVR està orientat a ulleres de realitat virtual sense tenir en compte el seu posicionament en l'espai, només la seva rotació. Per això, primer s'ha configurat la càmera de forma que la seva posició s'actualitzés en funció de la posició de l'avatar de l'usuari de la mateixa manera que es feia anteriorment. El canvi, llavors ha estat en la rotació de l'avatar segons la rotació de la càmera.

La primera implementació ha estat la bàsica, rotar l'avatar en l'eix Y en funció de la rotació de la càmera, ja que l'usuari serà qui tingui control sobre la rotació de la càmera. D'aquesta manera, l'usuari pot anar endavant o endarrere en funció de cap a on estigui mirant.

La segona implementació ha estat permetre una rotació de la càmera dins d'un cert nombre de graus sense moure l'avatar. Com passa amb la realitat, nosaltres podem moure el cap sense haver de rotar tot el cos, i estem limitats fins, aproximadament, la zona de les espatlles, és a dir, uns 90 graus a l'esquerra i a la dreta, a partir d'aquí, el cos es mou si augmentem el nombre de graus.

Hem adaptat aquesta situació i s'ha implementat seguint aquest criteri.

Precondicio: Cert.

Postcondicio: S'ha actualitzat la rotació de l'avatar en l'eix Y en funció de la rotació actual de la càmera.

$\underline{MovementPlayer - updatePlayer()}$

```
#graus <- nombre de graus de llibertat màxim de la càmera respecte el
          cos;
IF diferència de graus de càmera desplaçada a l'esquerra > graus THEN
     rotació_cos_Y <- rotació_camera_Y + 60;
ELSE IF diferència de graus càmera desplaçada a la dreta > graus THEN
     rotació_cos_Y <- rotació_camera_Y – 60;
END IF
```

Un cop implementat, s'ha observat que el desplaçament de l'usuari seguia l'orientació del cos, i degut a què encara no es disposava de cap controlador i l'usuari no tenia control directe sobre l'avatar, aquest per poder canviar d'orientació havia de rotar completament el cap. Aquest moviment d'orientació era poc natural, per la qual cosa es va triar la primera implementació.

Degut a què més endavant s'ha tingut la possibilitat de realitzar l'extensió a la realitat virtual amb un hardware adient, en aquest cas el de HTC Vive, s'han fet els canvis necessaris de l'asset de GoogleVR a l'asset que utilitza aquest hardware, el SteamVR i els explicarem a continuació. El codi realitzat per les dues implementacions explicades anteriorment esta disponible en el codi font del programa.

12.6.2. SteamVR

Abans d'explicar els canvis efectuats en el codi, s'explicarà com funciona el hardware que s'utilitzarà per simular la realitat virtual en aquest entorn.

El hardware consta de dos sensors, un casc i dos controladors. Els sensors serveixen per posicionar el casc i els controladors en l'espai real, i d'aquesta manera poder simular el moviment i la rotació en l'escena virtual. El casc permet a l'usuari veure l'entorn virtual en el que es troba, mentre que els controladors consten de diversos botons on cadascun d'ells es pot programar de la manera que es vulgui.

Figura 46: Hardware HTC Vive

En aquí s'ha disposat de controladors per a realitzar el moviment. Un factor a tenir en compte abans de la realització del moviment a partir dels controladors és que l'exposició contínua amb aquest hardware pot acabar produint mareig. Si l'usuari pot girar la càmera a través dels controladors, aquesta sensació de mareig augmenta, cosa que finalment s'ha decidit no implementar la direcció en funció dels controladors, sinó que s'han pensat dues maneres per tal que l'usuari pugui girar.

La primera és el gir en funció de la càmera, és a dir, el mateix mètode que s'havia utilitzat per les GoogleVR. Com que l'orientació es fa en funció de la càmera, limita el poder moure's cap a una altra direcció mentre es mira al voltant i resulta poc natural. És per això que la segona implementació, aprofitant que disposàvem de controladors, s'ha fet que l'orientació canviés en funció de l'orientació dels controladors. És a dir, s'actualitza l'angle Y de l'avatar en funció de l'angle Y del controlador dret.

Per tal d'aconseguir una immersió més elevada el més natural és que el cos de l'usuari controli el cos de l'avatar que es troba en l'escena, però degut a la limitació que ens posa el hardware actual, aquest canvi no es pot efectuar. És per això que la millor alternativa trobada i implementada ha sigut la d'orientar el cos en funció del controlador.

Degut a què ara es tenia la possibilitat de controlar l'avatar en funció del controlador, la implementació realitzada per a la limitació dels graus de llibertat de la càmera s'ha eliminat, ja que ara es el propi usuari que té la limitació físicament.

A part del moviment, també s'ha utilitzat el sistema d'Inverse Kinematics que proporciona Unity per tal que els braços de l'avatar es moguin en funció de la posició dels controladors. Tot i això, s'ha acabat descartant ja que provoca un flickering en l'animació quan l'usuari es troba en moviment.

13. Conclusions

Un cop s'ha finalitzat el projecte, els objectius plantejats en un principi s'han assolit correctament, creant d'aquesta manera una aplicació capaç de recrear un entorn de realitat virtual en una estació i simular l'evacuació en aquesta.

En el que respecta a la pròpia simulació, s'ha aconseguit realitzar un ambient força realista per part dels elements en l'escena tot i les limitacions que s'han trobat per aconseguir un rendiment estable. Un dels altres factors que també ha afectat de forma directa i positiva a la immersió ha sigut l'animació dels trens i les portes.

D'altra banda, per part del comportament dels agents, s'ha aconseguit que sigui força variat i, d'aquesta manera, augmentar la sensació d'immersió de l'usuari i el realisme en l'escena. A més, s'han estudiat i aplicat correctament les possibilitats respecte a les capacitats de moviment i interacció de l'usuari dins d'aquest entorn, cosa que ha augmentat considerablement la seva immersió en aquest.

A més, tot i que no s'han pogut realitzar experiments reals, si que s'han realitzat diverses fases de test del pilot experiment per part de les persones implicades en aquest projecte i altres voluntaris del grup ViRVIG. Aquestes proves pilot són fonamentals per millorar els detalls d'interacció i navegació i per comprovar, per exemple, que el sistema immersiu no mareja, ja que això suposa un problema molt important en els cascos de realitat virtual.

Tenint en compte tot això, degut a què la immersió i el realisme de l'escena ha augmentat, permetrà que les dades siguin més fiables a l'hora de realitzar experiments en el comportament dels usuaris en aquesta situació.

14. Treball futur

Tot i que el projecte ha aconseguit assolir els objectius principals, hi ha aspectes que poden ser millorats i que afectarien de forma directa o indirecta tant en la immersió de l'usuari com en el recull de dades, però que degut a la limitació de temps que hi havia no s'han pogut dur a terme.

En primer lloc, els models dels elements en l'escena poden ser optimitzats amb els coneixements necessaris respecte al modelatge d'objectes, cosa que ajudaria a millorar el rendiment en l'escena i, d'aquesta manera, augmentaria la bona sensació de l'usuari immers en l'entorn.

En segon lloc, les animacions per aquest projecte han estat limitades, cosa que ha afectat directament al comportament dels agents en l'entorn. És per això que la creació de pròpies animacions seria un punt a favor a l'hora d'augmentar el realisme en el comportament dels agents.

Per últim, en un principi es va tenir la idea de crear un sistema de pathfinding propi que incorporés un local avoidance per tal de millorar la navegació dels agents. Tot i això, finalment no s'ha implementat sinó que s'ha creat un sistema de local avoidance a part utilitzant el pathfinding de Unity com s'ha comentat anteriorment degut al temps que requeria recrear tot un sistema nou.

15. Referències

[1] Unity, «Unity Technologies,» [En línia]. Available: https://unity3d.com/es/. [Últim accés: Febrer 2017].

[2] Wikipedia, «Wikipedia the free encyclopedia,» 6 Febrer 2017. [En línia]. Available: https://en.wikipedia.org/wiki/Crowd_simulation. [Últim accés: Febrer 2017].

[3] A. Beacco i N. Pelechano, «Simulation, Animation and Rendering of Crowds in Real-Time,» Barcelona, 2014.

[4] Unity, «Unity Technologies,» [En línia]. Available: https://docs.unity3d.com/Manual/CollidersOverview.html. [Últim accés: Febrer 2017].

[5] N. Pelechano i J. M. Allbeck, «Feeling Crowd Yet?: Crowd Simulations for VR,» Barcelona.

[6] Wikipedia, «Wikipedia the free encyclopedia,» [En línia]. Available: https://en.wikipedia.org/wiki/Virtual_reality. [Últim accés: Febrer 2017].

[7] N. Pelechano i N. I. Badler, «Modeling Crowd and Trained Leader Behavior during Building Evacuation,» 2006.

[8] Mixamo, «Mixamo,» 2016. [En línia]. Available: https://www.mixamo.com/. [Últim accés: Febrer 2017].

[9] Unity, «Unity Technologies,» 2017. [En línia]. Available: https://docs.unity3d.com/Manual/class-BlendTree.html. [Últim accés: Febrer 2017].

[10] Trello, «Trello business class,» 2016. [En línia]. Available: https://trello.com/. [Últim accés: Febrer 2017].

[11] Blender, «Blender,» [En línia]. Available: https://www.blender.org/. [Últim accés: Febrer 2017].

[12] Sketchup, «SketchUp,» 2017. [En línia]. Available: http://www.sketchup.com/. [Últim accés: Febrer 2017].

[13] Wikipedia, «Wikipedia the free encyclopedia,» Febrer 2017. [En línia]. Available: https://en.wikipedia.org/wiki/Scrum_(software_development). [Últim accés: Febrer 2017].

[14] Sketchup, «3D Warehouse,» Trimble Inc., 2017. [En línia]. Available: https://3dwarehouse.sketchup.com/.

[15] Gimp, «Gimp GNU Image Manipulator Program,» 2017. [En línia]. Available: https://www.gimp.org/.

[16] A. Box, «Acapela Box,» 2017. [En línia]. Available: https://acapela-box.com/AcaBox/index.php.

[17] C. Bregler, «IEEE Xplore Digital Library,» Novembre 2007. [En línia]. Available: http://ieeexplore.ieee.org/abstract/document/4387955/. [Últim accés: 2017].

[18] J. van den Berg, M. Lin i D. Manocha, «IEEE Xplore Digital Library,» Maig 2008. [En línia]. Available: http://ieeexplore.ieee.org/abstract/document/4543489/. [Últim accés: 2017].

[19] H. Yeh, S. Curtis, S. Patil i J. van den Berg, «GAMMA, University of North California,» [En línia]. Available: http://gamma.cs.unc.edu/CompAgent/. [Últim accés: 2017].

[20] M. Kyriakou, X. Pan i Y. Chrysanthou, Interaction with virtual crowd in Immersive and semi-Immersive Virtual Reality systems, Wiley Online Library, 2016.

[21] J. Bruneau, A.-H. Olivier i J. Pettre, Going through, going around: A study on individual avoidance of groups, IEEE, 2015, pp. 520-528.

Registre sprints Scrumme

Blacklog list:

- Afegir animacions persones	DONE
- Millorar entorn 3D	DONE
- Millorar il·luminació	DONE
- Afegir comportaments a persones	DONE
- Entrada i sortida de persones als trens	DONE
- Afegir seqüència evacuació	DONE
- Millora moviment persones	DONE
- Afegir interacció usuari	DONE
- Afegir persones atenent botigues	DONE
- Efectes de so	DONE
- Ampliació projecte VR - Càmeres	DONE
- Problemes il·luminació	DONE
- Millores	DONE
- Sistema de compra NPC's	DONE
- Enregistrar comportament usuari	DONE
- Modificacions escena	DONE
- Modificacions visualització 2D	DONE
- Proxy Agents	DONE
- Error d'evacuació	DONE
- Ampliació a VR	DONE

Sprint 1: (14/02/17 - 28/02/17)

- Afegir animacions persones	DONE
- Millorar entorn 3D	DONE
- Millorar il·luminació	DONE

Sprint 2: (28/02/17 - 14/03/17)

- Afegir comportaments a persones
- Entrada i sortida de persones als trens

Sprint 3: (14/03/17 - 28/03/17)

- Afegir comportaments a persones DONE
- Afegir seqüència evacuació DONE
- Entrada i sortida de persones als trens DONE

Sprint 4: (28/03/17 - 18/04/17)

- Afegir Interacció usuari DONE
- Afegir persones atenent botigues DONE
- Efectes de so DONE

Sprint 5: (18/04/17 - 26/04/17)

- Ampliació projecte VR - Càmeres
- Problemes il·luminació
- Millores DONE
- Sistema de compra NPC's DONE

Sprint 6: (26/04/17 - 09/05/17)

- Ampliació projecte VR - Càmeres DONE
- Enregistrar comportament usuari DONE
- Problemes il·luminació DONE

Sprint 7: (09/05/17 - 23/05/17)

- Proxy Agents DONE
- Modificacions Escena DONE
- Modificacions Visualització 2D DONE

Sprint 8: (23/05/17 - 06/06/17)

- Local Avoidance
- Error d'evacuació DONE
- Ampliació a VR DONE

Sprint 9: (06/06/17 - 13/06/17)

- Local Avoidance DONE

Apèndix

Degut a què la majoria del codi és massa extens, no es posarà tot el codi implementat i es limitarà a posar alguns exemples. Tot el codi font es pot trobar dins del directori corresponent.

Guardar/Carregar dades

```
private void saveData()
    {
        FileStream file = File.Open(Application.persistentDataPath + "/npcData.dat",
FileMode.OpenOrCreate);

        bf.Serialize(file, frames);
        file.Close();

        StreamWriter sw = new StreamWriter(Application.persistentDataPath +
"/variablesData.dat");
        sw.WriteLine(FRAMERATE);
        sw.WriteLine(totalSimulationTime);
        sw.Close();

        StreamWriter sw2 = new StreamWriter(Application.persistentDataPath +
"/actionsData.dat");
        for (int i = 0; i < instance.actionsDone.Count; ++i)
        {
            string s = "Action " + (i + 1) + ": " + instance.actionsDone[i].act + ", Time: " +
instance.actionsDone[i].time;
            sw2.WriteLine(s);
        }
        sw2.Close();
    }


private void loadData()
    {
        if (File.Exists(Application.persistentDataPath + "/npcData.dat"))
        {
            FileStream file = File.Open(Application.persistentDataPath + "/npcData.dat",
FileMode.Open);

            frames = (List<SerializableData>)bf.Deserialize(file);
            file.Close();
        }

        if (File.Exists(Application.persistentDataPath + "/variablesData.dat"))
        {
            StreamReader file = new StreamReader(Application.persistentDataPath +
"/variablesData.dat");
            string fr;
            if ((fr = file.ReadLine()) != null) framerate = float.Parse(fr);
            if ((fr = file.ReadLine()) != null) totalTime = float.Parse(fr);
            file.Close();
        }

        timeline.GetComponent<Slider>().maxValue = frames.Count - 1;
    }
```

Classe Surrogate per fer Vector3 Serializable

```
using UnityEngine;
using System.Runtime.Serialization;
using System.Collections;

public class Vector3SerializationSurrogate : ISerializationSurrogate
{

    // Method called to serialize a Vector3 object
    public void GetObjectData(System.Object obj, SerializationInfo info, StreamingContext
context)
    {

        Vector3 v3 = (Vector3)obj;
        info.AddValue("x", v3.x);
        info.AddValue("y", v3.y);
        info.AddValue("z", v3.z);
    }

    // Method called to deserialize a Vector3 object
    public System.Object SetObjectData(System.Object obj, SerializationInfo info,
                                        StreamingContext context, ISurrogateSelector selector)
    {

        Vector3 v3 = (Vector3)obj;
        v3.x = (float)info.GetValue("x", typeof(float));
        v3.y = (float)info.GetValue("y", typeof(float));
        v3.z = (float)info.GetValue("z", typeof(float));
        obj = v3;
        return obj;
    }
}
```

*La funció initSerializableData() es crida un cop a l'inici de l'execució.

```
private void initSerializableData()
    {
        bf = new BinaryFormatter();
        SurrogateSelector surrogateSelector = new SurrogateSelector();
        Vector3SerializationSurrogate vector3SS = new Vector3SerializationSurrogate();

        surrogateSelector.AddSurrogate(typeof(Vector3), new
StreamingContext(StreamingContextStates.All), vector3SS);
        bf.SurrogateSelector = surrogateSelector;
    }
```

Classe DataManager.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization;
using System.IO;
using UnityEngine;
using UnityEngine.UI;

public class DataManager : MonoBehaviour {

    public Sprite play;
    public Sprite pause;
    public GameObject player;
    public GameObject NPC;
    public float speed;

    private GameObject velText;
    private GameObject velSlider;
    private GameObject start;
    private GameObject timeline;
    private GameObject iniPath;
```

```
    private GameObject endPath;
    private GameObject timer;
    private LineRenderer lineMaker;
    private bool running;
    private float framerate;
    private float totalTime;
    private float actTime;
    private List<GameObject> activeNPCs;
    private GameObject activePlayer;
    [SerializeField]
    private List<SerializableData> frames;
    private BinaryFormatter bf;

        // Use this for initialization
        void Start () {
        lineMaker = GetComponent<LineRenderer>();
        velText = GameObject.Find("ActualSpeed");
        velSlider = GameObject.Find("SpeedSet");
        start = GameObject.Find("Play");
        timeline = GameObject.Find("TimeLine");
        iniPath = GameObject.Find("InputStartSec");
        endPath = GameObject.Find("InputEndSec");
        timer = GameObject.Find("Timer");
        actTime = 0.0f;
        running = false;
        initSerializableData();
        loadData();
        activeNPCs = new List<GameObject>();
        activePlayer = Instantiate(player);
        updateTimer();
        drawPoints();
    }

        // Update is called once per frame
        void Update () {
                if (running)
        {
            if (actTime > framerate)
            {
                if (timeline.GetComponent<Slider>().value >=
timeline.GetComponent<Slider>().maxValue)
                {
                    start.GetComponent<Image>().sprite = play;
                    running = false;
                }
                else
                {
                    ++timeline.GetComponent<Slider>().value;
                    drawPoints();
                    updateTimer();
                    actTime = 0.0f;
                }
            }
            else actTime += Time.deltaTime*speed;
        }
        }

    public void clickButton()
    {
        if (running)
        {
            start.GetComponent<Image>().sprite = play;
            running = false;
        }
        else
        {
            start.GetComponent<Image>().sprite = pause;
            running = true;
        }
    }

    private void loadData()
    {
        if (File.Exists(Application.persistentDataPath + "/npcData.dat"))
```

```
        {
            FileStream file = File.Open(Application.persistentDataPath + "/npcData.dat",
FileMode.Open);

            frames = (List<SerializableData>)bf.Deserialize(file);
            file.Close();
        }

        if (File.Exists(Application.persistentDataPath + "/variablesData.dat"))
        {
            StreamReader file = new StreamReader(Application.persistentDataPath +
"/variablesData.dat");
            string fr;
            if ((fr = file.ReadLine()) != null) framerate = float.Parse(fr);
            if ((fr = file.ReadLine()) != null) totalTime = float.Parse(fr);
            file.Close();
        }

        timeline.GetComponent<Slider>().maxValue = frames.Count - 1;
    }

    private void initSerializableData()
    {
        bf = new BinaryFormatter();
        SurrogateSelector surrogateSelector = new SurrogateSelector();
        Vector3SerializationSurrogate vector3SS = new Vector3SerializationSurrogate();

        surrogateSelector.AddSurrogate(typeof(Vector3), new
StreamingContext(StreamingContextStates.All), vector3SS);
        bf.SurrogateSelector = surrogateSelector;
    }

    private void drawPoints()
    {
        int i = (int) timeline.GetComponent<Slider>().value;
        int dif = activeNPCs.Count - frames[i].pos.Count;
        if (dif < 0)
        {
            //N'hi ha de menys, s'han d'afegir esferes
            for (int j = 0; j < -dif; ++j)
            {
                GameObject s = Instantiate(NPC);
                activeNPCs.Add(s);
            }
        }
        else if (dif > 0)
        {
            //N'hi ha de mes, s'han de restar esferes
            for (int j = 0; j < dif; ++j) Destroy(activeNPCs[j]);
            activeNPCs.RemoveRange(0,dif);
        }

        for (int j = 0; j < activeNPCs.Count; ++j)
        {
            activeNPCs[j].transform.position = new Vector3(frames[i].pos[j].x, 1.0f,
frames[i].pos[j].z);
        }

        if (activePlayer == null) activePlayer = Instantiate(player);
        activePlayer.transform.position = new Vector3(frames[i].posPlayer.x, 1.0f,
frames[i].posPlayer.z);
        activePlayer.transform.localEulerAngles = new
Vector3(activePlayer.transform.localEulerAngles.x, frames[i].rotYPlayer,
activePlayer.transform.localEulerAngles.z);
    }

    private void updateTimer()
    {
        timer.GetComponent<Text>().text = (timeline.GetComponent<Slider>().value * framerate) +
" / " + (timeline.GetComponent<Slider>().maxValue * framerate);
    }

    public void beginDrag()
    {
```

```
        start.GetComponent<Image>().sprite = play;
        running = false;
    }

    public void endDrag()
    {
        updateTimer();
        drawPoints();
    }

    public void setVelocity()
    {
        velText.GetComponent<Text>().text = velSlider.GetComponent<Slider>().value.ToString();
        speed = velSlider.GetComponent<Slider>().value;
    }

    public void drawPath()
    {
        Debug.Log(totalTime);
        float iniSec = float.Parse(iniPath.GetComponent<InputField>().text);
        float endSec = float.Parse(endPath.GetComponent<InputField>().text);

        if (iniSec >= endSec) return;
        if (iniSec < 0) { iniSec = 0; iniPath.GetComponent<InputField>().text = "0"; }
        else if (iniSec > totalTime) { iniSec = totalTime;
iniPath.GetComponent<InputField>().text = totalTime.ToString(); }
        if (endSec < 0) { endSec = 0; endPath.GetComponent<InputField>().text = "0"; }
        else if (endSec > totalTime) { endSec = totalTime;
endPath.GetComponent<InputField>().text = totalTime.ToString(); }

        iniSec = ((int)(iniSec * 10)) / 10.0f;
        endSec = ((int)(endSec * 10)) / 10.0f;
        int posIni = (int)(iniSec / framerate);
        int posEnd = (int)(endSec / framerate);
        int i = posIni;

        Vector3[] positions = new Vector3[posEnd-posIni+1];
        for (; posIni <= posEnd; ++posIni)
        {
            positions[posIni - i] = new Vector3(frames[posIni].posPlayer.x, 1.0f,
frames[posIni].posPlayer.z);
        }
        lineMaker.positionCount = positions.Length;
        lineMaker.SetPositions(positions);

        Debug.Log(iniSec + " " + endSec);

        if (running)
        {
            running = false;
            start.GetComponent<Image>().sprite = play;
        }
    }

    public void cleanPath()
    {
        lineMaker.positionCount = 0;
    }

    public void cleanPoints()
    {
        Destroy(activePlayer);
        for (int i = 0; i < activeNPCs.Count; ++i) Destroy(activeNPCs[i]);
        activeNPCs.Clear();
    }
}
```

OnAnimatorIK() per a l'aplicació d'IK a les mans (desactivat degut a errors)

```
void OnAnimatorIK()
    {
        if (anim != null)
```

```
    {
        if (leftHand != null)
        {
            anim.SetIKPositionWeight(AvatarIKGoal.LeftHand, 1);
            anim.SetIKRotationWeight(AvatarIKGoal.LeftHand, 1);
            anim.SetIKPosition(AvatarIKGoal.LeftHand, leftHand.position);
            anim.SetIKRotation(AvatarIKGoal.LeftHand, leftHand.rotation);
        }
        else
        {
            anim.SetIKPositionWeight(AvatarIKGoal.LeftHand, 0);
            anim.SetIKRotationWeight(AvatarIKGoal.LeftHand, 0);
        }

        if (rightHand != null)
        {
            anim.SetIKPositionWeight(AvatarIKGoal.RightHand, 1);
            anim.SetIKRotationWeight(AvatarIKGoal.RightHand, 1);
            anim.SetIKPosition(AvatarIKGoal.RightHand, rightHand.position);
            anim.SetIKRotation(AvatarIKGoal.RightHand, rightHand.rotation);
        }
        else
        {
            anim.SetIKPositionWeight(AvatarIKGoal.RightHand, 0);
            anim.SetIKRotationWeight(AvatarIKGoal.RightHand, 0);
        }
    }
}
```