Aplicació de desenvolupament de videojocs fàcil per a tothom

Àlex Valls Torres

Directora: Marta Fairén
03-07-2020

Grau en Enginyeria Informàtica
Especialitat de Computació

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Resum

Aquest projecte consisteix en el desenvolupament d'una aplicació que permet crear videojocs senzills a usuaris que no disposen de coneixements tècnics en aquest àmbit. D'aquesta manera s'ha creat un motor gràfic amb unes funcionalitats adaptades a aquest tipus de públic per tal d'oferir una experiència còmoda i instructiva, al mateix temps que les eines proporcionades són suficientment flexibles per poder-se combinar entre elles de formes creatives i arribar a generar creacions més complexes. Així doncs s'ha utilitzat C++ i OpenGL per implementar aquest *game engine*, el qual s'ha programat de la forma més eficient possible i buscant un bon rendiment en l'ús de l'aplicació.

Resumen

Este proyecto consiste en el desarrollo de una aplicación que permite crear videojuegos sencillos a usuarios que no disponen de conocimientos técnicos en este ámbito. De esta forma, se ha creado un motor gráfico con unas funcionalidades adaptadas a este tipo de público, con la finalidad de ofrecer una experiencia cómoda e instructiva, a la vez que las herramientas proporcionadas son suficientemente flexibles para poderse combinar entre ellas de formas creativas y poder generar creaciones más complejas. Así que se ha utilizado C++ y OpenGL para implementar este *game engine*, el cual se ha programado de la forma más eficiente posible y buscando un buen rendimiento en el uso de la aplicación.

Abstract

The objective of this project is to develop an application that allows users with no technical knowledge to create simple video games. On this way, I created a game engine with adapted functionalities for this kind of public, in order to offer a comfortable and instructive experience and, at the same time, providing tools that they are flexible enough to combine themselves in creative ways to generate more complex creations. C++ and OpenGL have been used for programming this game engine, looking for a good performance when the application is used.

Índex

Llista de fotografies **3**

Llista de taules **5**

1. Introducció i contextualització

1.1. Context

Avui dia la indústria del videojoc està creixent molt ràpidament, amb previsions que estimen que el seu valor global l'any 2025 podria ser de 300 bilions de dòlars, com la que fa la revista nord-americana *Variety[1].* A més el sector que més destaca dins de la pròpia indústria és el de jocs per dispositius mòbils, els quals arriben a una massa de gent molt més elevada en comparació amb altres plataformes com consoles o jocs d'ordinador.

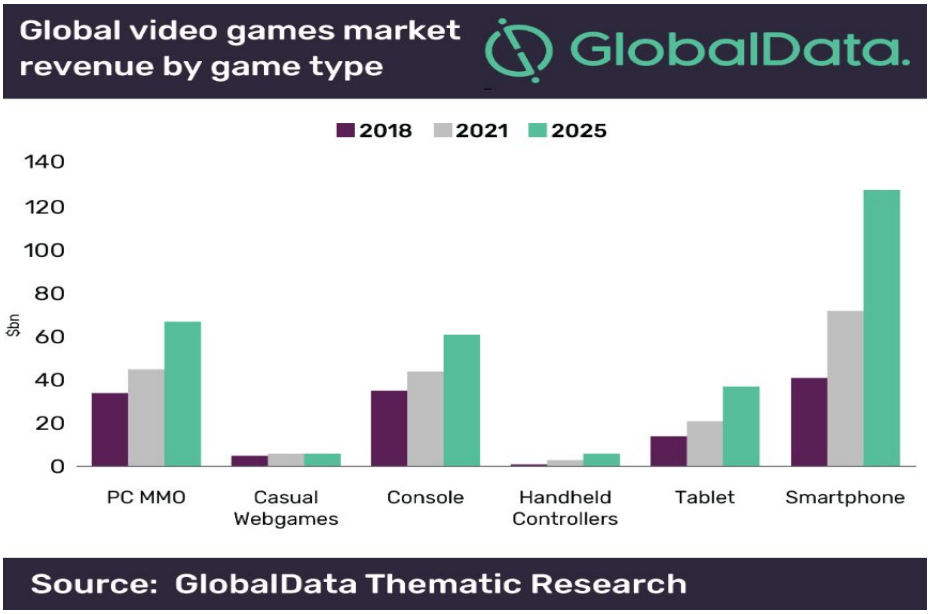

Figura 1. Valor del mercat dels videojocs per plataformes.

Així doncs, de la mateixa manera que creix el consum de videojocs, també creix el nombre de persones que tenen interès en treballar per crear-ne de nous. Això engloba molts tipus de perfils de persona, ja que dins de la creació d'un joc hi ha moltes branques ben diferenciades, com programació, disseny de nivells, música o modelat 3D per exemple, i cadascú es pot dedicar a l'àmbit que més l'interessi. D'aquesta manera per arribar a fer un joc AAA (aquesta és la terminologia per les grans superproduccions dins d'aquesta indústria) es necessiten centenars de persones en cada àrea de desenvolupament treballant durant anys per aconseguir el resultat final que es posa a la venda.

Tot i així també hi ha gent que decideix fer el seu propi joc de forma independent a una empresa, ja sigui en solitari o amb un grup de persones. Els videojocs creats d'aquesta manera reben el nom de jocs "indie" i la producció d'aquest tipus de jocs també està en un clar creixement en els últims anys, sortint al mercat productes amb cada cop més qualitat. Això és un clar indicador de què

l'interès i la motivació per crear videojocs propis va en augment i el conjunt d'eines existents al mercat, com *Unity[2]* o *Unreal Engine[3],* donen l'opció a aquests creadors a desenvolupar les seves idees.

Per tant, aquest Treball de Fi de Grau consistirà en el desenvolupament d'una d'aquestes eines per tal que pugui ser utilitzada de forma senzilla i usable per a una persona sense coneixements tècnics sobre la creació de videojocs. El projecte està inclòs dins de l'especialitat de Computació del Grau Enginyeria Informàtica de la Facultat d'Informàtica de Barcelona.

1.2. Estat de l'art

Tal com s'ha vist en l'apartat anterior, existeix un problema per algunes persones que voldrien crear videojocs i per tant, s'ha procedit a estudiar el mercat actual de *game engines* per veure quines opcions existeixen per aquest perfil d'usuari. Després d'haver trobat molts motors de molts tipus i complexitats diferents per desenvolupar videojocs, es pot concloure en què els 3 programes que són més utilitzats i més adequats pel tipus d'usuari al qual es vol dirigir aquest projecte serien:

Scratch*[4]:* És una eina i un llenguatge de programació gràfic per blocs que pot ser utilitzat per crear demostracions interactives i jocs simples i compartir-los amb altres persones. Creat per *Lifelong Kindergarten* l'any 2003 i amb actualment un total de 20 milions de projectes creats, és una molt bona opció perquè els nens aprenguin els primers conceptes de programació, com els bucles i condicionals per exemple, i puguin explorar la seva creativitat. A més, amb *Scratch* també es poden arribar a generar creacions moderadament complexes, ja que parteix de la idea de donar unes eines força bàsiques a l'usuari però que si se saben combinar correctament, es poden crear resultats curiosos i interessants.

Multimedia Fusion 2*[5]:* És un *game engine,* creat per *Clickteam,* que permet el desenvolupament de jocs 2D i altres aplicacions multimèdia per plataformes mòbils i ordinador. Ofereix un entorn més complex que les altres 2 opcions que s'estan comentant en aquest apartat i això el fa un programa més apte per usuaris adults amb algunes nocions bàsiques de programació i no tant per nens, ja que la seva interfície conté molta informació i és força més fàcil utilitzar-lo si l'usuari ja té una certa familiarització amb el món del desenvolupament de videojocs. Tot i així, té una versió més simple, anomenada *The Games Factory 2*, més utilitzada en camps educatius.

GameSalad Creator*[6]:* Aquest és un altre programa dirigit a persones sense coneixements de programació i educadors, el qual consisteix en un editor visual basat en la lògica del comportament dels objectes. Va ser creat l'any 2010 per *GameSalad Inc* i s'utilitza principalment en àrees educatives per aprendre conceptes de pensament lògic i resolució de problemes i, a més, també és molt freqüent el seu ús

per desenvolupadors de videojocs més experimentats, els quals fan primers prototips dels seus jocs en aquesta plataforma i d'aquesta manera poden comprovar si la seva idea és viable i bona de forma senzilla i per tant, eviten gastar temps innecessari en fer el desenvolupament real d'un joc que no s'hagués completat.

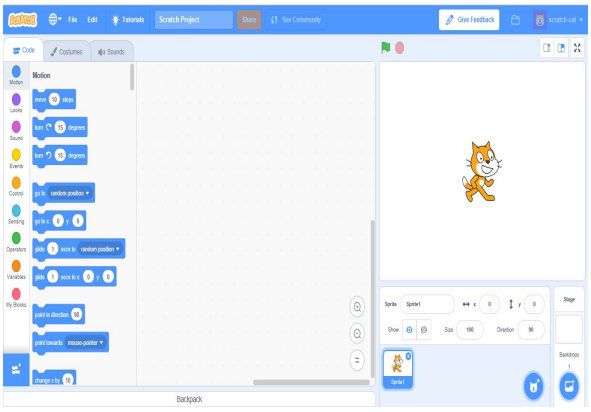

Figura 2. Captura de Scratch.

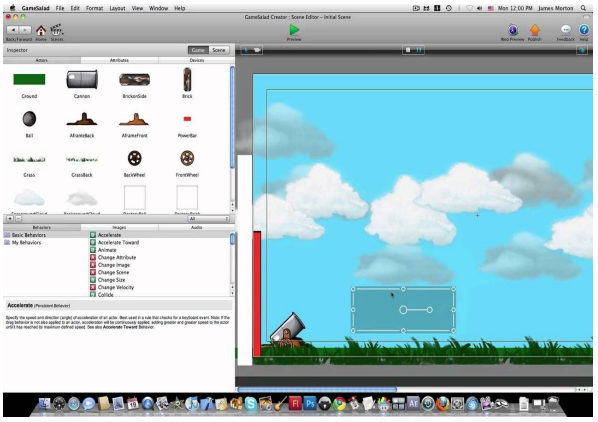

Figura 3. Captura de GameSalad.

Un cop s'han estudiat les possibilitats existents, es pot veure que principalment estan orientades a la creació de jocs 2D i que utilitzen entorns visuals amb eines senzilles per tal que l'usuari pugui familiaritzar-se ràpidament amb el programa, però alhora també les poden utilitzar usuaris més experimentats per generar creacions complexes.

Tot i així s'ha pogut veure que gairebé cap d'aquests programes permet desenvolupar jocs en 3D i per tant, amb el nostre motor s'intentarà cobrir aquesta branca. Així doncs, per aquest projecte es crearà un motor nou que no partirà de cap programa existent, el qual majoritàriament, es crearà amb idees pròpies, exceptuant algunes funcionalitats extretes de les eines mencionades anteriorment, les quals poden ser interessants d'aplicar en el nostre programa.

1.3. Actors implicats

A continuació es detallaran les persones implicades en el projecte, sigui directe o indirectament, tant en la part del desenvolupament com en la part del seu ús en un futur.

Desenvolupador: Com que aquest és un projecte individual només hi haurà un únic desenvolupador, el qual sóc jo i sóc el primer interessat en què es generi un producte satisfactori que compleixi els objectius establerts. Per tant això implica que sóc l'únic responsable de les tasques de planificació, documentació, programació, disseny i testeig.

Directora del projecte: El projecte està dirigit per la professora Marta Fairén, membre del departament de ciències de la computació de la UPC, la qual ajudarà, quan sigui necessari, i guiarà al desenvolupador per tal que el projecte avanci de forma correcta.

Testers: En alguns moments del projecte se sol·licitarà ajuda a algunes persones del meu entorn, les quals no tenen cap tipus de coneixement sobre el desenvolupament de videojocs ni programació en general, per tal de veure si el programa està ben adaptat a aquest estil d'usuari. A partir de les seves opinions es podran modificar o millorar algunes funcionalitats si es creu adient.

Usuari: Les persones interessades en crear el seu propi videojoc però no tenen la possibilitat d'aprendre tots els conceptes necessaris per dur a terme aquesta tasca. A més aquí també s'inclou el sistema educatiu, el qual podria utilitzar aquesta eina en la docència de les escoles per tal de proporcionar una aplicació als alumnes amb la que puguin aprendre conceptes de forma més interactiva i incentivar la seva creativitat.

La comunitat de jugadors: Les persones que juguen a videojocs també es veurien beneficiades indirectament gràcies als resultats obtinguts amb aquest *game engine*, ja que la gent podria jugar als jocs creats, si es dóna el cas en què aquests són publicats.

2. Abast del projecte

2.1. Formulació del problema

Un motor gràfic té moltes característiques que requereixen coneixements informàtics i matemàtics força avançats i això fa que una gran part de la població es vegi obligada a aprendre una gran quantitat de conceptes tècnics per fer un joc, encara que aquest sigui senzill. Per tant ens trobem en la situació en què el món dels videojocs està creixent i es va fent més popular i, a conseqüència d'això, cada cop hi ha més gent que vol crear jocs, però per poder-ho fer necessiten aprendre molts conceptes que en alguns casos no els és possible assolir. Aquesta circumstància pot ser deguda a àmbits econòmics o socials, entre altres, o bé senzillament que no tenen la suficient predisposició per posar-se a estudiar sobre aquest tema.

Fins i tot, si volem concretar més sobre una part de la població, podem analitzar el cas dels nens. Quan som petits és un moment en què els humans som molt creatius i poder crear un videojoc senzill en aquesta edat podria explotar molt més la nostra creativitat, i a més ho faria d'una forma divertida, ja que no deixaria de ser un joc.

Un cop vist això, podem considerar que crear una eina suficientment senzilla i flexible per persones que no tenen suficients coneixements per crear un videojoc, en un d'aquests *game engines* mencionats anteriorment, pot ser una bona idea per donar-los la possibilitat de desenvolupar les seves idees i expandir més el sector del videojoc.

2.2. Objectius i subobjectius

En aquest apartat s'explicaran els objectius principals i secundaris del projecte que intentaran donar solució al problema plantejat en els apartats anteriors.

Primerament, l'objectiu principal que ens proposem és construir un *game engine* que pugui utilitzar una persona que no sap programar ni coneix els conceptes tècnics de la construcció d'un videojoc, per tal de poder desenvolupar-ne un d'una forma senzilla i usable. D'aquesta manera el motor gràfic creat permetrà a l'usuari generar un joc 3D no gaire complex, utilitzant funcionalitats que substitueixen les tasques de programació utilitzades normalment en qualsevol altre motor, com pot ser *Unity* per exemple. Per tant, l'usuari només haurà d'aprendre a utilitzar les operacions que tindrà disponibles i, internament i de forma invisible per ell, es duran a terme els processos d'elaboració de codi.

Per altra banda, el segon objectiu més important és aconseguir una eina que sigui suficientment flexible perquè una persona, segons el seu grau de dedicació, pugui fer un joc més o menys complex. Per tal que això pugui passar, s'han de crear unes funcionalitats prou flexibles i combinables i així permetre al desenvolupador que, amb la seva imaginació, tingui prou llibertat per crear. Així doncs, és necessari pensar bé les diferents opcions que s'ofereixen en el programa per tal que limitin el menys possible la idea de la persona que vol crear el seu joc.

Seguidament, el primer subobjectiu del projecte és que pugui arribar a ser útil de cara al sistema educatiu. Com s'ha comentat en els apartats anteriors, arribar a crear una eina capaç d'estimular la creativitat dels nens i fer-ho d'una forma divertida és un objectiu molt interessant i amb la creació d'aquest motor gràfic es pot arribar a desenvolupar aquesta "joguina". Tot i així cal remarcar que aquest objectiu és menys rellevant que els dos anteriors i no es prioritzarà que el producte final es pugui utilitzar per fins educatius.

Finalment, l'última cosa que es vol aconseguir és crear una eina més per desenvolupar videojocs i expandir més el nombre de videojocs i el volum de la indústria. A més la complexitat dels jocs que es poden arribar a fer amb aquest motor és similar a la dels jocs de mòbil d'avui dia, els quals són els més jugats, com s'ha pogut veure quan s'ha contextualitzat el projecte. Aquest subobjectiu es pot considerar que s'aconsegueix de forma indirecta, ja que depèn dels jocs que creïn els usuaris que utilitzin la nostra aplicació.

2.3. Possibles obstacles i riscos

Per desenvolupar aquest projecte hi ha una sèrie de riscos que s'han de tenir en compte que poden arribar a succeir:

- Problemes de disseny: Es pot donar el cas en què un cop el projecte ha estat dissenyat i la fase de programació està avançada, ens trobem que el disseny previ de les funcionalitats no és del tot correcte o és millorable, i això ens impedeixi aconseguir l'objectiu de crear un sistema prou bo pel que fa a flexibilitat per l'usuari. Per tal d'intentar evitar que passi, és necessari fer un disseny molt bo de quines possibilitats s'han d'oferir i fins a quin punt es poden combinar les unes amb les altres.

- Substitució de les tasques de programació: Pot ser que en alguns casos adaptar parts de codi en una funcionalitat dins la interfície gràfica sigui realment complicat i això faria més difícil que el desenvolupador pugui crear jocs amb la complexitat que ell desitjaria. Això es pot solucionar de la mateixa manera que el problema anterior, pensant bé com crear el sistema abans de començar a programar-lo.

- Canvi de fase entre desenvolupament i joc: En un *game engine* hi ha la possibilitat d'executar el joc en la pròpia interfície de desenvolupament i, abans de l'inici del projecte sembla la part que més dificultats pot ocasionar a l'hora de programar.

- Limitació de temps: La durada del projecte està definida per la data límit de la presentació i s'han de complir els terminis establerts, per tant s'hauran de planificar bé les diferents fases del projecte i anar seguint les dates marcades en la planificació prèvia de la millor manera possible.

3. Metodologia i rigor

3.1. Metodologia de treball

La metodologia amb la qual es durà a terme aquest projecte consistirà en un mètode *agile kanban[7],* la qual permet estructurar les diferents tasques del projecte de forma visual i definir el flux de treball d'una forma precisa i flexible.

El desenvolupament del projecte se segmentarà en diferents tasques i s'organitzaran per tal de seguir un ordre de prioritats i requisits adequat per assolir els objectius, per tant primer caldrà fer les funcionalitats més bàsiques i seguidament anar fent les fases més complexes que no es podien desenvolupar si no s'havien fet les anteriors prèviament. D'aquesta manera aconseguirem completar els objectius marcats amb un resultat base i a partir d'aquí s'anirà perfeccionant la solució en les fases posteriors. A més, per tal de complir els terminis de temps establerts, caldrà planificar unes dates per definir quan una tasca ha d'estar acabada. Tot i així aquesta planificació prèvia sempre estarà subjecte a petites modificacions segons l'estat de desenvolupament del motor, cosa que la metodologia triada permet gràcies a la seva flexibilitat per gestionar les tasques.

3.2. Eines

Per desenvolupar aquest *game engine* farem ús de *Qt5[8]* per crear la interfície d'usuari i *OpenGL[9]* per la part de programació gràfica. Pel codi font s'utilitzarà C++ amb el suport del programa *Atom[10]* com editor de text. Per altra banda, es crearà un repositori a *Github[11]* per tal de gestionar el control de versions i tenir el codi pujat en un lloc segur. A més per gestionar la planificació del projecte s'utilitzarà *Trello[12]* per fer el seguiment de les tasques del projecte. Finalment també cal dir que s'aprofitaran conceptes i algunes parts de codi utilitzats en l'assignatura d'IDI, en la qual es podia arribar a crear una petita aplicació gràfica amb *OpenGL* i *Qt5*.

Figura 4. Logotips d'algunes eines utilitzades.

3.3. Validació dels resultats

Per tal d'anar validant durant el desenvolupament que el projecte funciona correctament i va pel bon camí utilitzarem dos tipus de proves. Primerament, per tal de comprovar que l'estat tècnic de l'aplicació s'ha programat de forma correcta, es faran proves pràctiques dins del motor utilitzant les noves funcionalitats per tal de veure com funcionen. Si veiem que tot va com ha d'anar passarem a programar la següent tasca seguint la planificació, en cas contrari es tractarà de trobar els errors i solucionar-los de la forma més eficient possible. Per altra banda, també caldrà veure si es va pel bon camí per assolir l'objectiu de què l'aplicació sigui usable per una persona sense coneixements tècnics, per tant es faran diferents experiments amb gent del meu entorn que tingui aquest perfil i així comprovar si les funcionalitats existents en aquell moment estan ben adaptades.

4. Planificació temporal

4.1. Descripció de les tasques

Primerament aquest apartat consistirà en definir les diferents tasques que formen aquest TFG, les quals deriven dels objectius marcats en la descripció de l'abast del projecte, i es justificaran els seus costos temporals i dependències entre elles. A més s'agruparan segons la seva temàtica per tal de classificar-les més clarament.

La data d'inici del projecte és el dia 26/01/2020, ja que es tenen en compte les reunions amb la directora per definir el projecte, i finalitzarà el dia de la presentació oral, la qual serà al torn del dia 3 de juny, per tant tindrà una duració aproximada de 5 mesos.

4.1.1. Definició de les tasques

En total aquest projecte constarà de 20 tasques que comportaran 553h de feina i estan definides de la següent manera:

Gestió de Projecte

En aquest grup hi consten les tasques dedicades a l'estudi del projecte i la gestió d'aquest per tal que tingui un correcte desenvolupament amb la quantitat de feina correcte.

- **Contextualització i Abast:** Es defineixen els objectius del projecte, es contextualitza, es justifica el perquè es desenvolupa i s'especifiquen quins mitjans s'utilitzaran. En aquesta tasca s'invertiran 20 hores de feina i depèn de cap altre, ja que és el primer estudi sobre el projecte que es fa.

- **Planificació Temporal:** Es defineixen les diferents fases del projecte, el cost temporal que tindran i els recursos que s'utilitzaran. En aquesta tasca s'invertiran 9 hores de feina i es fa després d'haver estudiat l'abast.

- **Pressupost i Sostenibilitat:** Es defineixen els costos econòmics aproximats del projecte i l'informe de sostenibilitat. En aquesta tasca s'invertiran 20 hores de feina i depèn de l'estudi de l'abast i de la planificació temporal.

- **Integració del document final:** A partir de les revisions de les 3 tasques anteriors, es fan les millores necessàries i s'ajunten per tal d'integrar-ho a la memòria final.

- **Reunions:** Aquesta tasca inclou les reunions i contactes amb la directora del projecte. Durant el projecte es faran diverses reunions, entre elles la reunió de seguiment, per tal de veure que el desenvolupament progressa adequadament. En aquesta tasca s'invertiran 10 hores de feina.

Treball previ

Aquí hi podem trobar les tasques que consisteixen en l'aprenentatge i estudi de les eines que s'utilitzaran i dels conceptes que s'hauran d'aplicar.

- **Estudi de les eines disponibles:** S'exploren les diferents opcions d'eines que existeixen per desenvolupar el nostre projecte i es trien les que més s'adaptin a les nostres necessitats per tal d'assolir els objectius de la forma més òptima possible. S'invertiran 27h després d'haver fet les tasques de l'apartat anterior.

- **Estudi de conceptes:** Tasca que consisteix en buscar i aprendre els conceptes necessaris per desenvolupar el projecte. En aquesta tasca s'invertiran 39 hores de feina i no depèn de cap altre.

- **Familiarització amb el programa:** Un cop s'han estudiat i escollit les eines amb les quals treballarem, cal un procés d'adaptació a elles. En aquesta tasca s'invertiran 25 hores de feina.

Disseny i implementació

En aquest grup s'inclouen totes les tasques relacionades amb la implementació del *game engine*.

- **Disseny de les funcionalitats:** Quan ja ens hem familiaritzat prou amb les eines de treball comencem a dissenyar les funcionalitats de l'aplicació i la seva estructura. Aquesta tasca és molt important per tal d'aconseguir un *game engine* flexible i fàcil d'utilitzar. En aquesta tasca s'invertiran 35 hores de feina.

- **Creació del Viewer:** El primer que caldrà programar després d'haver dissenyat el sistema és la part de l'aplicació que mostra les escenes 3D gràficament i partir l'aplicació d'aquest punt. En aquesta tasca s'invertiran 33 hores de feina.

- **Creació de la interfície gràfica:** Aquí es construirà el conjunt de *widgets* que estaran al voltant de la finestra on es mostra l'escena, els quals tindran la funció de modificar el que es mostra en el món 3D. En aquesta tasca s'invertiran 33 hores de feina i depèn de la creació del *Viewer* inicial.

- **Gestió dels *GameObjects*:** Es programarà la creació dels objectes de l'escena i les modificacions de les seves propietats. En aquesta tasca s'invertiran 36 hores de feina i depèn de les dues tasques anteriors.

- **Interacció de la càmera:** Programació del moviment de la càmera per l'escena i la interacció de l'usuari amb ella. En aquesta tasca s'invertiran 24 hores de feina i depèn de les tasques anteriors.

- **Canvi de mode desenvolupador/joc:** Programar la funcionalitat que permeti a l'usuari executar el seu joc en temps real dins el motor. En aquesta tasca s'invertiran 34 hores de feina i depèn de les tasques anteriors.

- **Moviments dels *GameObjects*:** Programar que durant el joc l'usuari pugui moure els objectes. En aquesta tasca s'invertiran 32 hores de feina i depèn de les tasques anteriors.

- **Interaccions entre *GameObjects*:** Programació dels comportaments entre objectes durant el joc. En aquesta tasca s'invertiran 40 hores de feina i depèn de les tasques anteriors.

Test del programa

Aquí estan incloses les tasques que serviran per provar el bon funcionament de l'aplicació.

- **Experiment real:** Consistirà en fer provar el motor a una persona del meu entorn que no tingui coneixements de creació de videojocs per veure fins a quin punt s'està assolint l'objectiu de fer un *game engine* per gent amb aquest perfil. En aquesta tasca s'invertiran 10 hores de feina i depèn de les tasques d'implementació.

- **Creació d'un joc de prova:** Crearé un joc amb el motor al final de la implementació per observar el funcionament de l'eina. En aquesta tasca s'invertiran 40 hores de feina i depèn de les tasques d'implementació.

Documentació final

Les tasques d'aquest grup són les següents:

- **Redacció de la memòria:** Completar la memòria a partir del que s'ha fet en les tasques de gestió de projecte. En aquesta tasca s'invertiran 65 hores de feina.

- **Presentació oral:** Preparació de la presentació del projecte davant del tribunal. En aquesta tasca s'invertiran 12 hores de feina i és necessari haver acabat tota la resta de tasques en aquell moment.

4.1.2. Taula resum de les tasques

Codi	Tasca	Temps	Dependència
T1	**Gestió de Projecte**	**68h**	
T1.1	Contextualització i Abast	20h	
T1.2	Planificació Temporal	9h	T1.1
T1.3	Pressupost i Sostenibilitat	20h	T1.1,T1.2
T1.4	Integració del document final	9h	T1.1,T1.2,T1.3
T1.5	Reunions	10h	
T2	**Treball previ**	**91h**	
T2.1	Estudi de les eines disponibles	27h	
T2.2	Estudi de conceptes	39h	
T2.3	Familiarització amb el programa	25h	T2.1
T3	**Disseny i implementació**	**267h**	
T3.1	Disseny de les funcionalitats	35h	T2.3
T3.2	Creació del Viewer	33h	T2.3
T3.3	Creació de la interfície gràfica	33h	T3.1, T3.2
T3.4	Gestió dels GameObjects	36h	T3.3
T3.5	Interacció de la càmera	24h	T3.4
T3.6	Canvi de mode desenv./joc	34h	T3.5
T3.7	Moviments dels GameObjects	32h	T3.6
T3.8	Interaccions entre GameObjects	40h	T3.7
T4	**Test del programa**	**50h**	
T4.1	Experiment real	10h	T3.8
T4.2	Creació d'un joc de prova	40h	T3.8
T5	**Documentació final**	**77h**	
T5.1	Redacció de la memòria	65h	T1.4
T5.2	Presentació oral	12h	T5.1
	Temps total	553h	

Taula 1. Taula resum de les tasques.

4.2. Diagrama de Gantt

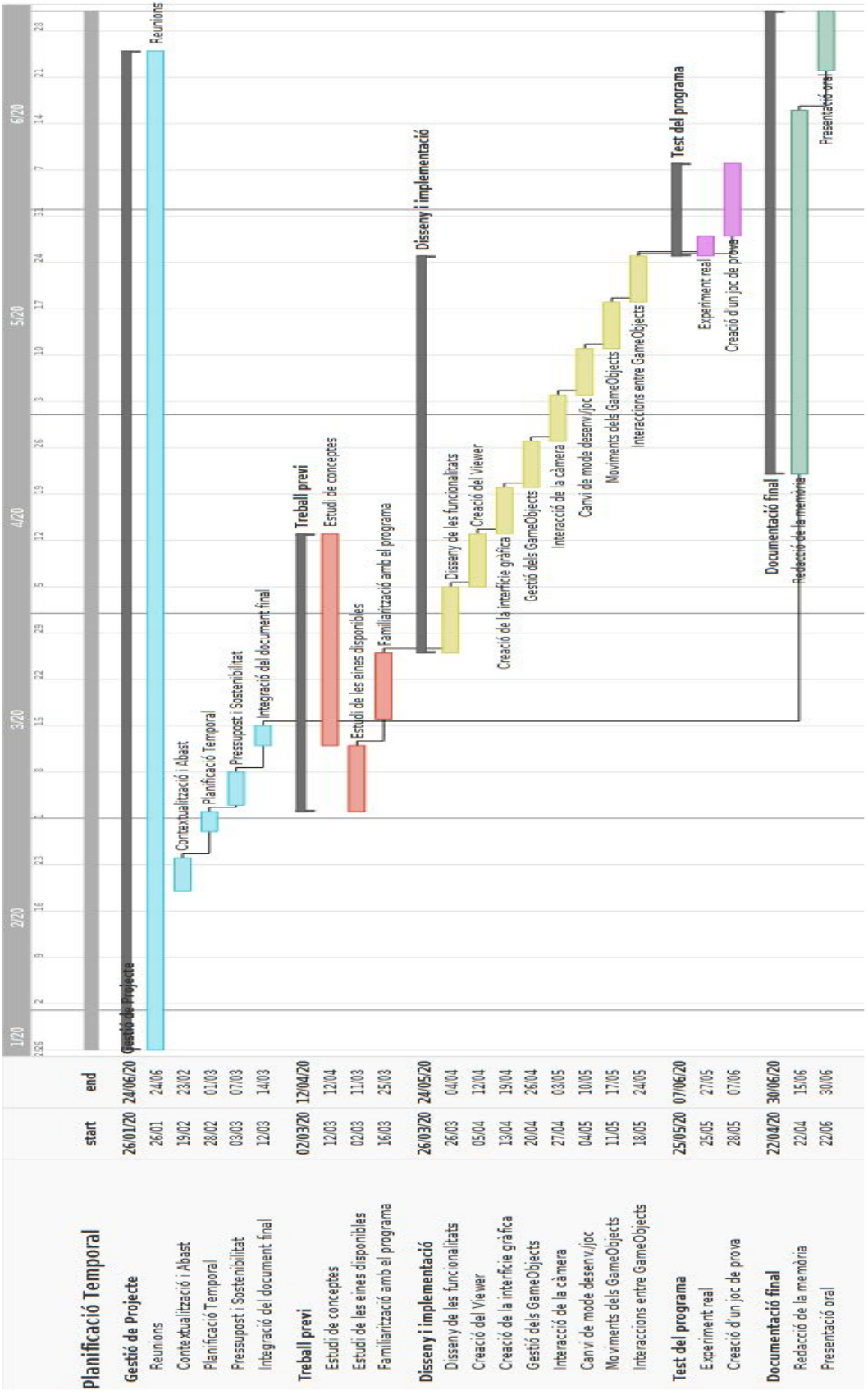

Figura 5. Diagrama de Gantt.

4.3. Gestió del risc

A continuació es passarà a analitzar les possibles alternatives i conseqüències dels riscos plantejats en l'apartat sobre l'abast del projecte. Cal dir que cap dels riscos necessitaria recursos addicionals per tal de solucionar-los, ja que la majoria d'ells consisteixen en la implementació del programa.

Problemes de disseny

Si ens trobem amb errors de disseny un cop hàgim començat a implementar l'aplicació s'haurà d'afegir una tasca de redisseny per solucionar el problema que hem trobat i tot seguit adaptar, si és necessari, les funcionalitats ja programades al nou disseny. Així doncs aquest error ens costaria bastantes hores de feina addicional, les quals serien més o menys segons la magnitud del problema, per tant no es pot fer una estimació exacte del temps extra d'aquesta nova tasca.

Substitució de les tasques de programació

En el cas que una funcionalitat que es vulgui implementar sigui més complicada de programar del que estava previst caldrà ampliar el seu temps de dedicació i per tant endarrerir el començament de les tasques que en depenguin. Per tant caldrà redissenyar-la o bé solucionar el problema en el codi i això, com en el risc anterior, ocasionarà unes hores de feina extra que actualment no es poden saber amb exactitud.

Canvi de fase entre desenvolupament i joc

Aquest risc és el mateix cas que l'anterior, ja que és una tasca d'implementació en la qual poden haver-hi errors i per tant provocarà costos temporals extres. Així doncs, per tal d'evitar aquest problema i els anteriors és necessari dissenyar bé el sistema des del començament.

Limitació de temps

Aquest últim problema apareix a causa de les dates que limiten el projecte, tot i així, si es segueix la planificació feta, no hi hauria d'haver cap problema en aquest aspecte. Per altra banda si un dels riscos comentats anteriorment provoca uns costos temporals extres elevats, la limitació de temps ens obligarà a dedicar més hores diàries a les tasques per tal d'arribar correctament als terminis marcats. Per tal de prevenir que això sigui un problema greu, en la planificació s'han deixat, en la mesura del possible, uns dies lliures al mes de juny per tenir una mica de marge d'error.

4.4. Modificacions respecte la planificació inicial

En aquest apartat s'explicaran els canvis que hi ha hagut en el desenvolupament del projecte respecte al que es va planificar a l'inici. Primerament un dels canvis principals és que l'escriptura de la memòria s'ha anat fent durant tot el desenvolupament i no només en la part final com es va plantejar al principi, ja que d'aquesta manera es podien anar redactant el funcionament de les diferents funcionalitats a mesura que s'anaven implementant i així fer una descripció més recent de cadascuna. Per tant aquesta tasca ha ocupat el mateix temps que es va estimar però s'ha repartit durant les fases d'implementació.

Per altra banda amb la tasca d'experimentar amb gent del meu entorn ha succeït el mateix, ja que s'han anat fent diferents proves al llarg del projecte. Aquestes proves han estat 3 i més concretament han estat després de les següents tasques: *Interacció de la càmera (T3.5), Canvi de mode desenv./joc (T3.6) i Interaccions entre GameObjects (T3.8).* D'aquesta manera un cop es va arribar a aquests 3 punts del desenvolupament, es va poder testejar si les funcionalitats implementades en aquell moment funcionaven de forma correcta i si estaven ben adaptades pel tipus d'usuari al qual va dirigit aquest *game engine.*

Altrament, les tasques assignades a la fase de treball previ que havien de servir per a l'estudi van durar menys temps de l'esperat, cosa que va permetre que la fase d'implementació s'avancés unes setmanes. Gràcies a aquest temps guanyat es va acabar abans la fase d'implementació de les tasques previstes i es va decidir aprofitar aquest temps a programar i dissenyar el sistema d'àudio del programa i a perfeccionar les funcionalitats que ja estaven fetes per tal que el motor fos més complet.

5. Gestió econòmica

En aquest apartat es definiran els costos econòmics que comporta aquest projecte, agrupats segons la seva tipologia, i es justificaran en els següents apartats els motius de cada despesa.

5.1. Costos dels recursos humans

Primerament es tractaran els costos destinats a recursos humans, és a dir, la suma de diners invertida en els sous dels treballadors que hagin participat en cada tasca. Per tant primer es marcaran els sous dels diferents rols dels treballadors i després es calcularà el cost de cada tasca segons quins rols hi han estat implicats.

Així doncs, per aquest projecte es necessitarà un cap de projecte per tal de gestionar-lo i dirigir-lo, un programador que implementi el codi del *game engine*, un dissenyador de software que pensi i especifiqui com han de ser les funcionalitats del sistema i finalment un *tester* que faci les proves finals per comprovar si els resultats compleixen els objectius marcats. A continuació es pot veure el sou per hora mitjà a Espanya d'aquests rols amb el 30% de seguretat social inclòs, segons la web especialitzada *indeed.es [16]:*

Rol	€/hora
Cap de projecte	18 €
Programador	18 €
Dissenyador de software	16 €
Tester	13 €

Taula 2. Sou per hora dels rols del projecte.

Seguidament es passarà a desglosar el cost de cada tasca vista en el Gantt segons els rols dels treballadors que la duen a terme, per tant per calcular el cost de cada tasca només caldrà multiplicar el sou per hora dels treballadors implicats pel nombre d'hores que hauran de dedicar. En el cas que més d'un rol es dediqui a una mateixa tasca, el nombre d'hores d'aquesta es dividirà proporcionalment al nombre de treballadors per tal que tots li dediquin el mateix percentatge de temps, és a dir que si en una tasca que dura 20h, per exemple, hi treballen 2 rols diferents, cadascun d'ells hi dedicarà 10h.

Codi	Tasca	Hores	Rols	Cost(€)
T1	**Gestió de Projecte**	**68h**		**1224**
T1.1	Contextualització i Abast	20h	Cap projecte	360
T1.2	Planificació Temporal	9h	Cap projecte	162
T1.3	Pressupost i Sostenibilitat	20h	Cap projecte	360
T1.4	Integració del document final	9h	Cap projecte	162
T1.5	Reunions	10h	Cap projecte	180
T2	**Treball previ**	**91h**		**1574**
T2.1	Estudi de les eines disponibles	27h	Cap projecte	486
T2.2	Estudi de conceptes	19,5h 19,5h	Dissenyador Programador	663
T2.3	Familiarització amb el programa	12,5h 12,5h	Dissenyador Programador	425
T3	**Disseny i implementació**	**267h**		**4736**
T3.1	Disseny de les funcionalitats	35h	Dissenyador	560
T3.2	Creació del Viewer	33h	Programador	594
T3.3	Creació de la interfície gràfica	33h	Programador	594
T3.4	Gestió dels GameObjects	36h	Programador	648
T3.5	Interacció de la càmera	24h	Programador	432
T3.6	Canvi de mode desenv./joc	34h	Programador	612
T3.7	Moviments dels GameObjects	32h	Programador	576
T3.8	Interaccions entre GameObjects	40h	Programador	720
T4	**Test del programa**	**50h**		**650**
T4.1	Experiment real	10h	Tester	130
T4.2	Creació d'un joc de prova	40h	Tester	520
T5	**Documentació final**	**77h**		**1386**
T5.1	Redacció de la memòria	65h	Cap projecte	1170
T5.2	Presentació oral	12h	Cap projecte	216
	Cost total	553h		**9570 €**

Taula 3. Desglossament dels costos en sous de cada tasca.

5.2. Costos genèrics

A continuació passarem a tractar els costos genèrics, els quals fan referència a les despeses que comporten el hardware i el software utilitzats i el lloguer d'un espai de treball.

5.2.1. Hardware i software

Primerament, els costos de hardware només estan compostos per l'ordinador portàtil que s'utilitzarà per fer tot el projecte. Per tant en la següent taula es pot veure com s'extreu el seu cost utilitzant la "*line depreciation formula*", la qual té la següent forma:

$$\frac{Cost\ del\ producte - Valor\ del\ producte\ acabada\ la\ vida\ útil}{Anys\ de\ vida\ útil}$$

Producte	Preu	Vida útil	Valor amb 5 anys	Cost
ASUS ROG GL753V	1200 €	5 anys	300€	**180 €**

Taula 4. Cost del hardware.

Per altra banda el cost de software és 0 €, ja que tots els programes utilitzats són software lliure i gratuït, com per exemple Github, Qt5, Trello o Atom.

5.2.2. Espai de treball

Per realitzar el projecte s'ha decidit llogar un espai de coworking a Barcelona, d'aquesta manera el cost per l'espai es redueix a una tassa mensual, la qual depèn del local escollit, i alhora evitem costos per electricitat, aigua... Així doncs, a partir de les dades extretes de la web *Coworking Spain [13]*, el cost d'un d'aquests espais és aproximadament de 200 € al mes, per tant utilitzar-lo durant quatre mesos implicarà que el cost final serà de **800 €**.

5.3. Costos addicionals

5.3.1. Contingència

Per tal de tenir suficients diners per cobrir els possibles imprevistos del projecte es reservarà un petit fons de contingència, el qual es fixarà en un 15% dels

costos totals del projecte. Per tant si apliquem un 15% a tots els costos avaluats fins ara obtenim el següent resultat:

Tipus	Cost	Cost de contingència (15%)
Recursos humans	9570 €	1435.5 €
Hardware i software	180 €	27 €
Espai de treball	800 €	120 €
Total	**10550 €**	**1582.5 €**

Taula 5. Costos de contingència.

5.3.2. Imprevistos

Finalment, també s'han de tenir en compte els riscos que s'han considerat en els lliuraments anteriors, ja que aquestes afectacions generarien un cost econòmic addicional. Els problemes que es van plantejar comportaven, bàsicament, temps de desenvolupament extra, per tant només cal tenir en compte el cost d'aquestes hores per saber l'impacte econòmic dels imprevistos del projecte. Així doncs si es considera que els tres riscos que es van marcar afegirien cadascun unes 15h de feina aproximadament, el càlcul del cost addicional seria el següent:

Tipus		Cost
Recursos humans	15h*16 € + 30h*18 €	780 €
Espai de treball	2 setmanes*50 €/setmana	100 €
Total		**880 €**

Taula 6. Costos en imprevistos.

5.4. Cost total

Recursos Humans	Hardware i software	Espai de treball	Contingència	Imprevistos	Total
9570 €	180 €	800 €	1582.55 €	880 €	**13012.55 €**

Taula 7. Cost total del projecte.

5.5. Control de gestió

Per tal de controlar que el pressupost que s'ha fet es va seguint correctament durant el projecte, es farà un anàlisi cada vegada que s'acabi una tasca del Gantt per tal de veure la diferència entre el cost estimat i el cost real segons el nombre d'hores que s'han dedicat a completar-la. D'aquesta manera podrem fer un seguiment precís sobre cada tasca i s'anirà guardant per cada tasca les possibles desviacions i així tindrem una visió més global de les despeses extres durant el desenvolupament. Per tant el càlcul per saber la desviació econòmica d'una tasca seria:

Preu desviació (€) = (Preu estimat - Preu real) * Hores dedicades

Si ens trobem en el cas que hem gastat més diners dels previstos seria el moment d'utilitzar el fons de contingència i intentar millorar l'eficiència de les tasques posteriors per compensar les despeses extres, reduint el nombre hores de feina. Per altra banda si gastem menys del previst no caldria fer cap modificació en la planificació inicial i es guanyaria una part de diners que es podria utilitzar per cobrir possibles desviacions negatives de tasques futures.

6. Sostenibilitat i compromís social

6.1. Dimensió Econòmica

+ Respecte el Projecte posat en producció (PPP): Reflexió sobre el cost que has estimat per la realització del projecte?

Els costos estimats pel projecte són els mínims possibles perquè el desenvolupament de l'aplicació sigui correcte i el resultat final sigui de bona qualitat. A més considero que s'ha fet una bona previsió i hi ha una seguretat econòmica si sorgeixen imprevistos.

+ Respecte la Vida Útil: Com es resolen actualment els aspectes de costos del problema que vols abordar (estat de l'art)?

Actualment una manera per reduir costos en el món dels videojocs és treure el producte en accés anticipat per tal de generar beneficis durant una fase força inicial del projecte.

+ En què millorarà econòmicament (costos...) la teva solució respecte a les existents?

La meva solució utilitza un espai de coworking per treballar i un sol ordinador, per tant això abarateix força el cost total del projecte respecte altres solucions existents.

6.2. Dimensió Ambiental

+ Respecte el Projecte posat en producció (PPP): ¿Has estimat l'impacte ambiental que tindrà la realització del projecte?

L'únic impacte ambiental que tindrà el meu projecte és l'ús elèctric del meu ordinador, ja que tot el projecte consisteix en el desenvolupament del *game engine*.

+ Respecte el Projecte posat en producció (PPP):¿T'has plantejat minimitzar-ne l'impacte, per exemple, reutilitzant recursos?

L'única manera de minimitzar l'impacte és reduint el nombre d'hores de treball, però això és un escenari poc probable.

+ Respecte la Vida Útil: Cóm es resol actualment el problema que vols abordar (estat de l'art)?, i. ¿En què millorarà ambientalment la teva solució respecte a les existents?

La meva solució només té un impacte elèctric en el medi ambient, per tant és més òptima en aquest aspecte respecte a altres solucions que treballen amb més nombre d'ordinadors i que no treballen en un espai de coworking.

6.3. Dimensió Social

+ Respecte el Projecte posat en producció (PPP): Què creus que t'aportarà a nivell personal la realització d'aquest projecte?

Aquest projecte em permetrà aprendre a construir un motor per desenvolupar videojocs i els conceptes que aprendré em seran molt útil de cara al meu futur professional en la indústria del videojoc.

+ Respecte la Vida Útil: Cóm es resol actualment el problema que vols abordar (estat de l'art)? i. ¿En què millorarà socialment (qualitat de vida) la teva solució respecte les existents?
Com s'ha anat dient durant les entregues anteriors aquest projecte busca donar una solució que pràcticament no existeix en el mercat actual, així doncs amb el programa que es generarà moltes més persones podran crear els seus propis videojocs.

+ Respecte la Vida Útil: Existeix una necessitat real del projecte?

Realment el projecte no és una necessitat vital o una solució per a gran part de la població, però sí que dóna una eina a un sector de gent que no té l'opció de crear un joc de forma senzilla avui dia.

6.4. Informe de sostenibilitat

El fet de fer l'enquesta per EDINSOST m'ha servit per comprovar la meva actitud i coneixements respecte als temes de sostenibilitat en les TIC i he pogut observar que tinc un nivell mitjà en aquesta àrea. A més m'he adonat que tots els projectes tenen unes conseqüències en 3 dimensions diferents: dimensió econòmica, dimensió social i dimensió mediambiental.

Tot i així, haig de dir que encara no he participat mai en cap projecte d'aquest estil i per tant el meu compromís en la sostenibilitat d'un projecte és teòrica, però considero que estaria força implicat en no deixar de banda aquests factors. Per tant, aquesta enquesta ha fet que en el desenvolupament del meu projecte tingui molt més en compte els factors de sostenibilitat.

Finalment també cal dir que les preguntes m'han estat útils per aprendre nous conceptes en aquest àmbit, ja que no tenia un coneixement profund de la matèria i algunes terminologies m'han estat difícils d'entendre.

7. Conceptes teòrics i eines

En aquest apartat es procediran a explicar els conceptes teòrics més tècnics que són necessaris per entendre com s'ha implementat el projecte i en els quals s'ha dedicat una part del temps del projecte a estudiar-los.

7.1. Game Engine

Un *game engine[14]* és un software que conté un conjunt de rutines de programació i permet dissenyar i crear videojocs de forma ràpida i eficient. Ara es procedirà a explicar més detalladament els seus principals components per tal d'ensenyar el seu funcionament i les característiques que pot arribar a oferir algun dels motors més potents del mercat.

7.1.1. Motor gràfic

Aquest element és el que permet renderitzar gràfics en 2D o 3D en temps real i la part central del sistema, ja que és l'encarregada de generar i mostrar tot el que es programi en les altres funcionalitats. Normalment les aplicacions d'aquest tipus permeten renderitzar les escenes en 2 modes diferents: mode desenvolupador, en el qual l'usuari crea i gestiona els diferents elements del món que està generant i mode joc, en el qual es pot iniciar una simulació en temps real del joc que s'està creant i d'aquesta manera es pot comprovar, de forma gairebé instantània, si s'està desenvolupant de forma correcta.

Per altra banda, un altre element molt important del motor gràfic és el processament de la il·luminació i d'efectes visuals, els quals tenen una importància molt rellevant en aconseguir que el joc resultant tingui l'aspecte desitjat. Aquest sistema es pot utilitzar de forma senzilla i crear un sol focus de llum que afecta a tota l'escena de forma uniforme o es pot arribar a utilitzar per fer ús de tècniques com el *raytracing*[15], el qual consisteix en traçar rajos que reboten pels diferents objectes de l'escena per tal d'intentar simular el funcionament de la retina humana. En la següent figura es pot veure a l'esquerra una escena del joc *Grand Theft Auto V* i a la dreta la mateixa imatge amb *raytracing* aplicat.

Figura 6. Comparativa del *raytracing*.

7.1.2. Sistema d'animacions

Aquesta eina serveix per donar la possibilitat als dissenyadors de moure els objectes de l'escena de forma orgànica i fluida, així doncs, es pot aplicar tant per animar els moviments complexos dels personatges principals com per simular el comportament senzill de les fulles d'un arbre per exemple. A més, aquest sistema és utilitzat per generar les escenes prerenderitzades del joc, les quals no es generen durant l'execució d'aquest. En molts casos les animacions tenen una relació directa amb el sistema de físiques, el qual els hi dóna un aspecte molt més realista, com pot ser el cas en què un cos humà caigui d'algun lloc i interaccioni correctament amb el terreny que l'envolta.

7.1.3. Motor de físiques

El motor de físiques té la funció d'emular el comportament de les lleis de la física en el joc en temps real. Això inclou dotar els objectes d'interaccions i moviments realistes a l'hora de col·lisionar entre ells i aplicar forces, com la força de fricció o gravetat per exemple. Aquest motor pot arribar a ser molt flexible si conté forces opcions i pot permetre al desenvolupador crear un sistema de físiques que s'adapti, en gran mesura, a la idea del joc que vol dissenyar, ja que depenent del joc es necessita un motor de físiques més o menys avançat. Com es pot comprendre un joc de naus espacials no necessita un sistema tan complex com el del joc *Control[17]*, el qual basa gran part de la seva jugabilitat en les físiques dels objectes.

Entrant més en detall, els dos mètodes més senzills per detectar col·lisions consisteixen en envoltar els objectes per una esfera contenidora o bé per una caixa rectangular i si qualsevol altre objecte les penetra, es considera que hi ha hagut una col·lisió. A partir de la zona on ha succeït l'impacte i les forces dels objectes en aquell instant, el sistema de físiques intenta simular, de la forma més realista possible, les noves forces i direccions, tot i així existeix la limitació de la precisió dels nombres que representen aquestes dades, ja que si aquesta és prou baixa pot provocar errors que afecten el resultat final.

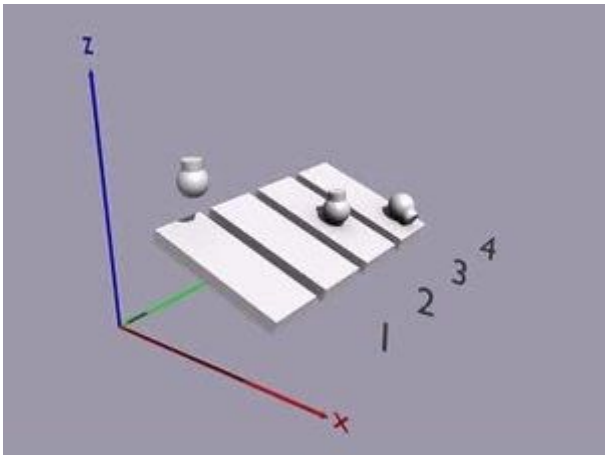

Figura 7. Representació del sistema de físiques.

En la Figura 7, es poden veure 4 exemples clars de com es pot aplicar un sistema de físiques. En el cas número 1, no hi ha físiques, cosa que implica que l'objecte es mantingui flotant. Seguidament, en el cas número 2 s'ha afegit únicament el sistema de forces, per tant a l'objecte se li ha aplicat la força de gravetat i ha marxat del camp de visió. En el tercer exemple s'han implementat les deteccions de col·lisions de forma que l'objecte es queda dret sobre la plataforma. Finalment quan s'afegeixen les dinàmiques d'un cos rígid, aquest ja no queda dret, sinó que cau de forma més realista.

7.1.4. Sistema d'àudio

El sistema d'àudio serveix per fer que el joc generi una experiència més immersiva al jugador. Aquesta funcionalitat pot ajudar a entendre el que està passant en el joc si en el moment en què succeeix un event sona un efecte de so i pot enriquir molt més l'ambientació de l'escena, fins i tot, segons la idea del joc, l'àudio pot ser la peça principal perquè el joc sigui divertit. Jocs com el *Geometry Dash[18]* demostren com una música i efectes de so ben fets poden pujar notablement el nivell d'un videojoc. A més actualment s'estan implementant sistemes d'àudio 3D, els quals generen una percepció dels sons molt més avançada i que en jocs d'acció o de por permeten al jugador localitzar de forma més precisa d'on prové el que escolta.

7.1.5. Lògica de programació

La lògica de programació és tot el sistema que engloba la implementació de diferents algoritmes i regula el funcionament intern del videojoc, així doncs l'usuari s'encarrega de generar *scripts* que gestionen els comportaments dels objectes durant el joc. Aquesta és la secció que més depèn dels coneixements tècnics del desenvolupador, ja que com més coneixements de programació tingui, més complexes seran els jocs que podrà crear.

A més, en aquesta part és on es troben els sistemes d'intel·ligència artificial. Habitualment, en aquest apartat s'utilitzen arbres de decisió, els quals ramifiquen les accions i preses de decisions, i també intervenen eines de percepció de l'entorn per tal que el sistema d'intel·ligència artificial rebi les dades necessàries de l'estat actual del joc. Fins i tot els motors més avançats del mercat proporcionen eines per aplicar tècniques de *Machine Learning* als seus jocs, els quals proporcionen un grau de precisió i qualitat més elevat.

Algunes altres funcionalitats menys importants i més comunes en aquest tipus de programes serien el sistema de gestió de memòria, eines de *Debugging* i control dels *inputs* i *outputs* vinculats a l'aplicació.

Actualment els motors més utilitzats són Unity i Unreal Engine, els quals ofereixen un gran espectre de possibilitats a l'hora de crear jocs. Es poden crear des de videojocs 2D molt senzills per plataformes mòbils fins a jocs 3D molt més espectaculars per consoles o ordinador. Per altra banda també existeixen altres motors no tan genèrics que han estat creats només per fer un sol tipus de joc o que s'enfoquen en un tipus de públic concret.

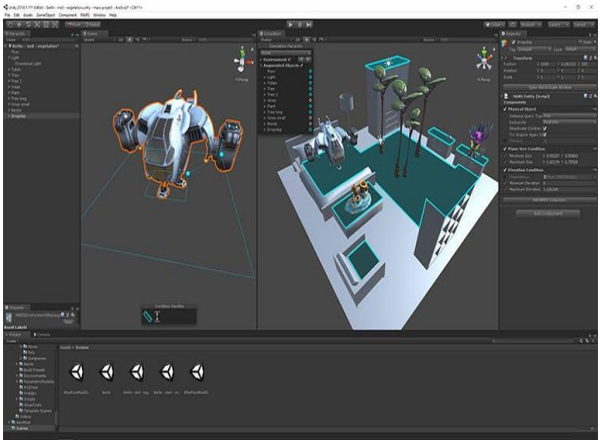

Figura 8. Captura de Unity 5.

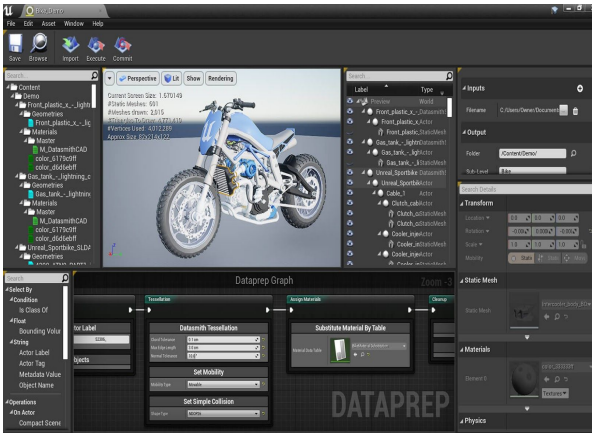

Figura 9. Captura de Unreal Engine 4.

7.2. OpenGL

Aquest apartat es dedicarà a explicar quines eines ofereix OpenGL i com s'han aplicat a l'hora d'implementar aquest motor. De forma general, aquesta aplicació gràfica permet renderitzar gràfics 2D i 3D i per fer-ho, fa un conjunt de processos que es poden observar a la Figura 10, en els quals es processen les dades de les entitats de les escenes per tal que es pintin de la manera correcta. Aquest pipeline gràfic té certes parts programables que permeten a un desenvolupador de gràfics modificar com vol que el seu món virtual es mostri, escollint efectes d'il·luminació, si aplicar o no ombres o reflexions per exemple.

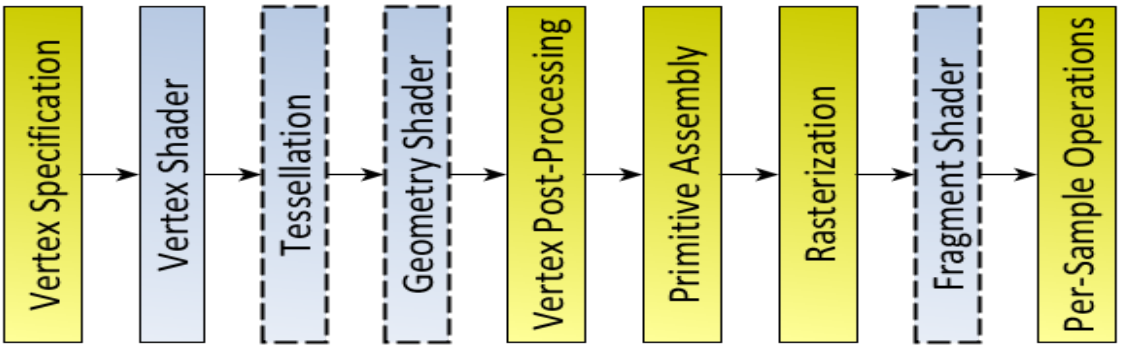

Figura 10. Pipeline Gràfic d'OpenGL.

Tot i així, en aquest projecte no s'han modificat gaire les diferents fases que formen el pipeline d'OpenGL, ja que només s'han modificat el *Vertex Shader* i *Fragment Shader*, els quals processen els colors dels objectes, per crear el model d'il·luminació, el qual s'explicarà amb més detall posteriorment en l'apartat 8.4. No s'ha considerat necessari modificar res més perquè només fent aquest canvi ja s'aconsegueix un resultat a l'escena prou bo i adequat pel tipus de públic al qual va dirigit aquest *game engine*.

Així doncs, s'ha considerat que és interessant explicar com s'han treballat les transformacions dels sistemes de coordenades per tal que el món virtual, el qual sempre és en 3D, es vegi de forma correcta i adient per l'usuari. Aquest procés consisteix en el conjunt de modificacions en cadena que s'apliquen als vèrtexs de l'escena per adaptar els valors de les seves posicions a la càmera que mostra l'espai 3D. Els passos a seguir que s'han considerat més rellevants per explicar són els següents:

- **Passar de coordenades de model a món:** Aquest primer pas consisteix en transformar els vèrtexs dels models, els quals en un inici tenen els valors en funció del sistema que es va establir quan el model va ser dissenyat, a coordenades del món virtual. Aquest pas és molt important fer-lo per tal d'assegurar que en el moment en què s'apliquin transformacions geomètriques

als objectes, aquestes no es faran en funció de les coordenades internes del propi model. El procés tracta de traslladar l'objecte a l'origen de l'escena, aplicar-li les rotacions i escalats corresponents i finalment traslladar-lo a la posició del món on ha d'estar. Per fer aquests càlculs s'utilitzen les funcions de la llibreria glm, les quals internament tenen implementades operacions amb matrius, com per exemple les de la figura 11, les quals són les matrius que s'utilitzen per aplicar rotacions en cadascun dels 3 eixos cartesians.

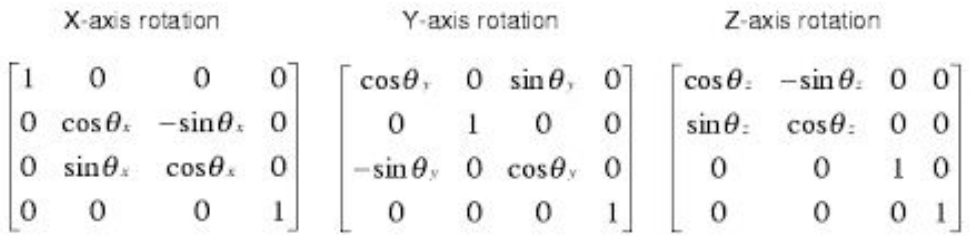

Figura 11. Matrius de rotació.

- **Passar de coordenades de món a càmera:** Un cop el model geomètric està definit en coordenades del món virtual, és necessari aplicar-li una nova transformació per passar-lo al sistema de referència de la càmera que mostra l'escena. En aquest cas també es poden utilitzar composicions de translacions i rotacions, les quals venen definides en funció de la posició i orientació de la càmera.

- **Passar de coordenades de càmera a *clipping*:** Aquest pas permet suprimir en l'escena tots els vèrtexs que no es vegin en la regió de visió definida, la qual es defineix en aquesta mateixa transformació i depèn del tipus de càmera (perspectiva o ortogonal). En el cas d'aquest projecte s'ha decidit que només es pot utilitzar la càmera perspectiva, però pel cas d'escenes 2D seria més interessant utilitzar l'opció ortogonal.

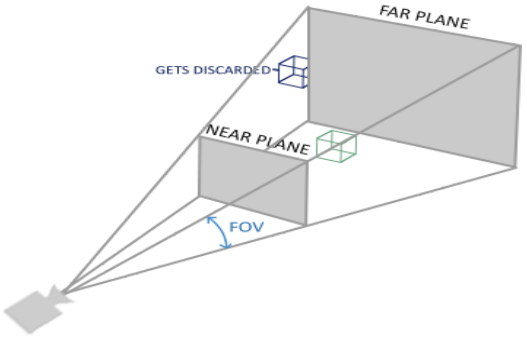

Figura 12. Esquema del frustum de la càmera.

Així doncs, gràcies a aquest procés de transformacions, és més senzill implementar les funcionalitats del motor, ja que només cal aplicar-ho a tots els elements de l'escena de forma automàtica i centrar-se en la resta d'operacions que componen les funcionalitats del motor.

7.3. Qt5

Qt és una aplicació que permet crear interfícies gràfiques entre altres coses, la qual en aquest projecte servirà per implementar tots els elements relacionats amb la interacció amb l'usuari, per tal que pugui fer les modificacions que prefereixi en l'escena 3D. Més concretament, pel desenvolupament d'aquest projecte s'ha utilitzat *QtDesigner*, el qual és una versió de Qt5 que permet crear una interfície de forma més visual i intuïtiva.

Aquesta aplicació conté un conjunt d'elements, anomenats *widgets*, que ofereixen diferents tipus d'interaccions segons l'ús que es vulgui donar, per exemple es poden afegir línies de text on l'usuari pot escriure, botons, *sliders*... D'aquesta manera es poden agrupar tots aquests elements amb diferents contenidors i generar l'estructura de la interfície. A més també existeix l'opció de modificar el comportament d'aquests *widgets*, promocionant-lo a una classe feta pel programador en la qual hi hagi implementat el nou funcionament i per tant, aconseguir adaptar els elements a les necessitats de l'aplicació.

Per altra banda, Qt ofereix un mecanisme molt pràctic per relacionar els *widgets* de la interfície entre ells, el qual consisteix en *signals* i *slots*. Els *signals* són senyals que envia un element quan hi ha hagut alguna interacció amb ell, la qual haurà definit el programador. En canvi els *slots* són els encarregats de captar els *signals* que s'activin i executar els canvis que aquest provoca. Per exemple en un botó es pot activar un *signal* quan es prem o quan es deixa de fer-ho, entre altres casos, i just després, els slots assignats a captar la interacció amb el botó, passaran a executar-se canviant l'estat del programa tal com hagi definit prèviament el desenvolupador. Es pot veure més clarament el concepte en el següent esquema, en el qual també es pot observar que un *signal* pot tenir més d'un slot assignat i viceversa.

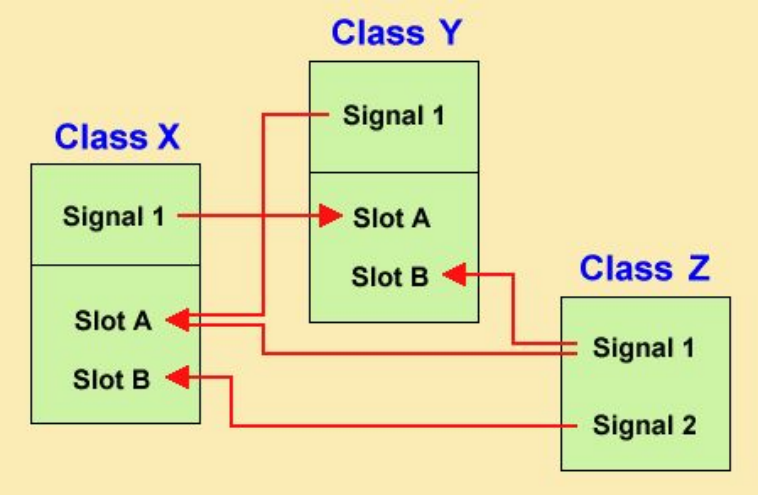

Figura 13. Esquema de *signals* i *slots*.

8. Implementació

En aquesta secció del document s'explicarà el funcionament de les diferents funcionalitats que s'han implementat en el motor de forma precisa, detallant com s'han programat i la idea general que tenen darrere. El projecte s'ha anat construint començant per les parts que són el nucli del programa i després es van anar afegint la resta de components.

8.1. Preparació del viewer OpenGL

Primerament, cal explicar que el projecte es basa en una interfície de Qt5, en la qual s'hi ha inserit un *OpenGL Widget* en el que es renderitzaran tots els gràfics de l'escena. Per tal que aquest element pogués mostrar tot el que es volia, s'ha promocionat a una classe pròpia en la que s'han afegit els mètodes, slots i signals necessaris que permeten executar totes les funcionalitats del motor i les relacions entre la interfície i el codi.

Al voltant del viewer gràfic d'OpenGL s'aniran afegint els diferents elements d'interacció (botons, caselles per escriure text...), els quals s'expliquen més en detall en l'apartat *8.9*. Això és degut a què s'ha considerat preferible explicar els components de la interfície un cop s'hagin detallat les funcionalitats implementades, per tal d'entendre de forma més clara que fa cada element que la forma. Així doncs un cop es va tenir aquest *form* muntat, es van començar a programar les propietats pròpiament pertanyents a un motor gràfic.

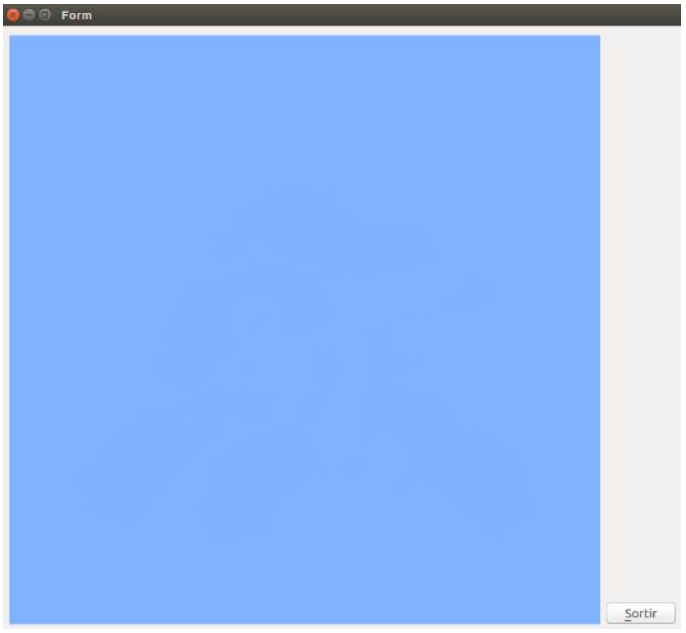

Figura 14. Captura del *form* base amb el viewer.

8.2. Gestió dels *GameObjects*

El primer que s'ha implementat en el desenvolupament del motor és el sistema de gestió dels *GameObjects*. Aquesta és la nomenclatura tècnica que tenen els objectes que componen les escenes en un *game engine* i són els components que fonamenten principalment el nostre sistema. Això és degut a que dins el motor aquests objectes tenen unes propietats i accions assignades i tots els canvis que es produeixen en el joc depenen d'aquests elements.

Així doncs aquest motor permet generar objectes en l'escena amb unes propietats per defecte i sense cap acció assignada. A més, aquest objecte creat passa automàticament a estar emmagatzemat en el controlador de domini. Les propietats que s'han considerat adequades que l'usuari pugui modificar per aquest sistema són les següents:

- Posició: Són les coordenades x,y,z de l'objecte en *World Space*. Per defecte aquestes els valors de les 3 components és igual a 0.

- Rotació: Són els valors dels 3 angles de rotació en els 3 eixos en graus sexagesimals. S'utilitza aquesta unitat de mesura perquè per l'usuari serà molt més senzill treballar-hi en comparació amb els radians. Per defecte els 3 components també valen 0.

- Escala: És la propietat que regula la mida de l'objecte en cadascun dels 3 eixos. Per defecte els 3 valors que conté valen 1, cosa que implica que l'objecte té la seva dimensió base, en canvi, si es posa algun dels components igual a 2 per exemple, augmentarà el doble el seu volum en l'eix especificat.

- Identificador: És el component de l'objecte que el distingirà de tots els altres, ja que el seu identificador és únic per cada *GameObject* que estigui generat a l'escena.

Per tant a partir d'aquestes propietats l'usuari pot situar l'objecte en el lloc de l'escena i amb la col·locació que ell prefereixi, tot i així encara falta escollir l'aspecte estètic que tindrà. Per fer-ho, el desenvolupador podrà escollir entre diferents models que s'han precarregat en el motor i aquests podran ser formes geomètriques simples, com un cub o una esfera, o podran ser models força més complexes com una figura de Lego o un Drac.

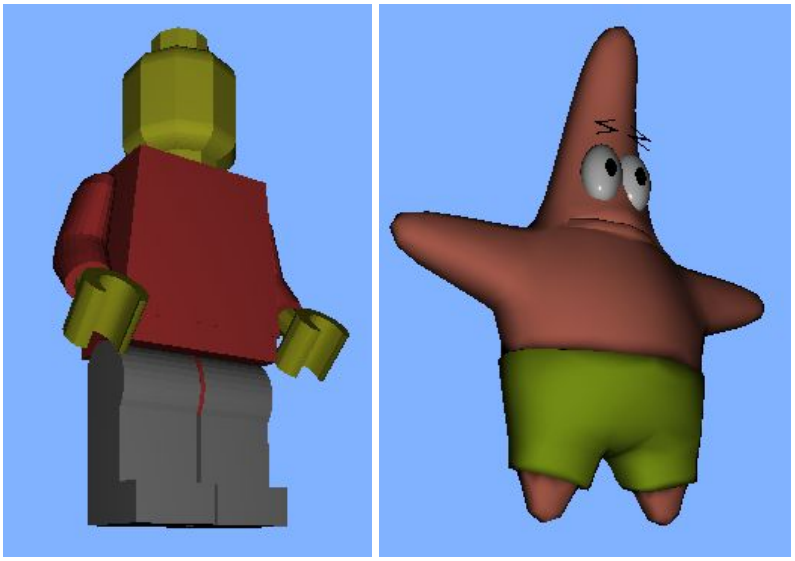

Figura 15. Captures d'alguns possibles models dels *GameObjects*.

Per altra banda, es va considerar una bona idea fer una distinció interna entre els diferents tipus d'objectes que hi ha en un joc, ja que en funció de l'entitat jugable que representen tenen un comportament diferent. Així doncs es va procedir a analitzar quines categories d'objectes apareixen en un joc de forma habitual intentant agrupar-los de la forma més distintiva possible segons les seves funcionalitats pràctiques dins d'un videojoc. D'aquesta manera es va decidir crear 4 grups de *GameObjects*: Jugador, Enemic, Col·leccionable i Objecte estàtic. Aquesta classificació es fonamenta en la idea de què el Jugador és el tipus d'objecte que es controla interactivament, l'Enemic és l'entitat que obstaculitza al jugador a complir el seu objectiu, el Col·leccionable és aquell amb el qual pot interactuar el jugador i l'Objecte estàtic és un component que forma el mapa del joc. A més també es va considerar necessari marcar un límit de 4 jugadors actius a l'escena, ja que són les entitats interactives i no es pot permetre a un usuari novell crear un nombre infinit de jugadors, ja que col·lapsaria el joc. Tot i així, aquesta és una decisió de disseny i aquest motor podria ser escalat per tal de permetre un nombre més gran de jugadors actius, però sempre tenint en compte les capacitats suportades per la targeta gràfica.

Una vegada es va tenir clara aquesta divisió, es va implementar de manera que l'usuari pugui generar un *GameObject* decidint quin d'aquests 4 tipus vol. Això permet crear una visió més senzilla al desenvolupador de l'estructura del joc que està creant, ja que si vol crear un enemic per exemple, aquest ja tindrà un comportament que vindrà predefinit pel motor i ell no haurà de preocupar-se en detallar aquesta distinció. Internament, el que s'ha fet ha estat crear 4 subclasses filles de la classe *GameObject* i així estructurar els diferents comportaments de cada grup més precisament, mantenint les propietats comunes entre ells.

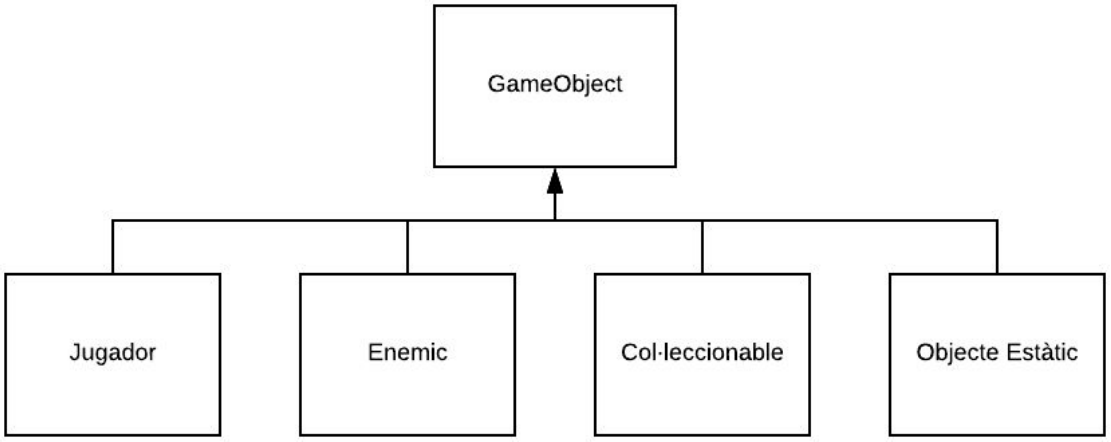

Figura 16. Classificació dels *GameObjects*.

En conclusió, aquesta funcionalitat consisteix en què l'usuari podrà crear, modificar i eliminar objectes de l'escena, detallant les seves propietats, tant geomètriques com estètiques, i concretant quin tipus d'entitat jugable serà en funció del seu comportament desitjat en el joc.

8.3. Interacció amb la càmera

La següent funcionalitat que es va implementar va ser la interacció amb la càmera de l'escena, la qual permet a l'usuari moure's i fer *zoom* lliurement per l'escena i crear el seu món virtual amb facilitat i comoditat. El funcionament d'aquesta càmera mòbil consisteix en només haver d'utilitzar el ratolí, de manera que amb els botons dret i esquerre es pot traslladar i rotar la vista respectivament i amb la roda ajustar el *zoom*. S'ha considerat que aquests controls són els més adequats per tal que aquesta funcionalitat s'executi de forma intuïtiva i senzilla per part del desenvolupador.

Així doncs, les mecàniques de canviar la posició de la càmera i de fer zoom tenen una implementació moderadament simple, ja que només és necessari aplicar translacions a l'escena per simular correctament l'efecte desitjat. En canvi, la programació de la rotació de la vista té més complexitat geomètrica i això és degut al fet que s'han utilitzat angles d'Euler[19].

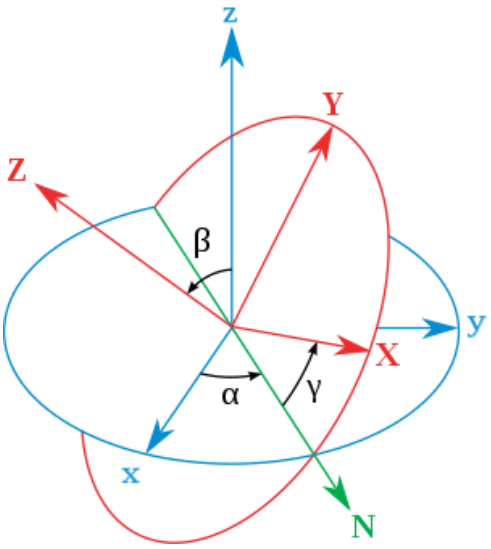

Figura 17. Esquema dels angles d'Euler.

Els angles d'Euler són 3 angles creats pel matemàtic suís Leonhard Euler que serveixen per descriure l'orientació d'un sistema de coordenades mòbil respecte a un altre que està fixat. El concepte es pot entendre gràcies a la figura anterior. En l'esquema que es pot veure existeixen 2 sistemes de coordenades, el blau i el vermell, dels quals el primer està fixat i l'altre té una orientació mòbil. Llavors el primer pas és escollir un pla en els 2 sistemes de referència, els quals han de ser homogenis entre ells (això significa que sí que es pot escollir el parell de plans xy i XY', però no el parell xy i XZ). Seguidament, on intersecccionen els 2 plans escollits es traça la línia verda i a partir d'ella ja podem extreure els 3 angles d'Euler.

- α és l'angle entre la línia verda i l'eix x.
- β és l'angle entre els eixos z i Z.
- γ és l'angle entre la línia verda i l'eix X.

Per tant, un cop explicat el procediment matemàtic, la manera d'implementar-ho en el motor ha estat tenir guardats aquests tres angles, els quals indiquen l'orientació de la càmera respecte als eixos cartesians sense cap rotació, i anar modificant els seus valors en funció de la direcció en la que l'usuari mou el ratolí amb el botó dret premut.

Per altra banda, un cop va estar implementada la càmera en el mode desenvolupador del sistema, es va procedir a programar-la pel mode joc, és a dir, programar la càmera que es veuria durant l'execució del joc. És important recalcar que la càmera que utilitza el jugador no ha de ser necessàriament la mateixa que la que utilitza el desenvolupador, ja que la segona ha de permetre molta més llibertat de moviments i la primera s'ha de limitar a la idea del joc. Així doncs, en aquest *game*

engine es permet a l'usuari canviar a la càmera del joc, modificar la seva posició i rotació com es vulgui, i tornar a la càmera de desenvolupador només seleccionant i desseleccionant una casella per triar quina de les 2 vol visualitzar. D'aquesta manera qui utilitzi aquesta aplicació, podrà tenir control sobre les 2 vistes de forma senzilla.

Finalment, un últim detall sobre aquesta funcionalitat és que quan l'usuari crea un *GameObject*, la càmera enfoca directament a l'objecte creat per tal que no es perdi temps buscant en quin punt de l'escena s'ha generat i així facilitar encara més la feina al desenvolupador.

Per concloure, la interacció amb la càmera consisteix en què es pugui moure, rotar i apropar dins l'escena lliurement només utilitzant el ratolí i que aquesta es mogui automàticament a qualsevol nou objecte creat. A més, es crea una independència entre les càmeres del mode desenvolupament i mode joc i una fàcil accessibilitat entre elles, cosa que permet treballar sobre les escenes de forma còmoda i pràctica.

8.4. Il·luminació

El sistema d'il·luminació és la part de la implementació on entra en joc la programació de shaders, ja que com s'ha dit abans en l'apartat focalitzat en OpenGL (apartat 7.2), són els encarregats de processar els colors dels vèrtexs i per tant també processen la informació relacionada amb la llum de l'escena.

Així doncs el que s'ha fet és generar un focus de llum en un punt de l'escena, el qual l'usuari té l'opció d'escollir la seva posició, que dóna un efecte lumínic en el que tots els punts de l'escena en els quals arriba llum des del focus la reben de la mateixa forma. D'aquesta manera, a partir de la posició d'aquest punt de llum els shaders s'encarreguen de fer els càlculs geomètrics necessaris per processar el color de cada fragment dels models, els quals depenen del model d'il·luminació utilitzat.

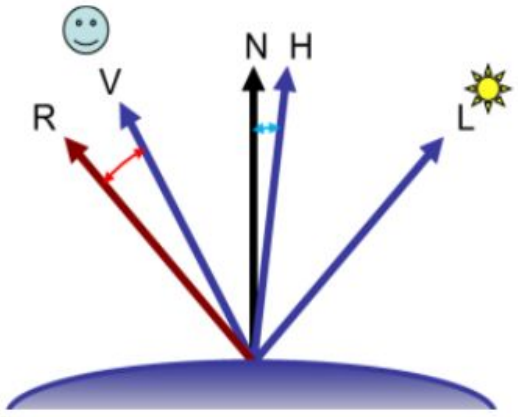

Figura 18. Esquema del model de Phong.

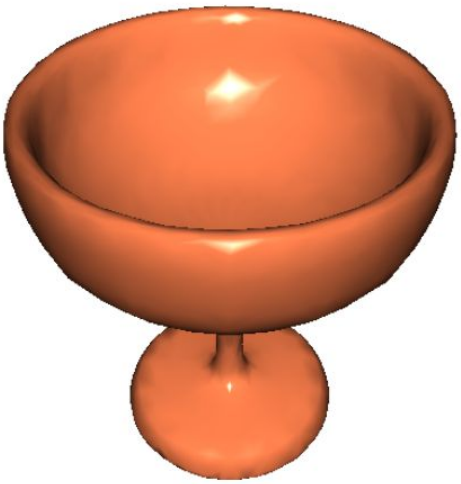

Figura 19. Copa amb el model de Phong.

En el cas del motor que s'està creant s'ha decidit utilitzar el model d'il·luminació de Phong[19], el qual dóna un aspecte als objectes com el que es mostra a la figura 19. S'ha optat per aquest model perquè s'ha considerat que mostra l'escena de forma força estètica i pel tipus d'usuari al qual va dirigit el joc no és necessari que hi hagi una il·luminació hiperrealista i a més, al ser un model empíric, el seu temps de càlcul és molt ràpid. Entrant més en detall, aquest model utilitza les components ambient, difusa i especular de la llum de l'escena i les suma utilitzant la següent fórmula:

$$K_aI_a + K_dI_d(N \cdot L) + K_sI_s(R \cdot V)^s$$

Seguint l'esquema de la figura 18, el significat de les variables que apareixen en la fórmula són els següents:

- Ka, Kd, Ks = reflectivitat ambient, difosa i especular del material
- s = factor de brillantor del material.
- Ia, Id, Is = propietats ambient, difosa i especular de la llum
- N = vector normal unitari
- L = vector unitari cap a la font de llum
- V = vector unitari del vèrtex cap a la càmera
- R = reflexió del vector L respecte N. Es pot calcular com R=2(N·L)N-L

Així doncs, com que en aquesta aplicació es vol que tots els punts de l'escena rebin la llum de la mateixa manera, però que aquesta pugui variar segons la posició escollida per l'usuari del focus de llum, l'única component que es modificarà serà el vector L, el qual només defineix la direcció des de la qual es rep la llum. Per tant amb aquest sistema d'il·luminació es genera un efecte força estètic a l'escena i es permet al desenvolupador canviar la posició d'on prové la llum de la manera que més s'adeqüi al seu joc.

8.5. Mode *Game*

A continuació, un cop es va tenir preparada la part bàsica del mode desenvolupador, el qual permetia generar una escena amb objectes i càmera tal com volgués l'usuari, es va procedir a crear el mode *Game*. Aquest mode és aquell que dóna l'opció de simular el joc que s'està creant en temps real i d'aquesta manera poder veure si funciona correctament i si l'experiència de joc és la desitjada.

Així doncs, la interacció que té l'usuari amb aquesta funcionalitat en el motor que s'està construint és força simple, ja que només cal prémer el botó *Play* per encendre la simulació i el botó *Pause* per parar-la mentre s'està executant. En el moment en què s'activa el joc, tots els elements de la interfície que pertanyen al mode

desenvolupador passen a estar deshabilitats, per tal d'impedir que l'usuari modifiqui dades de l'escena durant la simulació. És important vigilar aquesta possibilitat perquè un joc s'executa amb unes dades definides, i si aquestes canvien de sobte, la consistència del joc es veurà afectada perquè poden haver-hi canvis en dades que no siguin congruents amb la lògica i estructura que s'estava seguint. A més, com que l'aplicació s'està construint per gent que no ha de saber aquest tipus de conceptes, és més rellevant encara prevenir que no pugui succeir.

A nivell implementació, durant el mode joc, el programa tindrà dues tasques principals que haurà de seguir per tal d'executar totes les accions dels objectes. La primera d'elles serà estar pendent de qualsevol event de teclat que es detecti, i això ho farà utilitzant la classe de Qt *QKeyEvent[20]*, la qual permet captar les entrades de teclat a l'aplicació. Aquesta tasca servirà per gestionar les accions que s'activen a partir de les tecles, les quals haurà creat l'usuari abans de l'execució i funcionament de les quals es detallarà més profundament en l'apartat 8.6. Per tal de tenir un ràpid accés a aquest tipus d'accions, es va crear un diccionari que relacionava cada tecla amb l'acció que s'havia d'executar, i a més, també és útil per controlar que no hi hagi dues accions assignades al mateix botó del teclat.

Per altra banda, la segona tasca que ha de complir aquesta funcionalitat és executar de forma contínua les accions independents de l'entrada per teclat, és a dir, aquells comportaments que succeeixen de forma automàtica durant la simulació. Per implementar-ho s'ha utilitzat la classe de Qt *QtTimer[21]*, la qual permet activar un temporitzador que vagi repintant l'escena constantment amb les dades actualitzades correctament en cada moment. Així doncs, durant la simulació del joc es van executant les accions automàtiques, com pot ser un moviment constant d'un objecte de dreta a esquerra, de forma que la informació de l'escena va canviant i es va pintant alhora en el viewer d'OpenGL.

Finalment, quan el control de les accions està implementat, només queda programar la gestió de les dades del món 3D, ja que durant la simulació es van modificant, però en el moment en què aquesta s'atura els objectes han de tornar a l'estat previ a l'execució del joc. Per tant en el moment en què es prem el botó *Play* es guarden totes les dades dels objectes que estan subjectes a canvi en una estructura de dades a part, i quan l'usuari decideix aturar es tornen a passar les dades originals als objectes de manera que mantinguin el seu estat previ a simular el joc. A més, les estructures de dades on s'havia copiat la informació s'eliminen per tal d'optimitzar la despesa en memòria, ja que només s'utilitzen durant l'execució del joc, la qual no ocupa un gran percentatge del temps d'ús de l'aplicació.

Per concloure doncs, el mode *Game* consisteix en la simulació del joc en temps real, en la qual l'usuari pot provar com evoluciona el seu desenvolupament. Aquesta funcionalitat és la que mostra l'escena amb la càmera de joc que s'havia especificat amb anterioritat i gestiona els 2 tipus d'accions existents en el sistema: accions a l'espera d'entrades per teclat i accions amb comportament automàtic i continu.

8.6. Sistema d'Accions

El sistema d'accions és l'encarregat de fer que els objectes tinguin un comportament establert durant el funcionament del videojoc. En l'apartat anterior, s'havien agrupat les accions segons el tipus d'activació que tenien, és a dir, si necessitaven o no una entrada per teclat per executar-se. En canvi, a l'hora de classificar-les de cara a l'usuari, s'han distingit segons el tipus de comportament que tenen i s'han creat 2 grups ben diferenciats. El primer d'ells engloba les accions que tracten els moviments i transformacions geomètriques dels objectes dins l'escena i el segon inclou aquelles que tracten les interaccions entre *GameObjects*. Tot i així, abans d'especificar amb més detall les accions en concret que l'usuari pot seleccionar, s'explicarà la idea general del funcionament d'aquest sistema en el motor que s'està creant.

Així doncs, el funcionament per crear i assignar una acció a un objecte és ben senzill: primerament l'usuari seleccionarà un dels objectes que estan creats i a continuació triarà de quina de les accions disponibles vol fer ús. Les accions disponibles són aquelles que el tipus concret de *GameObject* amb el que s'estigui treballant permet que se li siguin assignades, és a dir, si es vol afegir una acció a un enemic, només es podran seleccionar accions que un enemic pugui executar, i aquestes accions poden ser o no ser compartides amb els altres tipus d'objectes d'aquest motor. A més és important destacar que algunes accions bloquegen la possibilitat d'elecció d'algunes altres, ja que hi ha comportaments incompatibles entre algunes d'elles. Aquesta restricció és important controlar-la en aquest *game engine* que s'està generant perquè facilita molt el desenvolupament a l'usuari, el qual no ha de pensar les compatibilitats entre les accions i pot assignar-les de forma molt senzilla.

Un cop el desenvolupador ha seleccionat l'acció que vol afegir a l'objecte, podrà concretar els paràmetres d'aquesta tal com ell més ho prefereixi segons les necessitats del joc que està creant. Aquestes propietats específiques són diferents en la majoria de les accions, ja que cadascuna segueix un comportament diferent i ha de tenir en compte diferents variables. Per exemple en el salt és necessari indicar l'altura màxima d'aquest, cosa que no succeeix en les altres accions. A més, dins d'aquests paràmetres, s'inclou el mode d'activació de la tecla, és a dir que l'usuari pot escollir si s'executarà de forma automàtica o determinar quina tecla del teclat té assignada.

Llavors quan s'han determinat els paràmetres i l'usuari ha decidit guardar-los, al sistema intern passa a existir la nova acció amb les propietats escollides i un objecte de l'escena assignat a ella i per tant, en el moment en què es passa al mode *Game* es gestiona aquesta acció tal i com s'ha explicat en l'apartat anterior. Més

concretament, el que es fa és cridar a la funció corresponent que hi ha dins de la classe del *GameObject* que té l'acció, ja que allà és on hi ha les dades que el representen i és més òptim executar l'acció en la pròpia classe. A més, com que cada tipus d'objecte està definit en subclasses diferents i cadascuna té implementat un comportament diferent, l'acció s'executarà tenint en compte la diferència entre els tipus d'entitats del joc.

De cara a l'usuari, el sistema d'accions és senzill d'utilitzar i accessible, ja que per ell tot aquest sistema intern és invisible i ell l'únic que fa és seleccionar l'objecte al qual vol afegir una acció, seleccionar quina i definir els seus paràmetres. A partir d'aquí, un cop comença la simulació del videojoc les accions es desenvolupen com ell ha determinat. A més, el desenvolupador també té permès canviar les propietats de les accions existents i eliminar-les si és necessari, sempre mantenint correctament la consistència de l'aplicació.

A continuació es passarà a explicar més amb detall quines accions s'han implementat i com.

8.6.1. Accions de moviment

Aquest tipus d'accions són aquelles que engloben els canvis de posició i canvis geomètrics de l'objecte.

Desplaçament

Si l'usuari vol que l'objecte es mogui en alguna direcció li ha d'afegir una acció de desplaçament, la qual li permet moure's en línia recta en els 3 eixos de coordenades en sentit positiu i negatiu, és a dir que es pot moure de 6 maneres diferents (x+, x-, y+, y-, z+, z-). Això es va considerar que no era suficientment clar pel tipus de públic al què va dirigit aquest motor i per tant es va decidir ramificar aquesta acció en 6 diferents: Anar endavant, Anar endarrere, Anar a l'esquerra, Anar a la dreta, Baixar i Pujar. Aquest canvi permet que l'usuari vegi de forma més clara quina direcció vol que tingui el moviment, sense haver de pensar en conceptes geomètrics.

Tot i així es va veure que si els moviments predefinits que s'oferien no tenien en compte l'orientació de l'objecte, per l'usuari seria molt més complex utilitzar aquestes accions, perquè si un *GameObject* té aplicada una certa rotació, podria donar-se el cas en què l'usuari vulgui que l'objecte avanci a l'esquerra i vagi endarrere, per exemple. Per tant es va decidir fer els desplaçaments en el sistema de coordenades generat en funció dels angles de rotació i d'aquesta manera, si l'usuari vol que l'objecte es mogui endavant, sempre ho farà segons la seva orientació.

Per tal d'explicar de forma més entenedora com s'ha implementat aquesta acció tenint en compte la rotació de l'objecte, s'utilitzarà l'esquema que hi ha a continuació:

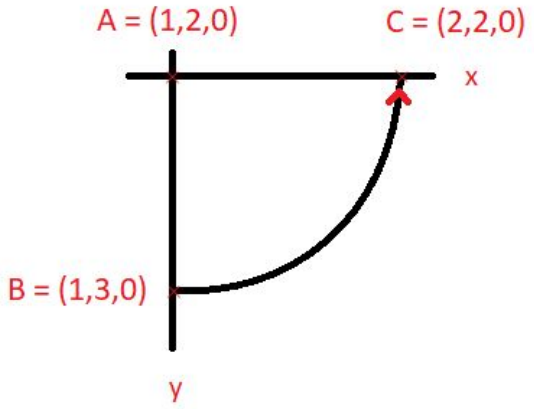

Figura 20. Esquema acció desplaçament.

Partint d'aquest exemple, se suposarà que tenim un objecte en el punt A, el qual està en la posició = (1,2,0) i està girat en l'eix Z 90º, és a dir que actualment està mirant en direcció x+. Llavors l'usuari vol que el *GameObject* es mogui una unitat cap endavant, per tant des de la seva perspectiva hauria de traslladar-se al punt C, ja que està orientat en aquella direcció. Ara bé, com que la implementació de l'acció anar endavant no sap en quina orientació en concret està l'objecte, està programada perquè l'objecte es mogui al punt B sempre i tot seguit, mitjançant matrius de rotació, s'aconsegueix trobar el punt on ha d'anar realment el *GameObject*. Aquestes matrius que s'utilitzen són les que s'han comentat en l'apartat 7.2, les quals utilitzen els angles que descriuen l'orientació de l'objecte.

A banda d'això, aquest conjunt de 6 accions té dos paràmetres: el mètode d'activació i la velocitat del moviment. El primer d'aquests es defineix seleccionant una de les tecles del teclat que permet l'aplicació o bé escollint l'opció de moviment automàtic. El segon paràmetre senzillament és el valor que fa augmentar o disminuir la velocitat del moviment.

Així doncs, l'usuari amb aquestes 6 accions pot fer moure un objecte en qualsevol de les 6 direccions possibles, de forma automàtica o prement una tecla per cadascuna, tenint en compte l'orientació. A més, aquestes accions es poden assignar a jugadors i enemics, amb el matís que en el segon cas no es pot assignar una tecla al moviment, sinó que sempre es fa automàticament, ja que un enemic no és una entitat controlable en un joc.

Rotació

Aquesta acció permet que l'usuari canviï l'orientació de l'objecte en l'eix que ell desitgi. Així doncs, les propietats que conté són: mode d'activació, velocitat i eix de rotació, dels quals els 2 primers tenen el mateix funcionament que s'ha explicat en els desplaçaments. El tercer paràmetre defineix en quin eix es produirà la rotació de

l'objecte i en una mateixa acció només se'n pot seleccionar un alhora. Aquesta restricció s'ha imposat per tal de simplificar més el concepte de la rotació i si l'usuari vol que l'objecte pugui girar en els 3 eixos haurà d'assignar 3 accions diferents en les quals s'especifiqui un eix diferent. D'aquesta manera el desenvolupador pot tenir una tecla diferent per cada eix de rotació i tenir un control del personatge més consistent durant la simulació del joc.

Així doncs l'usuari pot assignar a qualsevol tipus d'objecte aquesta acció, però seguint la mateixa restricció que en el cas anterior, només es permet rotar com a conseqüència d'una tecla al jugador. Per tant amb la rotació i els desplaçaments disponibles, l'usuari ja pot atribuir un moviment totalment lliure per tot l'espai als objectes, ja que amb la rotació ha desbloquejat la possibilitat de canviar l'orientació del *GameObject* i així poder-se desplaçar per tots els punts del món virtual.

Escalat

Aquesta acció és la que permet a l'usuari canviar la mida d'un objecte durant l'execució del joc. Això pot ser útil en situacions com per exemple agafar un objecte que faci canviar el tamany al personatge o fer canviar les seves dimensions en moments concrets a conveniència del joc. L'escalat està disponible únicament pels jugadors.

Els paràmetres que conté l'acció són: la tecla d'activació i el factor d'escala. El primer d'ells aquesta vegada no pot ser el comportament automàtic, ja que s'ha considerat que en aquest cas un objecte que escali de forma contínua durant l'execució del joc no seria una bona opció per l'usuari. Això és degut a que a la llarga l'objecte acabaria sent molt petit o realment enorme, cosa que a un usuari sense gaires coneixements sobre l'àmbit li pot suposar un problema i s'ha preferit no donar-li aquesta possibilitat. Per altra banda, el factor d'escala és un sol número que s'aplica sobre les 3 dimensions de l'objecte de forma uniforme per tal de facilitar el concepte al desenvolupador i que per tant, si vol duplicar la mida d'un objecte, només hagi de posar aquest factor igual a 2, per exemple. En el cas que s'ha explicat anteriorment en què l'usuari vulgui canviar les propietats d'escala del *GameObject* en mode desenvolupador, sí que està permès modificar els 3 eixos de forma independent per tal que es pugui modificar la forma de l'objecte.

Salt

L'acció de salt permet a un jugador fer un salt vertical, el qual pot estar combinat amb altres moviments ja comentats, com una rotació o un desplaçament, convertint-lo en una acció a l'aire. D'aquesta manera, aquesta acció té una altura màxima a la qual pot arribar l'objecte respecte al punt on ha començat el moviment i que l'usuari pot especificar com atribut de forma relativa a la posició actual de l'objecte, és a dir, que si el desenvolupador considera que l'altura de salt és 3, l'objecte farà un salt arribant a 3 unitats per sobre de la posició en què estava.

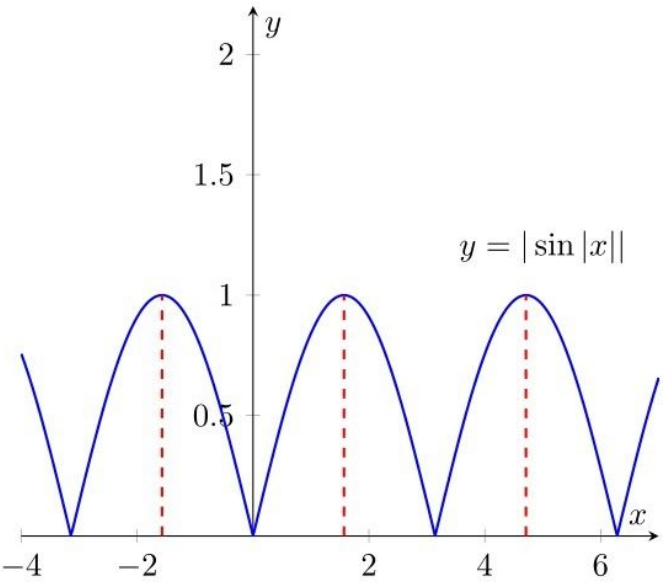

Figura 21. Funció sinusoidal amb valor absolut.

Per tal que el salt sigui una acció fluida i que es vegi natural s'ha utilitzat una funció sinusoidal amb valor absolut per simular el moviment vertical, com es pot veure en la figura anterior. D'aquesta manera en cada unitat de temps varia el valor de l'eix vertical de la posició de l'objecte donant un efecte pràcticament idèntic a un salt real. La fórmula per calcular l'altura en cada instant de temps és la que apareix a continuació, en la qual el valor de x és el valor d'un angle que va de 0 a π i que es va incrementant en funció del temps.

$$y = \text{Altura màxima} * |\sin(x)|$$

Com es pot veure en la fotografia anterior, allà el valor de l'altitud màxima és 1 i per tant el senyal no passa mai d'aquest valor en l'eix y. Així doncs, amb aquesta implementació l'usuari només ha d'establir un valor a l'altura i el moviment es realitzarà generant l'efecte desitjat. A més també cal dir que es va plantejar el dubte de si utilitzar la funció sinus o cosinus, però es va poder veure ràpidament que la primera era més simple, ja que no calia afegir un desfasament a l'angle inicial.

Per altra banda, aquesta acció torna a permetre l'elecció en el mode d'activació, cosa que gràcies a la funció sinusoidal és molt senzill de programar, ja que si el salt és constant, només cal repetir contínuament la funció i si s'activa en funció d'una tecla, només caldrà fer un dels cicles en què l'angle x avança de 0 a π. A més a més, el salt s'ha implementat restringint que es pugui saltar mentres un altre salt està actiu, donant l'opció al desenvolupador de només poder fer un salt simple però evitant que es puguin fer salts infinits sense tornar a l'altura original.

Així doncs amb aquesta acció i les implementades anteriorment, l'objecte es pot moure lliurement per l'escenari i tenir desplaçaments per l'aire.

Desplaçaments repetitius

Aquesta acció consisteix en el moviment constant, automàtic i repetitiu d'un objecte simulant el comportament d'un pèndol en un dels 3 eixos, és a dir que permet a un GameObject desplaçar-se cap a una certa direcció durant una distància concreta, després canviar el sentit del desplaçament i seguir així de forma infinita. D'aquesta manera, per exemple, l'usuari pot fer que una plataforma es mogui d'esquerra a dreta continuadament només definint la distància que vol que duri el recorregut sencer, i un cop s'executi el joc el moviment es produirà de forma autònoma.

Així doncs, per tal de simplificar més les eines que s'ofereixen al desenvolupador, es va decidir ramificar aquesta acció en 3 comportaments diferents: Desplaçament de dalt a baix, Desplaçament d'esquerra a dreta, Desplaçament de davant a darrere. Amb aquesta classificació es permet a l'usuari definir de forma més senzilla aquest tipus de desplaçament, ja que només haurà d'escollir un dels 3 casos i especificar la distància.

En aquest cas, aquesta acció sempre és automàtica, ja que en la majoria dels casos on es pot utilitzar, no s'ha considerat que tingui sentit una activació del moviment mitjançant una tecla. Per altra banda, aquest tipus de desplaçament es pot assignar a enemics, col·leccionables i objectes estàtics i no es pot afegir en un jugador perquè aquest ha de poder tenir controlat el seu moviment la persona que juga al videojoc.

8.6.2. Interaccions

Aquest segon grup d'accions són aquelles que tracten les interaccions entre les entitats del joc, les quals succeeixen en el moment en què dos objectes col·lisionen entre ells. Per tal de definir quins comportaments havia de permetre tenir en compte l'entorn que s'està desenvolupant, es va procedir a analitzar quins tipus de relacions entre objectes succeeixen més habitualment en els videojocs, simplificar-les i introduir-les en el sistema. Així doncs, per trobar aquestes accions i deduir la forma més general d'aquestes, es van estudiar les relacions directes que solen tenir els diferents tipus d'entitats existents en l'aplicació, per exemple, què sol passar quan un jugador i un enemic col·lisionen?

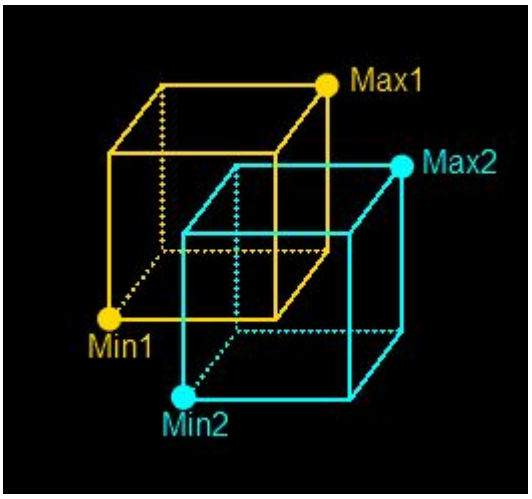

Figura 22. Solapament de caixes.

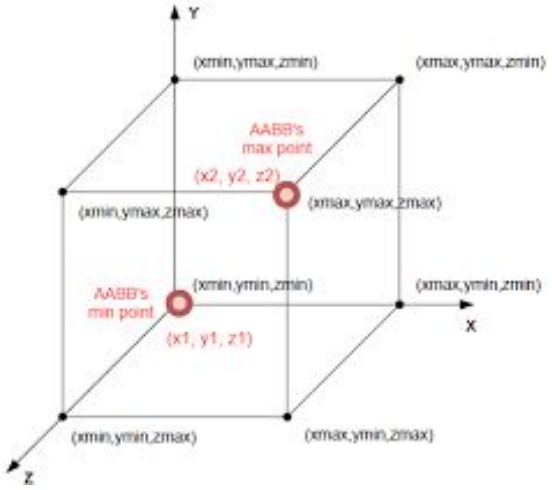

Figura 23. Estructura de la caixa contenidora.

Tot i així, abans d'especificar quines són les accions en concret, s'explicarà com funciona el sistema de col·lisions implementat. Cada objecte de l'escena pot estar contingut dins d'una caixa contenidora imaginaria que marca els límits del seu cos i per tant, per detectar si dos objectes impacten, només cal comprovar si les 2 caixes que els envolten se superposen en algun punt, com es pot veure en la figura 22. Per generar aquesta capsa cal trobar els punts màxims i mínims, en cadascun dels 3 eixos, on l'objecte té vèrtexs i la caixa es formarà a partir d'aquests punts tal com es mostra en la figura 23. A més, s'ha decidit que el propi objecte tindrà la seva caixa contenidora com a atribut, incloent-hi així també les possibles modificacions d'aquesta si el *GameObject* pateix alguna transformació geomètrica, per tal de no haver de calcular la caixa cada vegada que es comprova una col·lisió. Gràcies a aquest mecanisme es poden detectar si dues caixes se superposen si es compleix la següent condició, tenint un objecte A i B.

```
(aminx <= bmaxx and amaxx >= bminx) and
(aminy <= bmaxy and amaxy >= bminy) and
(aminz <= bmaxz and amaxz >= bminz)
```

Així doncs, per tal que les accions que s'anaven a implementar seguissin un comportament sòlid, es van haver d'afegir unes propietats als objectes que regulessin el seu comportament en el joc. Per tant es va procedir a afegir vida a tots els *GameObjects*, poder d'atac als jugadors i enemics i poder de curació als col·leccionables i jugadors. Aquestes característiques són valors numèrics que qualifiquen el nivell dels objectes en cadascun d'aquests àmbits i per tant serveixen per poder mesurar de forma quantitativa els canvis d'estat que es produeixen en una interacció. A més, quan un objecte arriba a tenir 0 de vida es considera que ha mort i a partir d'aquell moment es deixen d'executar les seves accions i no es pinta a l'escena. A continuació es poden veure quines són les accions específiques que s'han implementat:

Jugador ataca enemic

Si s'afegeix aquesta acció a un jugador, vol dir que quan col·lisioni amb un enemic, aquest segon rebrà un atac per part del primer i perdrà tanta vida com tingui el poder d'atac de l'agressor. L'usuari podrà especificar el nivell d'aquest poder i quan s'iniciï la simulació, l'acció es produirà de forma automàtica en cada impacte entre aquests dos tipus d'entitats.

Jugador agafa col·leccionable

Quan l'usuari assigni aquesta acció a un jugador implicarà que aquest podrà agafar un col·leccionable en el moment en què el toqui. En el moment en què això passi el col·leccionable desapareixerà de la partida i el jugador tindrà constància de què té un objecte més. En aquest cas el desenvolupador no haurà d'especificar cap paràmetre.

Enemic ataca jugador

Aquesta acció és la versió oposada a la primera que s'ha explicat, ja que és el cas en què un enemic ataca a un jugador. De forma similar a com succeïa abans, l'usuari afegirà aquesta acció a un enemic, concretant el poder d'atac que té aquest, el qual farà perdre vida al jugador en qualsevol impacte entre ells. Per tant amb aquesta acció i la que ja s'ha explicat es permet que els enemics i jugadors tinguin una relació d'interacció mútua si el desenvolupador ho desitja.

Col·leccionable cura jugador

En aquest cas, si l'usuari afegeix aquesta acció a un objecte col·leccionable i especifica el poder de curació, aconsseguirà que quan aquest col·lisioni amb un jugador, aquest segon recuperi la quantitat de vida equivalent al poder que s'ha detallat. D'aquesta manera s'aconsegueix que quan el jugador agafi un objecte rebi alguna cosa a canvi, la qual és vida amb aquesta acció.

En conclusió, aquest sistema d'accions permet a l'usuari generar l'escenari d'un joc en els que els objectes es poden moure lliurement per ell, sigui de forma automàtica o mitjançant tecles, i fer diverses interaccions entre ells que modifiquen els seus estats.

8.7. *Game Conditions*

Un cop l'usuari té la possibilitat d'organitzar l'escena i els comportaments dels objectes segons les seves preferències, per tal que es pugui arribar a crear un joc falta l'opció de definir quan aquest acabarà, definint quan l'usuari guanya o perd. Si no s'implementés aquesta característica, l'usuari podria crear una escena on tots els objectes podrien executar les seves accions però mai podria especificar quan es considera que l'usuari ha guanyat o perdut i per tant quan acaba el joc.

Així doncs, com s'ha fet en algunes funcionalitats anteriors, es va procedir a analitzar quines són les condicions de victòria i derrota més comunes en els jocs actuals i es van seleccionar les que es van considerar més generals i que proporcionaven un espectre més ampli en les possibilitats de jocs que es poden generar. Per tant les condicions escollides són les següents:

Condicions de victòria

- **Obtenció de col·leccionables**: Quan aquesta condició està activa, el jugador guanya la partida quan ha aconseguit obtenir una quantitat de col·leccionables concreta, la qual ha estat determinada quan es crea el joc per part del desenvolupador. Perquè aquesta condició es pugui complir és necessari que el jugador tingui assignada l'acció que li permet agafar col·leccionables.

- **Arribar a la meta**: Quan aquesta condició està activa, el jugador pot guanyar la partida en el moment en què aconsegueix sobrepassar una posició en concret de l'escenari. Per definir aquesta meta, el desenvolupador pot especificar un pla definint un dels 3 eixos de coordenades i un valor, per exemple z i 5 respectivament, i en el moment en què el jugador traspassa el pla on z = 5, es considera que ha guanyat. Aquesta condició pot ser molt útil per jocs on l'objectiu és arribar a un lloc en concret del mapa.

- **Eliminar enemics**: Quan aquesta condició està activa, el jugador aconsegueix guanyar la partida en el moment en què ha eliminat a un cert nombre d'enemics. Com en el cas dels col·leccionables, el desenvolupador és l'encarregat de determinar aquest número i és necessari que el jugador tingui assignada l'acció que li permet eliminar enemics.

Condicions de derrota

- **Mort del jugador**: Quan aquesta condició està activa, es considera que la partida està perduda quan han mort un nombre en concret dels jugadors. El desenvolupador té la capacitat de determinar aquest número dins el límit de 4 jugadors que hi ha marcat en el sistema i s'ha d'haver assignat als enemics l'acció d'atacar als jugadors, ja que són l'única entitat que els pot eliminar.

8.8. Sistema d'àudio

Un cop l'usuari ja té l'opció de crear un joc amb els seus elements, comportaments, càmera, condicions... és el moment en què es pot afegir una nova funcionalitat que li proporciona la possibilitat d'afegir un grau de qualitat als seus videojocs, i aquest nou element que s'ha implementat és el sistema d'àudio. Amb aquest sistema el desenvolupador pot donar més personalitat i immersió a les seves creacions, alhora que li permet modificar-ne l'estil i l'ambientació de la forma que més ho prefereixi.

Així doncs es va decidir que els 2 casos en què es permet reproduir un so en els jocs creats mitjançant aquest *game engine* són quan hi hagi una execució d'una acció en la qual s'ha assignat un efecte de so i durant la simulació de tot el joc amb música de fons. S'ha pensat que si s'oferien aquestes 2 opcions a l'usuari ja disposa d'un espectre de possibilitats suficient pel desenvolupament, ja que amb això es pot crear un joc amb música en el qual moltes de les accions dels objectes produeixen un efecte acústic.

Entrant més en detall, les accions en les que existeix l'opció d'afegir àudio són totes les interaccions entre objectes (explicades a l'apartat 8.6.2) i el salt, ja que són les accions en què un efecte de so queda millor. L'usuari pot seleccionar quina font d'àudio reproduir en cada cas de la mateixa manera que especifica els paràmetres de la pròpia acció i s'han afegit efectes com per exemple un so típic en videojocs d'agafar una moneda.

Per altra banda, si el desenvolupador vol que es reprodueixi música de fons en el seu videojoc, podrà seleccionar, com una de les característiques generals del joc, quina font d'àudio reproduir, la qual sonarà en bucle fins al final de l'execució. El tipus de melodies que s'ofereixen a l'usuari són bandes sonores clàssiques del món del videojoc, com per exemple les músiques del Zelda o del Sonic.

Per tant, amb aquesta nova funcionalitat el desenvolupador pot generar jocs molt més atractius i donar-li jugabilitats més interessants, ja que es poden crear jocs que tinguin relacions molt directes amb els sons que es reprodueixen durant l'execució.

8.9. Interfície

En aquest apartat s'ensenyarà el resultat final de la interfície gràfica, mostrant amb detall com l'usuari pot interactuar amb cadascuna de les funcionalitats explicades anteriorment. Tot i així abans és important recalcar que el disseny de tots els elements de la interfície s'han dissenyat tenint en compte que aquest motor va dirigit a un públic sense coneixements tècnics i per tant s'ha intentat crear un entorn en el qual el desenvolupador pugui estar còmode i pugui entendre el funcionament dels mecanismes implementats de forma senzilla i accessible.

Primerament, en la següent figura es pot veure la perspectiva general del motor, en el qual destaca el viewer de OpenGL al centre de la interfície envoltat pel conjunt d'element que permeten la interacció amb totes les funcionalitats. A més, si es compara amb la Figura 14, vista en apartats anteriors, es pot veure l'evolució que ha tingut des de l'inici del projecte.

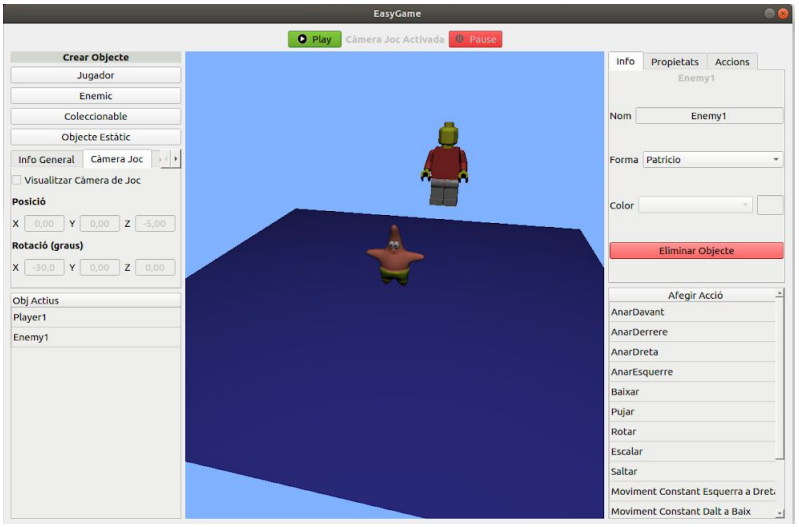

Figura 24. Visió general de la interfície.

Per altra banda, els elements relacionats amb la gestió dels *GameObjects* són els que es mostren en les següents fotografies. En la figura 25 es pot veure que l'usuari només necessita prémer un botó per crear un objecte, i en següents dues fotos es mostra com es poden canviar les seves propietats (model, escala, nom...).

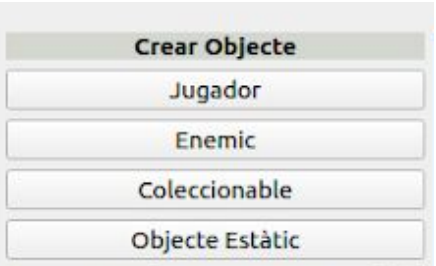

Figura 25. Botons de creació.

Figura 26. Informació general.

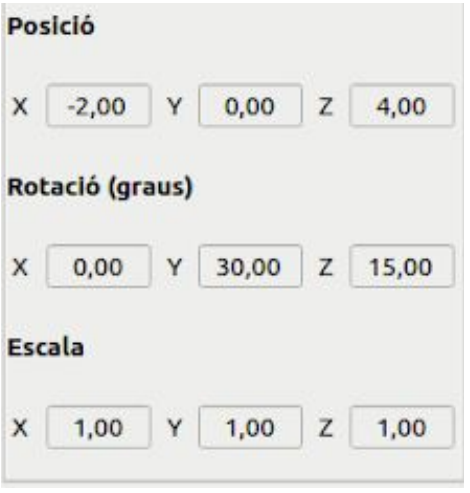

Figura 27. Propietats geomètriques.

A continuació, quan l'usuari vol seleccionar un dels objectes que ha generat, només ha de fer *click* sobre l'element de llista de la figura 28 i podrà modificar les seves propietats o eliminar l'objecte amb el botó vermell que apareix en la figura 26. A més, un cop té un objecte seleccionat, pot afegir les accions que desitgi a aquest mitjançant la llista d'accions que es pot observar a les figures 29 i 30. En aquestes dues últimes fotos es mostra un exemple entre les diferents accions que s'ofereixen segons el tipus d'objecte que està seleccionat, en aquest cas són les accions disponibles per un enemic i un col·leccionable respectivament.

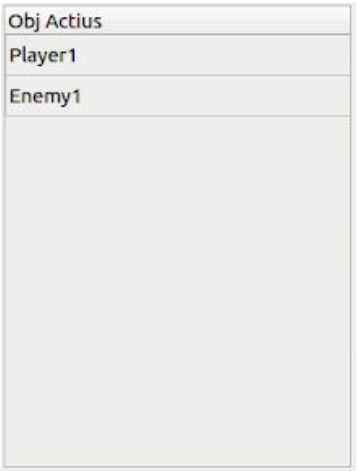

Figura 28. Llista d'objectes.

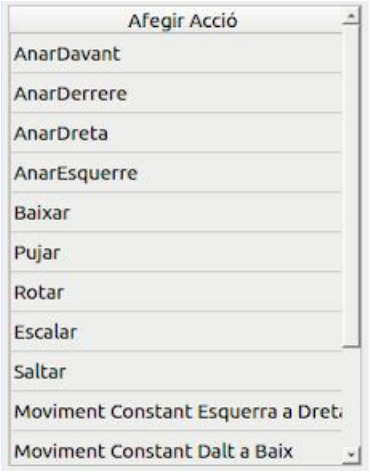

Figura 29. Accions d'enemic.

Figura 30. Accions d'item.

Després d'això, per activar i desactivar la simulació del joc en temps real es pot fer a partir dels botons que es veuen a continuació. Quan s'executa el videojoc el text entre els botons es torna més fosc per tal d'indicar que la càmera del joc està activada i així aclarir a l'usuari quina de les 2 càmeres s'està veient.

Figura 31. Botons de control de la simulació del joc.

Tot seguit es pot veure com l'usuari pot canviar la càmera del joc si es vol que aquesta sigui fixa. L'únic que ha de fer és seleccionar el *checkbox*, el qual si està actiu fa que es mostri en el viewer la càmera *in-game*, i modificar els paràmetres de posició i rotació al seu gust. Per tal de simplificar el funcionament, si el *checkbox* no està activat, l'usuari no pot modificar les propietats de la càmera.

Figura 32. Càmera de joc.

Entrant més en detall en el sistema d'accions, en la figura 33 es pot observar la llista d'accions actives que té en un moment concret un objecte, les quals es poden seleccionar i posteriorment eliminar o modificar seleccionant els botons destinats a complir aquestes funcionalitats. Quan l'usuari crea una nova acció o decideix modificar-ne una, apareixen a la interfície les propietats de l'acció en qüestió de la manera que es mostra en les figures 34, 35 i 36.

Figura 33. Llista d'accions.

Les següents 3 imatges permeten observar que cada tipus d'acció té unes característiques diferents que pot especificar el desenvolupador, de les quals algunes d'elles tenen petites anotacions explicatives per tal que l'usuari entengui com s'utilitzen. A més, en les que s'ha considerat adequat, existeix la possibilitat d'escollir un efecte de so.

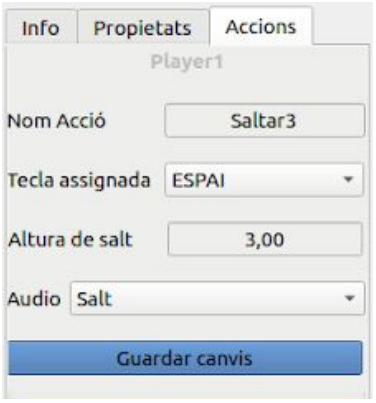

Figura 34. Saltar.

Figura 35. Atacar Enemic.

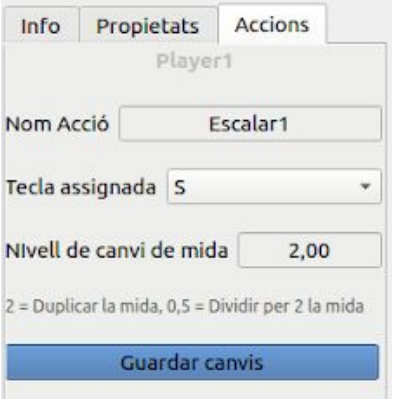

Figura 36. Canvi d'escala.

Finalment, l'última part de la interfície que queda per mostrar és aquella en la qual l'usuari pot definir les condicions de victòria i derrota del joc. A les figures 37 i 38 es poden visualitzar, respectivament, com es pot especificar quan es guanya i quan es perd, seleccionant quina condició es vol que es compleixi i concretant els valors que la defineixen.

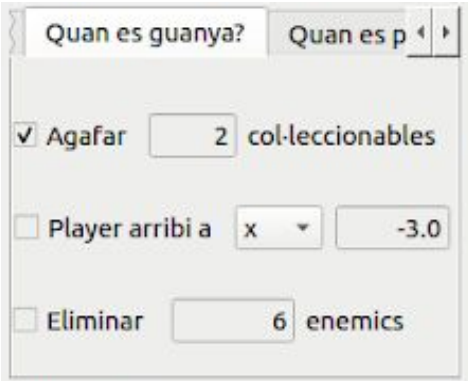

Figura 37. Condicions de victòria.

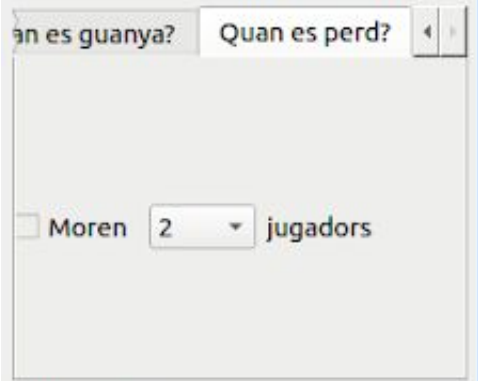

Figura 38. Condicions de derrota.

9. Conclusions

En aquest apartat procediré a explicar les conclusions sobre aquest projecte, les meves valoracions personals i algunes possibles ampliacions futures.

Primerament, la conclusió més important que es pot extreure és que el projecte ha seguit un desenvolupament correcte i sense imprevistos importants durant tot el període de treball, assolint així tots els objectius marcats a l'inici d'aquest. Tot i que hi ha hagut algunes variacions respecte a la planificació prevista, es pot considerar que les modificacions fetes han estat bones decisions que han ajudat al fet que el rendiment del desenvolupament augmentés notablement, i per tant s'hagin complert tots els objectius a curt termini de forma satisfactòria.

Seguidament, una altra conclusió és que desenvolupar aquest motor m'ha fet assolir molts coneixements nous i millorar en aspectes com la planificació i la programació. Gràcies a aquest treball he après com plantejar i estructurar un nou projecte, analitzant les possibilitats existents al món i buscant com pot encaixar i quines opcions innovadores ofereix. A més m'ha servit per descobrir com es fa una anàlisi econòmic previ per comprovar la seva viabilitat real. Entrant més en la matèria que s'ha tractat, el fet d'analitzar els aspectes generals dels videojocs i aprendre les característiques que s'utilitzen per crear-los, em pot ser molt útil en un futur dins la indústria del videojoc.

A continuació, passaré a validar un per un si els objectius marcats a l'inici s'han assolit, valorant així el resultat final del projecte.

9.1. Validació dels objectius

El primer objectiu que es va fixar va ser construir un *game engine* que pugui utilitzar una persona que no sap programar ni coneix els conceptes tècnics de la construcció d'un videojoc. Aquest objectiu s'ha assolit amb èxit, ja que el motor ofereix unes funcionalitats ben adaptades al tipus d'usuari al qual va dirigit, simplificant molt la feina que ha de fer. El cas més clar es pot veure en el sistema d'accions, el qual en un motor gràfic més genèric requeriria tasques de programació i en el cas d'aquest projecte l'usuari només ha d'afegir l'acció que desitgi manualment. A més les diverses proves que s'han anat fent amb gent del meu entorn han tingut bons resultats, aconseguint crear algunes dinàmiques de joc senzilles sense gaire instrucció per la meva part.

El següent objectiu marcat era aconseguir una eina que sigui suficientment flexible perquè una persona, segons el seu grau de dedicació, pugui fer un joc més o menys complicat. Aquest objectiu també s'ha assolit amb èxit, ja que les diferents funcionalitats creades són simples però alhora són ben combinables i adaptables,

permetent que amb cert esforç es puguin crear jocs més complexes. D'aquesta manera l'usuari pot crear les entitats que vulgui, modificant-les amb el conjunt de models disponibles i les seves propietats geomètriques, amb llibertat de moviments i amb diferents opcions d'interacció entre elles.

El primer subobjectiu proposat és que pugui arribar a ser útil de cara al sistema educatiu. Aquest objectiu també s'ha assolit, ja que amb ajuda i instruccions prèvies, un nen pot utilitzar aquest motor per crear jocs molt senzills, i d'aquesta manera jugar amb les peces que s'ofereixen per crear els seus mons virtuals.

Finalment, l'última cosa marcada va ser crear una eina més per desenvolupar videojocs i expandir més el nombre de videojocs i el volum de la indústria. Aquest objectiu s'ha aconseguit perquè el *game engine* implementat permet crear jocs de diferents tipus, tan *runners* en els quals l'objectiu és arribar a un cert punt com jocs de plataformes en els quals s'han d'obtenir un cert nombre d'objectes.

En conclusió, els objectius fixats a l'inici s'han assolit correctament creant un motor suficientment simple i accessible per un públic més novell o infantil, però amb la flexibilitat adequada per generar creacions més complexes amb més dedicació.

9.2. Possibles ampliacions

En aquest apartat comentaré algunes possibles millores del sistema i ampliacions que es podrien afegir, les quals no s'han pogut implementar perquè el temps per fer el treball és limitat.

Primerament, el primer que afegiria serien més models pels objectes i més opcions d'àudio, ja que aquestes actualment són limitades i seria interessant oferir un ventall molt més ampli de personalització del joc. A més, el fet d'afegir més accions disponibles als objectes ampliaria molt el ventall de jocs que pot crear un usuari, com per exemple disparar o tornar-se invisible o immune quan agafi un col·leccionable.

Per altra banda, una millora possible seria permetre a l'usuari modificar les propietats geomètriques dels objectes amb el ratolí directament, és a dir, rotar o traslladar un personatge només movent el ratolí amb un botó premut. Actualment només pot variar aquests atributs de forma manual escrivint el valor numèric que vol assignar i aquesta opció li aportaria comoditat a l'hora de muntar l'escena.

Seguidament una altra funcionalitat que es va plantejar a la fase de disseny del projecte era implementar un sistema de físiques, en el qual els objectes apliquen i reaccionen a forces, com la de la gravetat per exemple. Es va descartar perquè és una tasca força complexa i no es va considerar que hi hagués suficient temps per implementar-la, alhora que no és tan imprescindible com les altres funcionalitats creades.

Finalment altres possibles addicions serien opcions per afegir textos, com per exemple cartells amb indicacions, més possibilitats de càmera com la primera persona o opcions de gestió de memòria del projecte, per tal que l'usuari pogués guardar i carregar les seves creacions.

Així doncs, totes aquestes possibilitats es podrien implementar si hi hagués més temps disponible, però tot i així el resultat final ofereix tots els aspectes més bàsics d'un joc i suficients per al tipus d'usuari al qual va dirigit el motor.

10. Bibliografia

1. «Video Games Could Be a $300 Billion Industry by 2025 (Report) – Variety». [En línia]. Disponible a: https://variety.com/2019/gaming/news/video-games-300-billion-industry-2025-report-1203202672/. [Consulta: 21-feb-2020].

2. U. Technologies, «Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations». [En línia]. Disponible a: https://unity.com/. [Consulta: 22-feb-2020].

3.«Unreal Engine | The most powerful real-time 3D creation platform». [En línia]. Disponible a: https://www.unrealengine.com/en-US/. [Consulta: 22-feb-2020].

4.« Què és Scratch?». [En línia]. Disponible a: https://www.scratchcatala.com/que-es-scratch/. [Consulta: 22-feb-2020].

5.«Clickteam - Multimedia Fusion 2». [En línia]. Disponible a: https://www.clickteam.com/multimedia-fusion-2. [Consulta: 23-feb-2020].

6. «Home • GameSalad», GameSalad. [En línia]. Disponible a: https://gamesalad.com/. [Consulta: 22-feb-2020].

7. «¿En qué consiste la metodología Kanban y cómo utilizarla?», APD España, 30-gen-2019. [En línia]. Disponible a: https://www.apd.es/metodologia-kanban/. [Consulta: 23-feb-2020].

8. T. Q. Company, «Qt | Cross-platform software development for embedded & desktop». [En línia]. Disponible a: https://www.qt.io. [Consulta: 23-feb-2020].

9. «OpenGL - The Industry Standard for High Performance Graphics». [En línia]. Disponible a: https://www.opengl.org/. [Consulta: 23-feb-2020].

13. «A text editor for the 21st Century», Atom. [En línia]. Disponible a: https://atom.io/. [Consulta: 23-feb-2020].

11. «Build software better, together», GitHub. [En línia]. Disponible a: https://github.com. [Consulta: 23-feb-2020].

12. «Trello». [En línia]. Disponible a: https://trello.com. [Consulta: 23-feb-2020].

13. «Indeed: Job Search». [En línia]. Disponible a: https://www.indeed.es/ [Consulta: 06-mar-2020].

14. Jason Gregory, *Game Engine Architecture*, Third Edition.

15. «Ray tracing (graphics) - Wikipedia». [En línia]. Disponible a: https://en.wikipedia.org/wiki/Ray_tracing_(graphics). [Consulta: 21-març-2020].

16. «Functions of Game Engines - Wordpress.com». [En línia]. Disponible a: https://jahmelcoleman.wordpress.com/games-development/functions-of-game-engines [Consulta: 21-març-2020].

17. «Control». [En línia]. Disponible a: https://www.remedygames.com/games/control/. [Consulta: 19-maig-2020].

18. «Geometry Dash». [En línia]. Disponible a: https://geometrydash.io/. [Consulta: 22-febrer-2020].

19. «Ángulos de Euler, Euler angles - qwe.wiki». [En línia]. Disponible a: https://es.qwe.wiki/wiki/Euler_angles. [Consulta: 4-maig-2020].

20. «Modelo de reflexión de Phong - Wikipedia». [En línia]. Disponible a: https://es.wikipedia.org/wiki/Modelo_de_reflexi%C3%B3n_de_Phong. [Consulta: 9-maig-2020].

21. «QKeyEvent Class - Qt GUI 5.14.2». [En línia]. Disponible a: https://doc.qt.io/qt-5/qkeyevent.html. [Consulta: 12-maig-2020].

22. «QTimer Class - Qt Core 5.14.2». [En línia]. Disponible a: https://doc.qt.io/qt-5/qtimer.html. [Consulta: 14-maig-2020].

11. Glossari

- **Clipping**: Mètode per habilitar o deshabilitar selectivament operacions de renderitzat dins una regió definida. S'utilitza per seleccionar quins píxels de l'escena s'han de mostrar per pantalla segons el posicionament de la càmera i així reduir el temps de càlcul.

- **Debugging**: Procés per identificar i arreglar errors de programació.

- **GameObjects:** Objectes fonamentals de l'escena que representen els personatges i els altres elements de l'escenari.

- **Pipeline gràfic:** Conjunt de processos que converteixen un conjunt de dades que formen una escena 3D en una imatge 2D que s'imprimeix per pantalla.

- **Vertex Shader:** Fase del pipeline gràfic que s'encarrega del processament dels vèrtexs dels objectes de l'escena.

- **Viewer**: Aplicació que permet mostrar els gràfics d'una escena 3D.

- **Widget**: Component visual d'una aplicació amb la qual l'usuari interacciona i que en el seu conjunt permeten l'ús de les diferents funcionalitats del programa.

- ***Scripts:*** Llenguatge de programació dedicat a l'automatització de rutines i aplicacions.

- ***Machine Learning:*** Subcamp de la intel·ligència artificial que té com a objectiu desenvolupar tècniques que permetin aprendre a les màquines.

- **Shader:** Programa informàtic dedicat a realitzar càlculs gràfics amb la possibilitat de compilar-se independentment.