Universitat
Autònoma
de Barcelona

4782: DISSENY I IMPLEMENTACIÓ D'UN VIDEOJOC 3D

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
David Rios Puig
i dirigit per
Felipe Lumbreras Ruiz
Bellaterra, 17 de setembre de 2012

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Felipe Lumbreras Ruiz

Professor de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en David Rios Puig

I per tal que consti firma la present.

Signat:

Bellaterra, 17 de setembre de 2012

Index

Capítol 1

Introducció

Aquest projecte fi de carrera d'enginyeria en informàtica tracta sobre el disseny i la implementació d'un videojoc 3D. En aquest primer capítol de la memòria podem trobar les motivacions que m'han portat a realitzar aquest projecte fi de carrera, els objectius que m'he proposat i un breu resum dels capítols que componen aquesta memòria.

1.1 Motivacions

Des de petit els videojocs han estat una de les meves passions. Més o menys a meitat dels anys noranta vaig rebre la meva primera videoconsola, una NES[1], amb uns quants jocs. Recordo llargues sessions jugant a clàssics com ara el *Super Mario Bros 3* o *The Legend of Zelda*, grans títols que van convertir els videojocs en una de les meves grans aficions.

Els videojocs van deixar de ser només una afició per a mi fa un parell d'anys, al realitzar l'assignatura "Gràfics per computador", on vaig adonar-me que podia realitzar videojocs per mi mateix. A partir d'aquest punt vaig començar realitzant primerament un clon del videojoc *Space Invaders* per aprofundir després en el desenvolupament de videojocs realitzant un parell de jocs de manera ja més seriosa que han estat publicats a XBLIG[2]. Durant aquest temps també he anat observant diferents ofertes de feina i realitzant alguna entrevista de cara a la meva incorporació al món laboral i en tots els casos m'he adonat que per entrar en aquest món es valoren molt els projectes ja realitzats, idealment en altres empreses, però també els projectes realitzats a nivell individual. Fins al moment els meus dos jocs realitzats per a XBLIG han estat la meva principal carta de presentació però evidentment dos projectes no són suficients. A més, els dos projectes han estat realitzats en C#[3], que tot i que actualment està valorat gràcies a que certes plataformes i motors de joc en fan ús, segueix sent menys valorat que C++[4], que es pot utilitzar en pràcticament qualsevol plataforma i que acostuma a

1 Nintendo Entretainment System, consola de vuit bits creada per Nintendo l'any 1983.
2 Xbox Live Indie Games, plataforma de Microsoft per a la distribució digital de videojocs creats principalment per estudis independents o particulars per a la consola Xbox360.
3 Llenguatge de programació orientat a objectes dissenyat per Microsoft l'any 2001.
4 Llenguatge de programació orientat a objectes dissenyat per Bjarne Stroustrup l'any 1983.

ser el llenguatge utilitzat en projectes de grans pressuposts. Destacar també que cap dels dos videojocs es un videojoc purament en 3D, de manera que en el meu currículum apareix un buit a cobrir.

És per tot això que he decidit realitzar aquest projecte fi de carrera. Realitzant un videojoc en 3D cobreixo un dels buits que apareixen en el meu currículum. A més, com veurem més endavant, afegirem un requeriment no funcional al projecte, obligant-me a que el joc sigui escrit en el llenguatge C++. D'aquesta manera, amb aquest projecte cobriré un altre dels buits del currículum, demostrant domini en el llenguatge C++ amb un projecte complet.

1.2 Objectius

L'objectiu principal d'aquest projecte fi de carrera és dissenyar i implementar un videojoc en 3D, des de la seva fase inicial, amb el document de disseny, fins a la seva implementació final. Aquest objectiu global el podem dividir en els següents objectius específics:

- Avaluar i escollir entre les diferents alternatives d'eines i llibreries disponibles per a realitzar videojocs, tenint en compte els requisits funcionals i no funcionals del projecte.
- Crear el document de disseny (la bíblia del videojoc) on s'especificarà els objectius del joc, la mecànica que ens permetrà assolir aquests objectius, com seran les diferents pantalles i mapes, etc.
- Si s'escau, dissenyar i implementar un motor de joc bàsic que permeti interconnectar les diferents llibreries en una mateixa aplicació que servirà de base per al joc. Aquest objectiu només s'haurà d'acomplir en cas de no escollir un motor complet sobre el qual realitzar el joc.
- Implementar el document de disseny mitjançant una metodologia iterativa.

A més d'aquests objectius, durant la realització d'aquest projecte fi de carrera han sorgit nous objectius gràcies a diferents oportunitats que han anat apareixent. Veurem el

perquè d'aquests nous objectius al llarg de la memòria amb més profunditat, però els citem ja que finalment han estat objectius del projecte:

- Dissenyar i implementar un motor multiplataforma fàcilment utilitzable i ampliable.
- Analitzar com Unity[5] pot ajudar a prototipar i testejar per estalviar temps de desenvolupament.

Cal destacar també un no-objectiu del projecte. Tot videojoc consta d'una part gràfica, ja sigui 2D o 3D, d'una part de programació i tecnologia i d'una part musical i d'efectes sonors. No obstant, en cap moment es menciona com a objectiu del projecte fi de carrera crear els gràfics del videojoc, capturar efectes de so o composar la música del joc. Com he dit la motivació principal del projecte és millorar el meu propi currículum, de perfil eminentment tècnic. Per tant, queda fora del projecte i dels meus interessos personals adquirir aquestes competències. Per poder mostrar els resultats del projecte utilitzaré recursos de tercers, bé gratuïts o bé de pagament (sempre posseint els drets sobre ells) i explicaré el procés emprat per a utilitzar-los.

1.3 Organització de la memòria

La memòria s'organitza en sis capítols, el contingut dels quals es resumeix a continuació:

Capítol 1 – Introducció

Primer capítol de la memòria, dedicat a la introducció del projecte. En aquest capítol podem veure la motivació per a la realització del projecte, els objectius i l'organització de la memòria.

5 Motor de joc multi-plataforma creat per Unity technologies

Capítol 2 – Document de disseny

Segon capítol de la memòria, dedicat al document de disseny. En aquest capítol quedarà definit com serà el videojoc i serà la base per prendre decisions sobre quin motor o quines llibreries utilitzar, la distribució del temps, etc.

Capítol 3 – Anàlisis / Estat de l'art

Tercer capítol de la memòria, dedicat a l'estat de l'art. En aquest capítol es mostraran les diferents eines, llibreries i motors de joc disponibles per a crear un videojoc, s'analitzaran els pro i els contra de cada solució i s'escollirà la més adient pel projecte.

Capítol 4 – Proposta de solució

Quart capítol de la memòria, dedicat al desenvolupament de la solució per a resoldre el problema. En aquest capítol veurem una primera aproximació al que serà l'arquitectura del motor del joc i del joc en sí mateix.

Capítol 5 – Implementació

Cinquè capítol de la memòria, dedicat a la implementació de la solució escollida. En aquest capítol s'explicarà la metodologia utilitzada pel desenvolupament, l'arquitectura de la solució en detall i altres detalls referents al desenvolupament del videojoc.

Capítol 6 – Resultats

Sisè capítol de la memòria, dedicat a la presentació dels resultats. En aquest capítol es mostrarà el resultat final del videojoc.

Capítol 7 – Conclusions i treball futur

Setè capítol de la memòria, dedicat a les conclusions del projecte. En aquest capítol es mostraran els objectius assolits, els no assolits i el possible treball futur a realitzar.

Capítol 2

Document de disseny

En aquest segon capítol dedicat al document de disseny s'especificarà com ha de ser el joc a realitzar. Pot sobtar trobar un document de disseny, que al cap i a la fi és completament lliure i poc lligat a l'àmbit tecnològic a primera vista, en una memòria de projecte fi de carrera però he cregut indispensable la seva inclusió en la mateixa com a un capítol propi ja que el document de disseny pot afectar directament a les següents decisions. L'exemple més evident és que no escollirem les mateixes eines per realitzar un joc en dues dimensions que en el cas d'un joc en tres dimensions. No obstant, la importància del document de disseny no es queda només aquí, si no que segons la pròpia temàtica del joc ens pot obligar a prendre altres decisions.

2.1 Vista general

2.1.1 Concepte de joc

En aquest videojoc encarnarem el paper d'un individu mort (representat pel seu esquelet) que es troba realitzant el camí necessari per poder descansar en pau. Aquest camí consistirà en superar una sèrie de laberints fins a arribar a la seva salvació. Per tant, tota l'acció del joc es concentrarà en aquests laberints que el jugador haurà de resoldre fent ús de la seva habilitat i el seu enginy per derrotar els enemics i solucionar els trencaclosques proposats.

2.1.2 Gènere del joc

El gènere del joc podria correspondre tant al gènere d'aventures, com al gènere dels trencaclosques o al gènere d'acció en tercera persona. No obstant, el definirem com a joc d'aventures ja que té pinzellades del gènere d'acció en alguns moments i pinzellades del gènere de trencaclosques en alguns altres, però cap dels dos gèneres és predominant.

2.1.3 Propòsit i públic objectiu

El propòsit d'aquest joc és la realització del projecte fi de carrera d'enginyeria informàtica. No obstant, es pretén que aquest projecte pugui convertir-se en un joc comercial un cop estigui acabat. Destacar que sobre la base d'aquest projecte es va realitzar una demostració per a un esdeveniment per a desenvolupadors realitzat a Saragossa i el joc va resultar premiat amb el primer premi del certamen.

Pel que fa al públic objectiu, el joc està enfocat a un públic molt ampli, ja que no requereix de grans hores d'aprenentatge per poder gaudir del joc i la seva estètica és apta per tots els públics, al ser acolorida i sense grans mostres de violència.

2.1.4 Amplitud del joc

L'amplitud d'un videojoc amb intensions comercials i d'un videojoc per a un projecte final de carrera és molt diferent. Tenint en compte això, ens proposarem crear un joc amb els següents elements, que en un futur seran fàcilment ampliables si es decideix seguir amb el projecte:

- Dues pantalles, incloent la pantalla inicial i un nivell complet.
- Cinc personatges no controlables, incloent guardians del laberint i enemics.
- Personatge controlable amb totes les seves accions correctament definides.

2.2 Jugabilitat i mecàniques de joc

2.2.1 Jugabilitat

L'objectiu del joc és que l'esquelet pugui descansar en pau. Per aconseguir-ho, haurem de superar tots els nivells fins a arribar al lloc de repòs. A l'inici de cada nivell (és a dir, de cada laberint) el jugador apareixerà caient del cel. Un cop a terra, el jugador podrà rebre les instruccions o pistes d'un personatge no jugador situat a l'inici del nivell i començarà a recorre el laberint. L'objectiu de cada nivell serà trobar el final d'aquest, que estarà custodiat per un altre personatge no jugador que ens enviarà a l'inici del següent nivell. A mesura que anem

recorrent el laberint ens trobarem amb objectes que ens seran útils durant la nostra aventura, enemics als quals derrotar o personatges als que podem ajudar.

2.2.2 Mecàniques

El joc estarà subjecte a la física. Tots els personatges i objectes estaran subjectes a ella. Els enemics eliminats no desapareixeran, així que romandran al terra de manera que serà possible col·lisionar amb ells.

2.3 Mapes del joc

En aquest apartat veurem els dos escenaris d'aquesta primera versió del joc, la pantalla d'inici i el primer dels laberints.

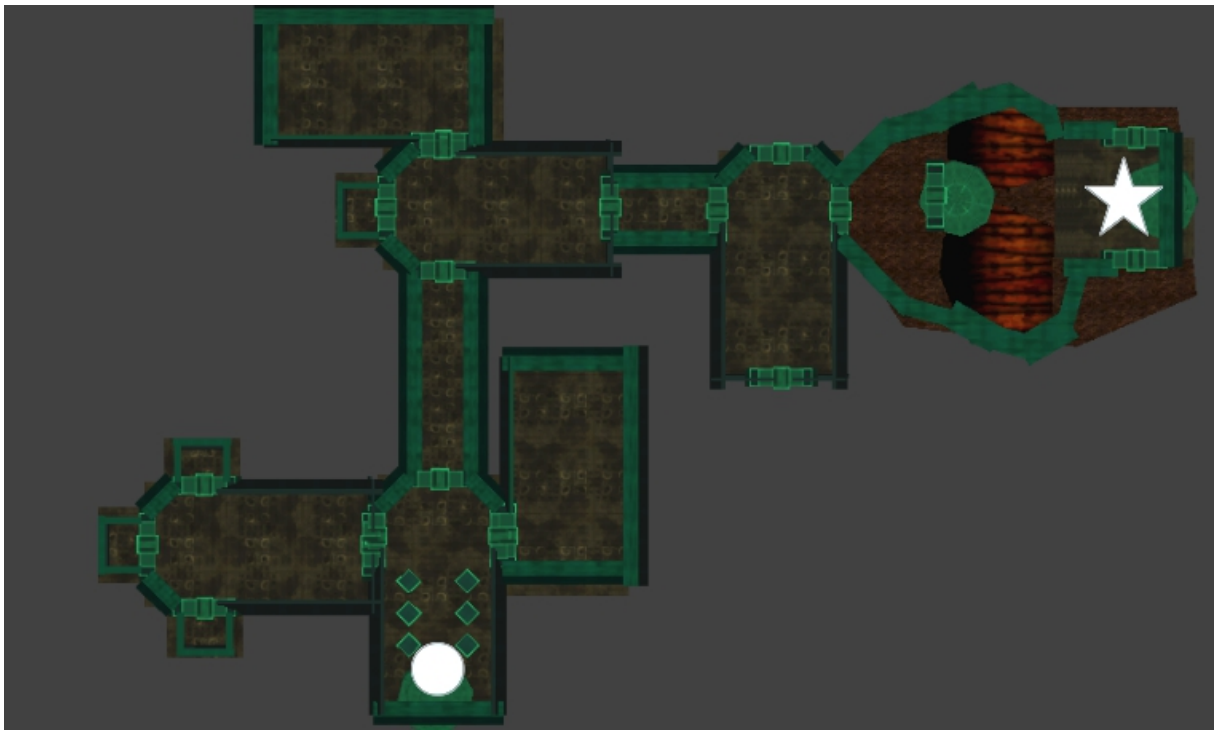
2.3.1 Pantalla d'inici

En iniciar el nostre joc ens apareixerà un món 3D on podrem escollir quina opció volem realitzar, si sortir del joc, començar una partida o, en un futur, escollir directament un dels mapes a jugar.

Imatge de la pantalla d'inici, on uns cartells ens indicaran el camí

2.3.2 Primer mapa

El disseny del primer mapa constarà d'un seguit d'estàncies que haurem d'anar superant per poder arribar a la part final del nivell. Així, començarem en l'estància marcada amb un cercle i el nostre objectiu serà arribar a l'estància marcada amb una estrella. El jugador, no obstant, no sabrà a quina estància s'ha de dirigir, de manera que segurament no seguirà el camí òptim.



Mapa del primer nivell del joc

2.4 Personatges del joc

En aquest apartat trobarem un llistat dels personatges que podran aparèixer al joc. En concret, podrem trobar esquelets, gremlins i trolls, amb diferents textures i colors. Tots els personatges tindran l'animació de caminar, de corre, de està en repòs i d'atacar. L'únic personatge controlable en aquesta primera versió del joc serà l'esquelet.

L'esquelet serà l'únic personatge controlat pel jugador. No obstant això, també apareixeran dins el joc tant esquelets amics com esquelets enemics. El seu nivell de vida, velocitat i força està equilibrat.

Els gremlins seran un dels altres tipus d'enemics que apareixeran al joc. En aquest cas seran enemics de poca resistència però més velocitat que no pas el nostre personatge.

L'últim dels enemics que apareixerà al joc seran els trolls. Enemics molt robustos, aguanten molt bé els cops i provoquen una gran quantitat de dany amb els seus atacs. No obstant, són enemics lents i fàcilment esquivables.

Capítol 3

Estat de l'art

En aquest tercer capítol dedicat al l'estat de l'art s'analitzaran les diferents eines i llibreries que podem utilitzar per crear un videojoc, els pros i els contres d'optar per cadascuna d'elles aixi com el procés que ens ha portat a escollir les eines que finalment s'han emprat per realitzar aquest projecte fi de carrera.

3.1 Motors de joc

En la creació de tot videojoc utilitzarem, bé de tercers o bé creades per nosaltres mateixos, llibreries que s'encarregaran de la part gràfica, de la part física, de gestionar l'entrada de l'usuari, de gestionar els elements sonors, etc. Podem definir el motor del joc com el conjunt unificat de totes aquestes llibreries comunes a tots els videojocs i que ens serveixen de base per realitzar el desenvolupament del joc. Per tant, la primera elecció d'un projecte serà decidir si escollim utilitzar un motor de joc complet (on ja s'inclouen tots els subsistemes) o bé optar per crear les nostres pròpies llibreries (o utilitzar llibreries de tercers) i unificar-les nosaltres mateixos.

3.1.1 Unreal Development Kit (UDK)

El primer dels motors de joc analitzat ha estat el Unreal Development Kit[UDK] creat per Epic Games[6]. Es tracta d'un dels motors més potents del mercat, incorporant el seu propi editor WYSIWYG[7], utilitzant la llibreria de física PhysX de Nvidia (una de les més emprades en la industria), a més de llibreries pròpies amb algoritmes d'intel·ligència artificial, sistemes de partícules, etc, per fer més fàcil el desenvolupament.

No obstant, compta amb un seguit de limitacions que han fet descartar-ne el seu us. La més restrictiva és el seu llenguatge de programació, UnrealScript, fet que contradiu el

6 Epic Games és una empresa nord americana creadora entre d'altres de la saga Gears of War (Xbox360) o Unreal Tournament

7 What you see is what you get, acrònim utilitzat per referir-se a editors visuals, on els resultats del que veiem són els resultats que obtindrem

requeriment no funcional que havíem definit a l'inici del projecte. A més, la seva llicència no és de les més accessibles, ja que si bé compta amb una llicència gratuïta per a estudiants, si volem realitzar projectes de forma professional haurem d'abonar un cost de 2500$ per equip, a més de 99$ per títol publicat i un 25% de les vendes realitzades a partir dels 5000$ d'ingressos. També s'ha de destacar que UDK només permet desenvolupar en plataformes Windows i per a plataformes Windows, de manera que si volem realitzar un desenvolupament multi-plataforma no podem comptar amb aquesta solució.

3.1.2 Unigine

Unigine[UGINE], creat per Unigine corp, és segurament un motor de jocs poc popular com a motor de joc però molt popular com a eina de per comprovar el rendiment gràfic de les targes gràfiques actuals. Aquest fet provoca que Unigine sigui una referència visual en el camp dels videojocs, incorporant les últimes tècniques en gràfics 3D tant pel que fa a DirectX com en el cas de OpenGL, suportant les últimes versions de les dues llibreries gràfiques. A més d'un apartat gràfic de primer nivell, Unigine incorpora també la gestió de la física, gestió de só, entrada, etc. Al seu favor destacar també la possibilitat de programar en C++ i el suport multi-plataforma (Windows, Linux, Mac OS X, Playstation3 i fins i tot versions per a Android i iOS). Per tot el comentat anteriorment Unigine semblaria la solució ideal per a la realització del projecte. No obstant això, Unigine compta amb un gran handicap, el seu preu. Les llicències de Unigine són molt costoses, sobre els 30.000$, fet que fa inassumible treballar amb aquesta eina, no només en un projecte fi de carrera si no també en un desenvolupament independent de baix cost.

3.1.3 Unity3D

L'últim però no menys important dels motors analitzats ha estat Unity[U3D], creat per Unity technologies, un equip que es dedica única i exclusivament a la creació d'aquest motor. El que més crida l'atenció d'aquest motor és la senzillesa d'ús, la claredat dels tutorials disponibles a la pròpia web i la gran asset store, una botiga online incorporada dins el propi motor de joc on podrem comprar sons, scripts, recursos gràfics, etc, que ens facilitaran la

creació del nostre videojoc. Tot això converteix a Unity amb el motor amb la millor corba d'aprenentatge, ja que podrem realitzar els nostres primers prototips de manera molt senzilla i barata (Unity disposa de llicència d'ús gratuïta a canvi d'afegir una pantalla de carrega amb el seu logotip). No obstant, Unity comparteix un dels inconvenients ja vistos en el cas d'Unreal Development Kit, el llenguatge de programació que hem d'utilitzar. Si bé en el cas de Unity el llenguatge escollit és C#, molt més popular que UnrealScript, no és el llenguatge de programació que havíem decidit a l'inici del projecte i, per tant, no podem escollir aquest motor de joc. S'ha de destacar, però, que si aquest requisit no existís, Unity seria sens dubte la solució escollida per a realitzar el videojoc.

3.1.4 Decisió final

Un cop analitzats tres dels motors de jocs més emprats avui en dia s'ha arribat a la conclusió que s'haurà d'optar per la segona opció, és a dir, crear el nostre propi motor de joc, ja que cap dels tres compleix tots els requeriments, bé per tenir un preu prohibitiu (cas del motor Unigine) o bé per utilitzar un llenguatge de programació diferent a C++, llenguatge decidit prèviament per tal de millorar el meu currículum (cas del motor UDK i del motor Unity3D).

3.2 Motors gràfics

Un cop aparcada la idea d'escollir un motor de joc complet el següent pas consistirà en analitzar quines llibreries tenim disponibles, amb quines llicencies, quines limitacions tenen i com funcionen combinades entre elles per tal de crear el nostre propi motor. Per fer-ho, s'ha optat per analitzar primerament els diferents motors gràfics 3D i, a partir d'aquí, anar escollint la resta de peces a utilitzar.

3.2.1 Irrlicht

Irrlicht[IRR] és un dels motors gràfics de codi obert (llicència basada en zlib/libpng) més utilitzats conjuntament amb Ogre3D, analitzat en el següent subapartat. Cal destacar els

projectes desenvolupats com a complement per part de la comunitat, com ara irrKlang [IRRK], un motor de so, o bé irrEdit[IRRE], un editor visual per facilitar la creació del videojoc. Dos punts més a favor d'Irrlicht són les seves capacitats multi-plataformes (disponible inicialment per a Windows, Mac OS X i Linux, però portable a altres plataformes) i el fet d'estar programat en C++, fet que fa que podem emprar aquest llenguatge (a més de molts altres) per a la creació del nostre videojoc. Com a punt millorable (que no negatiu) destacar els tutorials i la comunitat. S'ha de tenir en compte que el fet de no tenir una empresa 100% dedicada al desenvolupament del producte comporta certs problemes, com ara que el nivell dels tutorials sigui millorable o la necessitat de comptar de manera molt més important amb la comunitat, que no sempre ofereix un suport òptim. No obstant això, Irrlich compleix amb suficiència els requeriments demanats.

3.2.2 Ogre3D

El segon dels motors gràfics analitzats ha estat Ogre3D[O3D], motor que comparteix molts dels punts forts i dels punts dèbils de Irrlicht. Ogre3D també és un motor gràfic de codi obert (llicència MIT), multi-plataforma (millorant el nombre de plataformes respecte a Irrlicht, afegint suport també per a Android i iOS), gratuït i programat en C++. Respecte a Irrlicht, no obstant, ens trobem amb una comunitat més amplia i a primera vista més participativa, on podem trobar també projectes de tercers adaptats a aquest motor, com ara el projecte OgreSpeedTree[OET], dedicat a la vegetació, per citar-ne algun. Destacar també que, si bé no forma part d'Ogre3D, es dóna com a estàndard l'ús conjuntament amb Ogre3D de OOIS[8], una llibreria que s'encarrega de gestionar l'entrada de l'usuari, de manera que tenim multitud d'exemples on podem veure com s'utilitzen conjuntament.

3.2.3 Decisió final

Tenint dos motors gràfics molt similars i donat que els dos motors compleixen tots els requisits demanats, s'ha decidit escollir-ne un per a realitzar aquest projecte fi de carrera. En aquest cas l'escollit ha estat Ogre3D gràcies sobretot a la comunitat més activa que té darrera,

8 Object Oriented Input System, sistema per gestionar l'entrada de l'usuari d'ús molt senzill que forma part dels exemples d'Ogre

les múltiples plataformes suportades i al fet de tenir multitud d'exemples on a Ogre3D s'afegeix OOIS, fet que ens estalviarà preocupar-nos de del sistema d'entrada de l'usuari.

3.3 Motors de físiques

Un cop escollit el motor gràfic del nostre joc cal escollir l'altre gran motor necessari per a la creació d'un videojoc, el motor de físiques. S'ha de destacar que, si bé en el cas del motor gràfic l'hem escollit segons el nostre propi criteri, en aquest cas caldrà veure també com encaixa el nostre motor de físiques amb el nostre motor gràfic, bé per escollir un altre motor de físiques o fins i tot per replantejar la decisió presa anteriorment sobre el motor gràfic.

3.3.1 Bullet Physics

Bullet Physics[BLLT] és un dels tres grans motors de físiques (conjuntament amb Nvidia PhysX, analitzat posteriorment, i Havok) disponibles actualment i l'únic de codi obert, utilitzat tant en videojocs comercials (a destacar superproduccions com Grand Theft Auto IV o Red Dead Redemption) com en pel·lícules de cine (per exemple en Sherlock Holmes o en Bolt, de Walt Disney Animation Studios)[WKBLLT]. Encaixa perfectament en el nostre projecte ja que és una llibreria gratuïta i escrita en C++, a més de tenir una documentació acceptable darrera. Aquests fets demostren la maduresa de la solució, fet que converteix Bullet Physics en una candidata a ser escollida com a la llibreria de físiques a utilitzar.

3.3.2 Nvidia PhysX

Nvidia PhysX[PHSX] és l'estrella indiscutible dels motors de físiques. La seva utilització en grans videojocs com ara Batman: Arkham City, Mafia II o Metro 2033 demostren que ens trobem davant d'una solució d'alt nivell[WKPHSX]. Nvidia PhysX compta a més amb l'afegit de distribuir els binaris per a diferents plataformes, fet que fa que tot i que el codi no sigui obert podem crear els nostres jocs tant per Windows, com per Mac OS X, com per Linux, com per dispositius mòbils (Android i iOS). Destacar també la opció d'utilitzar una targeta de la família Nvidia com a segona targeta gràfica dedicada únicament als càlculs de

físiques, fet que fa possible assolir un nivell de destrucció d'escenaris difícilment possible realitzant tots els càlculs dins el processador principal. Com en el cas de Bullet Physics, Nvidia PhysX compta també amb una API escrita en C++ de manera que ens serà possible utilitzar aquesta llibreria conjuntament amb Ogre3D.

3.3.3 Decisió final

Un cop vistos i analitzats dos dels tres motors de físiques més importants (hem deixat fora de l'anàlisi Havok per ser una solució privativa com en el cas de Nvidia PhysX i no comptar amb experiència en aquesta eina, fet que si comptem en el cas de l'eina de Nvidia) arriba el moment de decidir-se per un dels dos motors. Per fer-ho, s'ha realitzat una escena de provar per comprovar com funcionen les dues llibreries de físiques al ajuntar-les amb la llibreria gràfica escollida anteriorment, Ogre3D.

L'escena de prova consta d'una petita malla simulant unes muntanyes per on deixarem caure unes petites esferes i comprovarem si segueixen el camí esperat. A més, a l'escena afegirem personatges controlables als que llençarem un altre grup d'esferes per comprovar si responen correctament a la col·lisió. També comprovarem que el món físic i el món gràfic encaixin correctament, és a dir, que si tenim una esfera de 1m de radi en el món físic en el món gràfic aquesta esfera estigui a la mateixa posició, tingui la mateixa dimensió i es mogui conjuntament a l'esfera física.

Per realitzar aquesta comprovació s'ha implementat una petita classe Debug Render. Aquesta classe utilitza la potència dels “listeners” d'Ogre3D (classe base a partir de la qual podem executar codi en un moment determinat del dibuixat de l'escena, en el nostre cas una vegada realitzat tot el dibuixat) i les funcionalitats de PhysX i Bullet Physics de retornar-nos una llista de les línies que composen el món físic per pintar-les mitjançant crides a OpenGL sobre la nostre escena gràfica.

En el cas de Bullet Physics hem tingut un petit problema amb la col·lisió esfera – personatge controlable, segurament atribuïble a la poca experiència amb la llibreria en qüestió Per contra, la col·lisió esfera – malla ha funcionat correctament, seguint en tot moment l'esfera la trajectòria esperada.

En el cas de Nvidia Physics ha funcionat correctament tant la col·lisió esfera – personatge controlable com la col·lisió esfera – malla. En aquest cas, a més, m'he trobat més còmode per el coneixement previ que ja tenia d'aquesta llibreria, tant per experiències prèvies personals com per l'experiència adquirida al realitzar les assignatures de programació de videojocs.

Vistos els resultats i tenint en compte la meva experiència personal, el motor de físiques escollit ha estat Nvidia PhysX, ja que s'adapta a les especificacions requerides i a més m'és més fàcil de treballar amb aquesta llibreria que no pas amb Bullet Physics.

3.4 Motor de so

Un cop escollit quin serà el nostre motor gràfic, el nostre motor de físiques i com gestionarem l'entrada de l'usuari queda per escollir el tercer gran pilar d'un videojoc, el motor de so.

3.4.1 Fmod

Fmod[FMOD], creat per Firelight Technologies, és una de les solucions més utilitzades, emprada tant per a la realització de jocs senzills, com ara Plants vs Zombies, com també per a superproduccions com ara Metroid Prime 3, Starcraft II o Diablo 3, per citar alguns exemples[WKFMD]. Fmod és més que una simple llibreria de so, ja que incorpora també un editor de so a més de la pròpia API. Si bé és una llibreria gratuïta per a projectes no comercials, Fmod té el handicap de ser una llibreria de pagament pels projectes comercials, amb un cost de 500$ per plataforma. És a dir, si volem realitzar un joc comercial per Windows, Mac i Linux haurem d'abonar 1500$ per joc.

3.4.2 FreeSLW

FreeSLW, creat per el programador Lukas Heise[FSLW], és una llibreria de so de codi obert (llicència GLPLv3) poc coneguda, creada sobre la llibreria de so OpenAL i distribuïda gratuïtament. Al contrari que en el cas de Fmod, FreeSLW es limita simplement a reproduir sons, no ofereix cap més funcionalitat. No obstant això, el seu funcionament és extremadament senzill, fet que la converteix en la llibreria ideal si no volem grans funcionalitats. A més, s'ha de destacar que la llibreria ha estat programada per a la seva utilització en Windows. Si bé a priori es pot utilitzar també en altres sistemes, en el cas de Mac OS X ens ha donat problemes que han fet impossible la seva utilització en aquesta plataforma.

3.4.3 SFML

SFML[SMFL] és una altre llibreria de codi obert (llicència zlib/libpng) dedicada a la reproducció de so però que, a més, ofereix funcionalitats de xarxa, gràfiques, de creació de finestres, etc. A primera vista pot semblar, per tant, que aquesta llibreria ofereix massa funcionalitats per les necessitats que volem cobrir. No obstant això, el seu disseny modular fa possible utilitzar només la part d'àudio en el nostre projecte. A més, s'ha de destacar la seva claredat en la API i el seu caràcter multi-plataforma, suportant tant Windows com Linux i Mac OS X, fet que fa de la utilització d'aquesta llibreria una tasca realment fàcil.

3.4.4 Decisió final

El nostre projecte fi de carrera, com veurem més endavant, va començar essent programat des d'un entorn Windows. En el seu moment, la solució escollida per a la gestió d'àudio va ser FreeSLW, per la seva facilitat d'ús, la seva distribució en solitari i la seva gratuïtat. No obstant això, la solució emprada finalment per a la realització d'aquest projecte fi de carrera ha estat SFML. Aquest fet és degut al canvi de sistema operatiu sobre el que s'ha realitzat el projecte (de Windows 7 a Mac OS X 10.7) que ha fet que FreeSLW ens donés problemes, fent impossible la seva utilització degut a la manca de suport per a aquesta

plataforma. Per tant, era més òptim treballar amb una solució que suporta directament la plataforma Mac que no pas modificar una llibreria existent per a donar-li funcionalitats.

3.5 Eines a utilitzar

Un cop decidides les llibreries bàsiques a utilitzar per a la realització del projecte queda només decidir quines eines s'utilitzaran per programar el joc, quines per tractar els recursos gràfics de tercers, etc.

3.5.1 Entorns de programació

Pel que fa als entorns de programació s'ha decidit escollir els dos grans entorns de programació actuals per l'experiència prèvia que ja posseïa en la seva utilització. Així, al programar sobre la plataforma Windows s'ha utilitzat Microsoft Visual Studio 2010 i al programar sobre la plataforma Mac OS X s'ha utilitzat Xcode 4.

3.5.2 Programari gràfic

Com hem dit en la introducció d'aquesta memòria, en aquest projecte no es realitzarà la creació dels gràfics ni dels sons del joc, de manera que s'utilitzaran recursos de tercers. No obstant, aquest fet no implica que no necessitem cap tipus de programari gràfic. Els gràfics escollits i mostrats en l'apartat 2 de la memòria ens venen donats en format .FBX, mentre que Ogre3D, eina escollida com a motor gràfic, utilitza gràfics amb format .mesh (a més del seu material i el seu esquelet, si s'escau). Per tant, necessitarem utilitzar alguna eina que transformi arxius en format .FBX a format .mesh. Per sort, existeix un plugin anomenat Ogremax[OMAX] que realitza aquesta funció. Aquest plugin funciona sobre 3DS max, Maya i Softimage/XSI i està disponible de manera gratuïta per projectes no comercials i per 250$ per a projectes comercials. En el nostre cas s'ha optat per utilitzar Ogremax sobre 3DS max per a realitzar les conversions, ja que 3DS max ofereix una llicència per a estudiants.

Capítol 4

Proposta de solució

Un cop decidides les llibreries i eines a utilitzar hem d'analitzar com dissenyarem el nostre joc. A causa de les eleccions realitzades en l'apartat anterior podem diferenciar dues parts del projecte, la creació del motor de joc (on acoblarem tots els subsistemes escollits) i la creació del joc pròpiament dit (on utilitzant el motor de joc creat anteriorment implementarem les funcionalitats descrites al document de disseny).

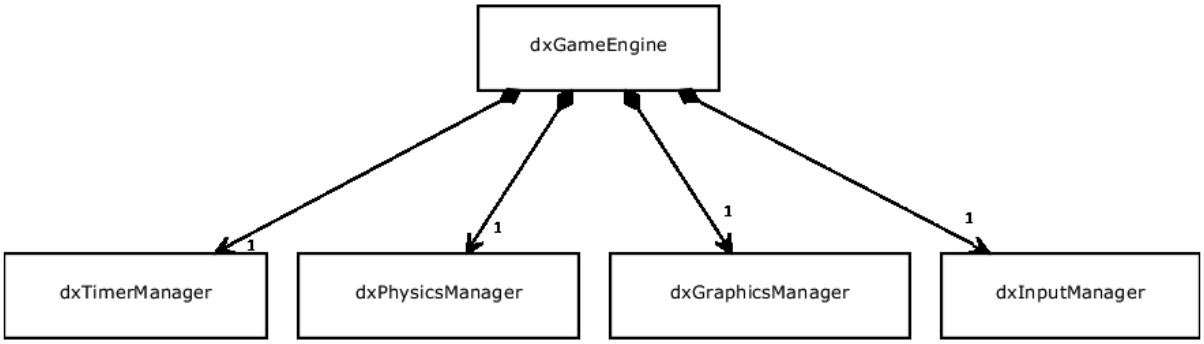
4.1 Disseny del motor de joc

En el disseny del motor del joc s'ha optat per intentar separar les tasques a realitzar en subsistemes el més independents possibles, de manera que sigui possible en un futur canviar, si és necessari, les llibreries escollides per altres llibreries de manera fàcil i sense haver de reescriure tot el motor de nou. És per això que s'ha optat per la creació del següents mànagers, classes encarregades de la gestió de la seva tasca en concret:

- Mànager dels gràfics: abstraurà tota la funcionalitat d'Ogre3D de manera que farà transparent la creació de la finestra del joc i el dibuixat dels objectes i, a més, s'encarregarà de proporcionar una API per a la creació d'escenes, pel seu canvi d'una a altre escena, la creació i la gestió de les càmeres de les escenes i la creació i gestió dels diferents objectes gràfics.
- Mànager de les físiques: de manera equivalent al mànager dels gràfics abstraurà tota la funcionalitat de PhysX de manera que farà transparent l'actualització dels objectes, el moviment dels controladors, etc, i a més proporcionarà una API per a la creació d'escenes físiques, el seu canvi d'una a altre escena, la creació i gestió dels objectes físics i la creació i gestió dels disparadors i de les col·lisions.
- Mànager d'entrada d'usuari: s'encarrega de gestionar l'entrada del ratolí, del teclat i del controlador de la videoconsola Xbox360.

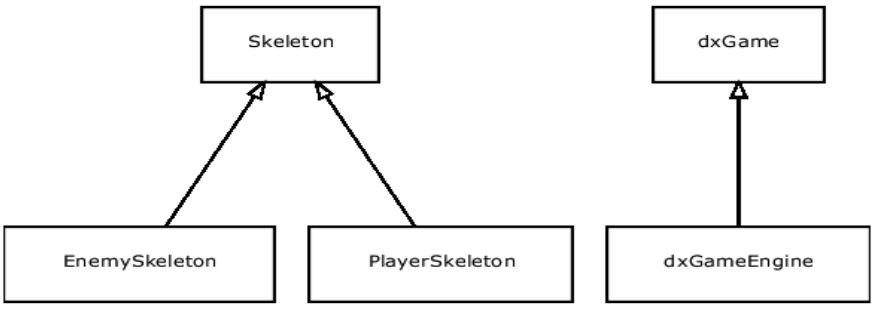
- Mànager del temps: s'encarrega de controlar el temps transcorregut entre actualitzacions del bucle principal, així com d'establir la velocitat del joc o, fins i tot, d'aturar el joc.

Tots aquests mànagers estaran continguts dins una classe motor de joc que serà l'encarregada d'actualitzar-los tots en cada actualització del bucle principal del joc. Per tant, el nostre motor de joc tindrà un esquelet bàsic com el descrit en el següent diagrama:



4.2 Disseny del joc

El disseny del joc pròpiament dit descansarà sobre el disseny del motor del joc. Així, tindrem una classe joc que estendrà les funcionalitats del motor del joc que ens permetrà crear les diferents classes d'elements concrets que necessitem. Veurem aquest fet amb més profunditat en el pròxim capítol de la memòria, però per citar un exemple, tindre una classe esquelet, que serà estesa per la classe esquelet enemic o esquelet jugador segons convingui. Ambdues seran creades pel propi joc i inserides dins al motor de joc per ser processades.



Capítol 5

Implementació

Un cop decidides les eines i llibreries a utilitzar i com serà l'esquelet bàsic tant del motor de joc com del joc pròpiament dit és el moment d'entrar en la implementació de la solució. Tradicionalment la planificació d'un projecte software constava d'un diagrama de Gant on s'especificava el nombre d'hores en les quals es realitzava una tasca, començant en un cert instant de temps i donant-se per acabada en un altre instant de temps. Un videojoc, no obstant, té una particularitat diferencial respecte a un desenvolupament software tradicional i és que en un videojoc podem implementar una funcionalitat però al mateix temps podem millorar una escena, ja sigui afegint objectes decoratius, més llums, algun enemic extra, afegir algun trencaclosques més, etc. Per tant, és molt més útil utilitzar una aproximació iterativa amb petites tasques a realitzar i sobre les que anirem millorant en futures iteracions. Grans companyies com Crytek[CRY01] utilitzen metodologies similars, evidentment en una escala molt diferent, a l'explicada anteriorment.

La implementació constarà de diferents iteracions. En la primera iteració crearem una primera versió del motor de joc. A partir d'aquí, anirem realitzant un seguit d'iteracions que ens portaran a la creació del joc pròpiament dit però que a l'hora ens faran millorar el motor de joc, bé perquè hem passat per alt alguna funcionalitat necessària o bé perquè ens hem adonat que es pot realitzar alguna millora per facilitar-nos la feina. Així, la implementació queda estructurada de la següent manera:

- Primera iteració: primera implementació del motor de joc (3 mesos)
- Segona iteració: millora del motor de joc (1 mes)
- Tercera iteració: primera implementació del joc (i millora del motor, si s'escau) (4 mesos)
- Quarta iteració: millora del joc (i millora del motor, si s'escau) (2 mesos)

Com més iteracions es puguin realitzar, millor serà la qualitat final del producte. Per tant, no s'ha d'entendre la llista anterior com a una llista tancada amb un final concret, si no que es

podrà ampliar. Cal destacar també que és possible que no s'arribi al final de la llista, però en qualsevol cas l'objectiu principal és tenir una primera versió del joc funcional.

5.1 Primera iteració

La implementació del projecte fi de carrera començarà amb la implementació del motor de joc que, com hem dit, serà la base sobre la que edificarem el joc final. En la primera iteració ens hem proposat com a objectiu aconseguir lligar correctament la part gràfica i la part física, creant els mànagers i les classes necessàries per aconseguir aquest objectiu. S'ha de destacar que per tal de tenir un motor el més plataforma possible s'intentarà crear com menys dependències millor entre els diferents mànagers i elements de cadascun dels sistemes, de manera que siguin fàcilment reemplaçables per altres en cas de ser necessari.

5.1.1 Elements comuns

El primer pas realitzat una vegada analitzats els elements necessaris ha estat crear una serie de classes comunes, per tal d'evitar utilitzar implementacions concretes de llibreries de tercers. Per citar un exemple, tant Ogre com PhysX tenen la seva pròpia implementació de vectors de tres components o de quaternions per a les rotacions. Per tant, utilitzar alguna de les dues implementacions obligaria a lligar el subsistema gràfic i el de físiques, cosa que no ens interessa. És per això que s'ha optat per la creació de les següents classes:

- dxVector3: representa la nostra pròpia implementació d'un vector de tres components. Ofereix funcionalitats com ara el producte vectorial, el producte escalar, obtenir la rotació necessària d'un vector a un altre, permet sumar, restar i multiplicar vectors del tipus dxVector3, obtenir les seves components, calcular la distancia al quadrat entre dos vectors, normalitzar el propi vector i fins i tot indicar que el vector és un vector error (per exemple, en el cas que alguna funció hagi de retornar obligatòriament un vector3 però el valor d'aquest vector no importi).
- dxQuaternion: representa la nostra pròpia implementació d'un quaternió. Ofereix la funcionalitat de crear un quaternió a partir d'una rotació i un vector de rotació i, a més, permet igualar i rotar un dxVector3 a partir del quaternió donat.

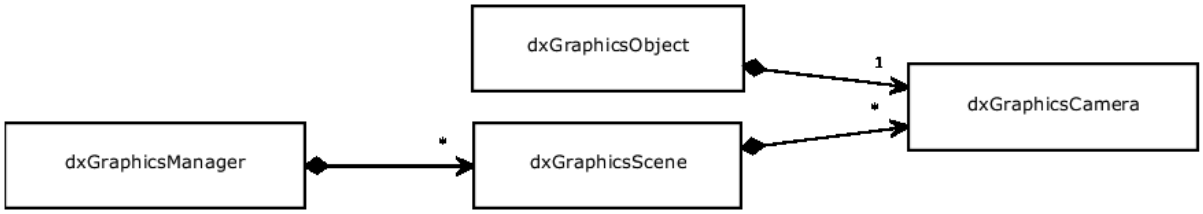
Podem veure les dues classes representades en el l'annex 1 diagrames 1 i 2

5.1.2 Sistema gràfic

Aquest sistema serà l'encarregat de crear la part gràfica de tots els objectes que creem, crear les escenes gràfiques, actualitzar-les, dibuixar la que correspongui, activar les càmeres, etc. En concret, s'han creat les següents classes, detallades a continuació:

- dxGraphicsManager: la classe per excel·lència de la part gràfica del joc. Configura la finestra de dibuixat, gestiona tota la funcionalitat relativa a la creació, activació i destrucció d'escenes, carrega els recursos gràfics, crea els objectes físics i permet gestionar els "listeners" d'Ogre3D, necessaris per a la funcionalitat de debug.
- dxGraphicsCamera: classe que encapsula la funcionalitat de les càmeres de Ogre3D. És l'encarregada de permetre'ns canviar la posició de la càmera, on està mirant aquesta, tornar la càmera als seus valors inicials i restaurar les característiques de la càmera després de perdre-les (molt útil en el cas que el nostre personatge entri en contacte amb la paret, com veurem més endavant).
- dxGraphicsScene: escena gràfica del joc. Aquesta classe és la que conté tota la funcionalitat relativa a activar o desactivar una càmera i inserir-les a la escena.
- dxGraphicsObject: part gràfica dels objectes que crearà el nostre motor. És l'encarregada de guardar la posició, rotació, jerarquia dins la escena i les animacions de l'objecte fent ús de les pròpies característiques d'Ogre3D. A més, podrà tenir associat un objecte del tipus dxGraphicsCamera que el seguirà, fet que permetrà crear càmeres en primera i tercera persona.

Podem veure com queden les quatre classes en els diagrames 3, 4, 5 i 6 de l'annex diagrames. El sistema queda relacionat de la següent manera:



El funcionament del nostre sistema gràfic serà el següent:

1. Crearem un objecte dxGraphicsManager i cridarem al seu mètode Init (admet com a paràmetres configuració de la resolució, el nom de la finestra i si volem executar el joc en pantalla completa), mètode que configurarà Ogre3D per tal de crear la finestra amb els paràmetres obtinguts.
2. Un cop el mètode nostre mànager estigui inicialitzat, podrem configurar els directoris on Ogre3D haurà de buscar els recursos gràfics mitjançant AddResourceLocation, que admet com a paràmetre una llista amb tots els directoris on ogre haurà de buscar.
3. Crearem almenys una escena mitjançant el mètode CreateScene, al que l'hi passarem com a paràmetre el nom de l'escena. Aquesta escena tindrà una càmera per defecte per evitar poder tenir una escena sense cap càmera, ja que no seria visible. Internament s'encarregarà de crear un Ogre::SceneManager que serà on afegirem tots els nostres objectes i càmeres.
4. Crearem els objectes que volem que apareguin a la nostre escena mitjançant els mètodes CreateEntity o CreateEntityWithCamera, segons convingui. Aquest mètode crearà les entitats d'Ogre3D i les afegira a un Ogre::SceneNode que a la seva vegada penjarà del node principal del Ogre::SceneManager corresponent, segons a quina escena creem cada objecte.
5. Opcionalment, crearem càmeres addicionals mitjançant el mètode CreateCamera, que seguiran el mateix sistema que la creació d'objectes.
6. Activarem la escena que desitgem. Aquest mètode internament dirà a la finestra d'Ogre3D (Ogre::RenderWindow) quin és el Ogre::SceneManager que s'ha de dibuixar.

7. Crearem un bucle infinit on anirem cridant a la funció Update per tal d'anar actualitzant la pantalla de dibuix. Internament aquest mètode es limitarà a la crida del mètode Update de la finestra d'Ogre3D.

5.1.3 Sistema de físiques

Aquest sistema serà l'encarregat de crear la part física de tots els objectes que creem, bé siguin a partir d'una malla, un controlador (de capsa o de càpsula) o un disparador, crear les escenes físiques i simular la que sigui necessària. En concret, s'han creat les següents classes, detallades a continuació:

- dxPhysicsManager: de manera anàloga a la classe dxGraphicsManager, s'encarregarà de la creació i activació de les escenes físiques, de la càrrega de recursos, de la creació dels objectes, controladors i disparadors, i de la simulació de l'escena que estigui activa en aquell moment.
- dxControllerHitReport: s'encarrega de gestionar el tema de les col·lisions dels controladors, és a dir, ens informarà si un dels personatges col·lisiona amb algun tipus d'element físic, ja sigui un altre controlador o un objecte estàtic. La funció única d'aquesta classe serà cridar a mètodes de la classe dxObjectHitReport passant la informació concreta dels objectes que han col·lisionat. Aquesta última classe serà la classe base sobre la que treballarem tot el tema de col·lisions.
- dxPhysicsSimulatonEventCallback: de manera anàloga a dxControllerHitReport, aquesta classe s'encarregarà de gestionar els esdeveniments que passen a l'escena. Bàsicament, ens informarà si algú ha entrat dins un disparador, cridant al mètode del propi disparador passant-li com a paràmetre l'objecte que ha entrat en ell.
- dxPhysicsObject: part física dels objectes que crearà el nostre motor. És l'encarregada d'informar-nos de la seva posició i orientació per tal que podem actualitzar correctament la part gràfica del objecte. A més, serà l'encarregada de moure els nostres controladors, aplicar la gravetat o no al controlador (en el cas dels controladors, l'aplicació de la gravetat l'hem de realitzar manualment, en la resta d'objectes s'aplica

automàticament) i definir si volem que el nostre objecte respongui a les preguntes que realitzem a l'escena (per exemple, si estem col·lisionant amb algun objecte).

- dxPhysicsScene: és l'encarregada de configurar la gravetat de la nostre escena i de realitzar les preguntes a l'escena de PhysX. Per exemple, podem preguntar si un raig col·lisiona amb algun objecte durant el seu recorregut i, en cas de ser afirmatiu, en quin punt o amb quin objecte està col·lisionant. A més ofereix informació de debug per tal de poder comprovar si el món gràfic i el físic estan funcionant correctament.

Podem veure com queden les cinc classes en els diagrames 7, 8, 9, 10 i 11 de l'annex diagrames.

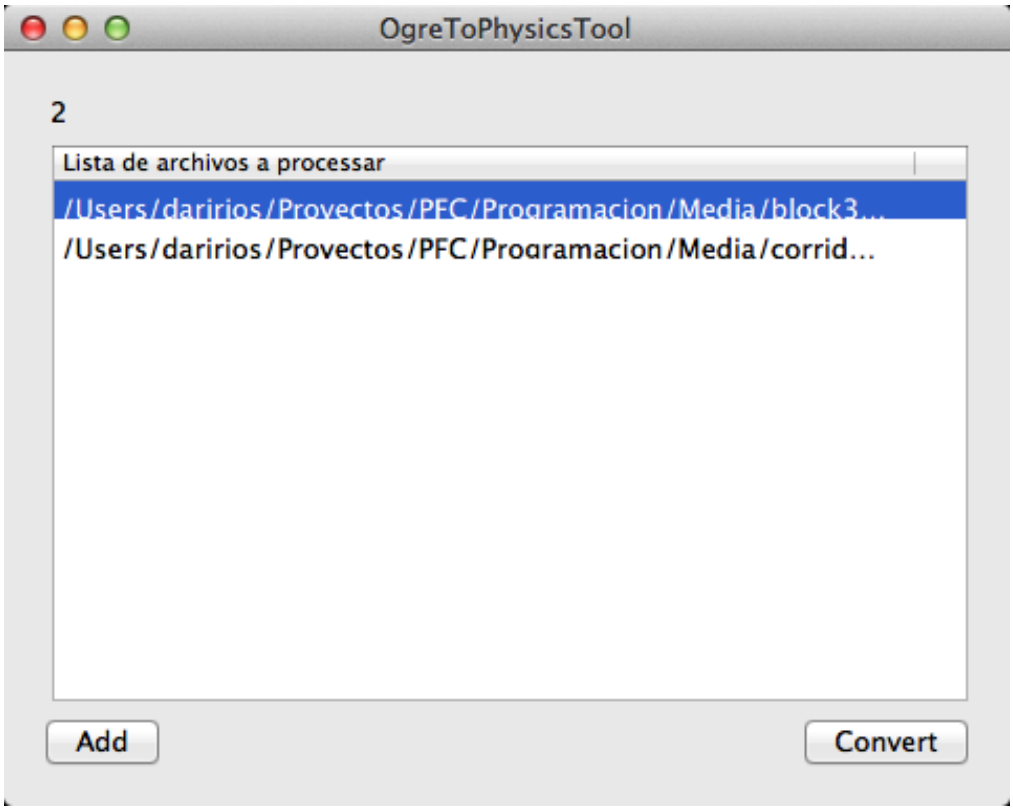
El funcionament del nostre sistema físic serà el següent:

1. Crearem un objecte dxPhysicsManager i cridarem al seu mètode Init, mètode que configurarà la llibreria PhysX creant els objectes necessaris i inicialitzant-los. A més, crearà un material per defecte que aplicarem als objectes que creem posteriorment.
2. Un cop tenim el mànager inicialitzat, indicarem a aquest on estan els recursos físics necessaris si volem crear algun objecte a partir d'una malla (fitxers .nxs). Ho farem mitjançant un mètode homòleg a l'utilitzat en el sistema gràfic, el mètode AddResourcesLocation.
3. Crearem almenys una escena física mitjançant el mètode CreateScene, que rebrà com a paràmetre el nom de l'escena i la gravetat que aplicarem inicialment a la mateixa.
4. Crearem els objectes físics que volem, bé siguin a partir d'una malla (en aquest cas seran objectes estàtics) mitjançant el mètode CreatePhysicsObjectFromMesh, bé siguin objectes dinàmics mitjançant controladors (tant de capsa com de càpsula) mitjançant els mètodes CreateCapsuleController o CreateBoxController o bé els disparadors, mitjançant el mètode CreateTrigger.
5. Un creats tots els objectes, activarem una de les escenes mitjançant el mètode ActivateScene.

6. Finalment, crearem un bucle infinit on simularem l'escena activa passant-li com a paràmetre el temps transcorregut entre simulacions

5.1.4 Unió dels dos sistemes

Una vegada acabats els dos sistemes (gràfics i físics) toca crear els elements necessaris perquè ambdós treballin conjuntament. El primer pas realitzat ha estat crear una aplicació externa (sobre la base del sistema gràfic i físic) per tal de poder crear els fitxers .nxs necessaris per a poder crear objectes físics a partir de malles gràfiques. Per fer-ho, s'ha heretat de la classe dxGraphicsManager i s'ha afegit un mètode per a realitzar el proces, anomenat CreatePhysicsMesh. Un cop hem tingut el mètode creat i hem pogut comprovar que funciona correctament s'ha creat una petita aplicació per a Mac OS X anomenada OgreToPhysxTool que permet processar arxius .mesh de manera seqüencial. Així, només hem de prémer el botó d'afegir, seleccionar les mesh que volem processar i prémer "Convert". El programa ens crearà els fitxers .nxs en el mateix directori on tenim situats els .mesh a convertir.

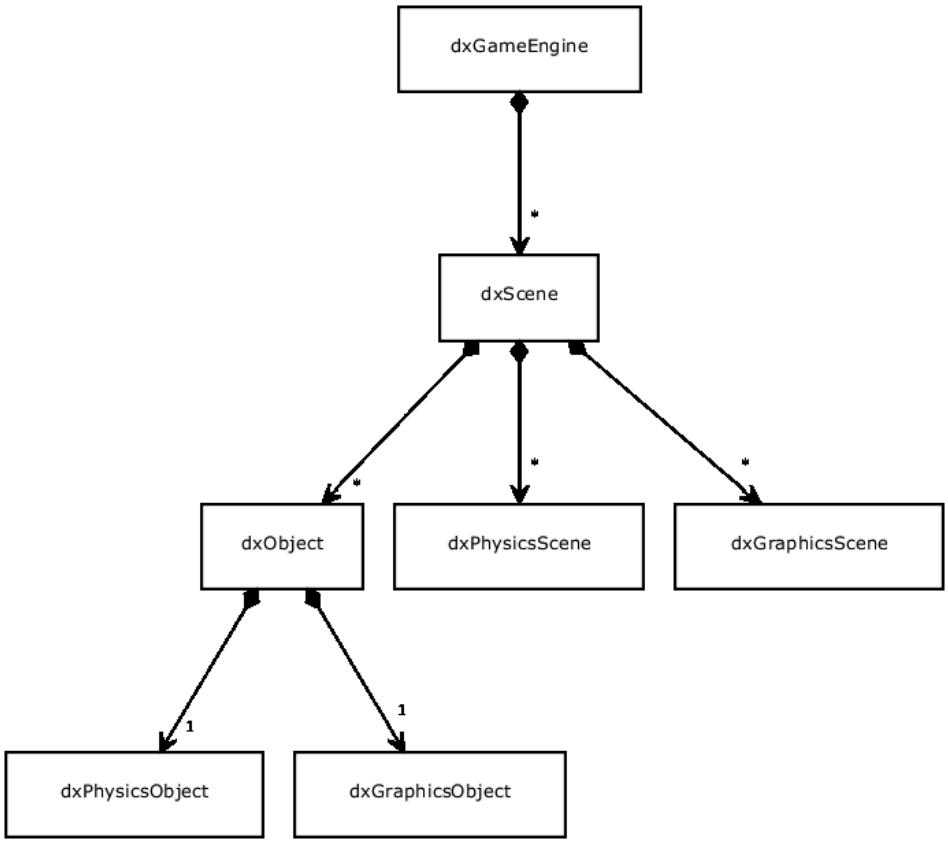


Captura de pantalla de l'aplicació OgreToPhysicsTool

El segon pas a realitzar ha estat la creació de les classes que uniran la part gràfica i la part física. Així, s'ha creat la classe dxObject, la classe dxScene i la primera aproximació a la classe dxGameEngine, comentades a continuació:

- dxObject: classe que conté la part gràfica i la part física d'un objecte. Serà la base sobre la que construirem tota la resta de classes (esquelets, portes, etc), tot dins el nostre joc seran objectes de la classe dxObject. La seva funcionalitat bàsica serà mantenir el món gràfic actualitzat segons el que està passant al món físic mitjançant el mètode Synchronize. Aquest mètode haurà de ser cridat per la funció Update del propi objecte o bé per la funció Update de la classe hereva de dxObject.
- dxScene: classe que conté la part gràfica i la part física d'una escena. Un cop creat un objecte del tipus dxObject aquest ha de ser inserit dins una dxScene per tal que aquest objecte pugui ser actualitzat correctament. Així, aquesta classe s'encarrega de gestionar tots els objectes continguts en una escena, actualitzar-los i oferir mètodes per recuperar-los, a més d'oferir funcionalitats extres com ara raigs de col·lisió (dels que podem recuperar o bé el punt de col·lisió o bé l'objecte amb el que ha col·lisionat el raig), la configuració de la gravetat o bé obtenir la informació de debug de l'escena.
- dxGameEngine: classe base de tot el projecte. Permetrà crear les escenes, activar-les, actualitzar l'escena activa i activar i desactivar el mode de debug. A més oferirà un mètode virtual per crear objectes anomenat CreateObject, que serà l'encarregat de ser sobreescrit per la nostra classe de joc i serà on crearem els diferents elements concrets que formaran el nostre joc (per exemple, els esquelets).

Podem veure com quedaran les tres classes en els diagrames 12, 13, 14 de l'annex de diagrames . La relació entre les classes, per tant, serà la següent:



El funcionament final del nostre motor de joc en la primera iteració serà el següent:

1. Processarem els arxius gràfics mitjançant 3dstudio max i el plugin OgreMax per tal de convertir els fitxers .FBX a format .mesh, format utilitzat per Ogre3D.
2. Processarem els arxius gràfics generats en el pas anterior per poder crear els arxius físics necessaris. Aquest pas el realitzarem mitjançant la nostra pròpia aplicació (OgreToPhysicsTool) creada per realitzar aquesta funcionalitat. Així, per cada fitxer .mesh que correspongui a objectes estàtics realitzarem la seva conversió per tal de crear els fitxers .nxs necessaris per permetre a PhysX crear les malles estàtiques.
3. Inicialitzarem el motor de joc mitjançant el seu mètode Init.
4. Especificarem la ruta on es troben tots els fitxers gràfics i físics mitjançant el mètode AddResourcesLocation.
5. Crearem les escenes que creguem necessàries mitjançant el mètode CreateScene.

6. Crearem els objectes que creguem necessaris mitjançant el mètode CreateObject.
7. Opcionalment, activarem el mode debug si s'escau mitjançant el mètode SetDebugMode.
8. Activarem l'escena que volem que sigui l'escena inicial mitjançant el mètode ActivateScene.
9. Executarem el mètode Run per tal d'executar un bucle infinit d'actualització, simulació i pintat de l'escena, o bé crearem un bucle infinit on anirem cridant al mètode RunOneStep.

5.2 Segona iteració

En aquesta segona iteració del motor del joc s'afegirà funcionalitat addicional a aquest, intentant completar el motor del joc anterior amb tot el necessari per tal de poder crear un videojoc. En concret, es crearan les classes necessàries per poder controlar el temps transcorregut entre fotogrames (necessari per a realitzar correctament els càlculs de les físiques) i per a permetre d'interacció de l'usuari, és a dir, poder utilitzar el teclat, el ratolí i un controlador de Xbox360 per controlar els menús i el nostre personatge.

5.2.1 Sistema del temps

Aquest sistema, encarregat de gestionar el temps transcorregut entre fotogrames, serà un dels sistemes més simples de tot el motor del joc. La seva funcionalitat es limitarà només en realitzar aquesta funció i calcular el nombre de fotogrames per segon als que va el nostre joc. No obstant això, serà en aquest sistema en el que haurem d'implementar en un futur, si s'escau, una funcionalitat que permeti saber si un determinat temps ha expirat. Per tant, és important separar aquesta funcionalitat en un sistema separat de la resta de sistemes. El sistema del temps, per tant, constarà únicament de la següent classe:

- dxTimerManager: classe encarregada de controlar el temps transcorregut entre fotogrames, calcular el nombre de fotogrames per segon als quals s'està executant el nostre joc i permet canviar el temps lògic transcorregut pel joc, és a dir, físicament

poden haver passat 16 milisegons entre fotogrames però el nostre mànager del temps ens retornarà 32 milisegons, de manera que la velocitat del joc serà el doble.

Podem veure com quedarà el sistema en el diagrama 15 de l'annex de diagrames.

El funcionament final d'aquest sistema serà el següent:

1. Inicialitzarem el sistema mitjançant la crida al mètode Init del sistema.
2. Per cada iteració del nostre motor de joc, cridarem el mètode Update. Aquest mètode s'encarregarà de calcular el temps transcorregut entre fotogrames i també de calcular el nombre de fotogrames per segon als que s'està executant el joc.
3. Obtindrem el temps transcorregut entre fotogrames mitjançant el mètode GetElapsedTime. Aquest resultat és el que passarem per paràmetre al mètode Update de cada sistema per tal que tots els sistemes sàpiguen el temps lògic transcorregut entre fotogrames i poder actualitzar-se en conseqüència.

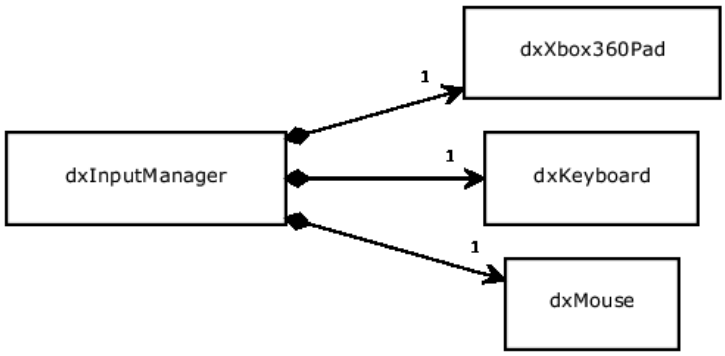
5.2.2 Sistema d'entrada del usuari

Aquest sistema serà l'encarregat de permetre a l'usuari comunicar-se amb el sistema. Per fer-ho, donarà suport a l'entrada per teclat, a l'entrada per ratolí i a l'entrada mitjançant un comandament de la videoconsola Xbox360. En concret, s'han creat les següents classes:

- dxKeyboard: classe encarregada de gestionar l'entrada mitjançant teclat. Ofereix la funcionalitat necessària per tal de poder conèixer si l'usuari ha començat a prémer una tecla, l'està prement o bé l'ha deixat de prémer. A més, ofereix la possibilitat de conèixer quina direcció ha premut l'usuari tant en les tecles ASDW com en les tecles de direcció (usualment utilitzades per controlar els jugadors).
- dxMouse: classe encarregada de gestionar l'entrada mitjançant ratolí. Permet controlar quin ha estat el moviment del ratolí respecte el fotograma anterior mitjançant l'increment de la seva posició horitzontal i vertical. A més, ofereix la funcionalitat per conèixer quin botó del ratolí s'ha començat a prémer per part de l'usuari, quin està prement i quin ha deixat de prémer.

- dxXbox360Pad: classe encarregada de gestionar l'entrada mitjançant el controlador de Xbox360. Ofereix la funcionalitat per saber en quina direcció estem movent qualsevol dels dos joystick o la creueta, i quins botons hem començat a prémer, estem prement o hem deixat de prémer.
- dxInputManager: mànager encarregat encarregat de gestionar tots els sistemes d'entrada de l'usuari. Serà l'encarregat d'inicialitzar els diferents sistemes d'entrada segons els nostres requeriments i d'actualitzar el seu estat en cadascun dels fotogrames del joc.

Podem veure com quedaran les quatre classes en els diagrames 16, 17, 18 i 19 de l'annex de diagrames. La relació entre les classes, per tant, serà la següent:



El funcionament final d'aquest sistema serà el següent:

1. Inicialitzarem el sistema d'entrada d'usuari mitjançant el mètode Init de la classe dxInputManager.
2. Afegirem els sistemes d'entrada que volem utilitzar mitjançant els mètodes AddXbox360PadSupport, AddKeyboardSupport i AddMouseSupport segons si volem utilitzar el controlador de Xbox360, el teclat o el ratolí.
3. Cridarem el mètode Update en cada actualització del motor del joc.
4. Obtindrem el controlador de Xbox360, el teclat o el ratolí mitjançant els mètodes GetXbox360Pad, GetKeyboard i GetMouse.
5. Consultarem l'estat del sistema d'entrada desitjat mitjançant les crides als mètodes propis de cadascun dels sistemes.

5.3 Tercera iteració

Un cop acabades les dues iteracions anteriors del motor del joc es considera que s'ha realitzat un motor de joc que ofereix les funcionalitats mínimes necessàries per poder construir un joc sobre ell. No obstant, és evident que no s'han pogut contemplar totes les necessitats que apareixeran durant la creació del joc, de manera que les haurem d'anar introduint en iteracions posteriors.

L'objectiu principal d'aquesta tercera iteració serà permetre crear un personatge controlable, moure'l per la escena, realitzar correctament les col·lisions i crear una càmera en tercera persona que segueixi el personatge.

S'ha de destacar, també, l'ús de Unity3D per realitzar aquesta part d'implementació del joc. Si bé en un principi aquest motor va ser descartat ja que no complia l'objectiu de ser programable en C++, durant el desenvolupament d'aquest projecte fi de carrera vaig tenir l'oportunitat de participar en un certamen de formació de desenvolupadors (AppCity2012[AC2012]) on vaig aprendre el funcionament d'aquest motor. Durant aquest certamen es va proposar als assistents realitzar un videojoc i, en el meu cas, vaig decidir utilitzar un document de disseny similar al aquí presentat. El certamen no podia anar millor i vaig guanyar el primer premi al millor joc desenvolupat durant el certamen[AEU3D], fet que em va fer guanyador d'una llicència d'ús professional d'aquest motor. Així doncs, s'ha aprofitat part del treball fet i les facilitats d'ús d'aquest motor per, per exemple, composar les escenes que finalment es veuran en aquest projecte fi de carrera.

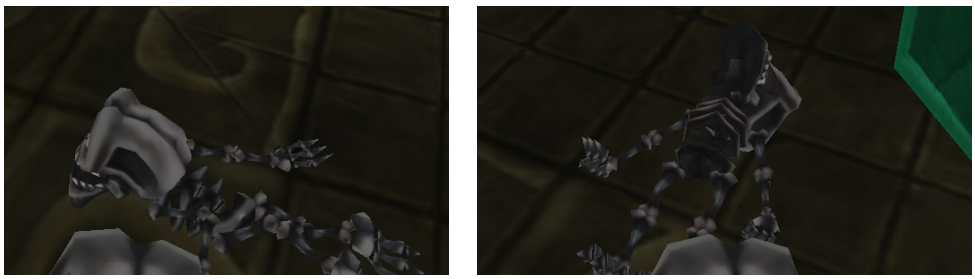
5.3.1 Creació del personatge esquelet

La classe Skeleton serà la classe bàsica que implementarà les funcions bàsiques que realitzarà qualsevol dels esquelets del joc. Aquesta classe heretarà de la classe dxObject, classe base de tots els objectes del joc i implementarà una màquina d'estats que permetran al esquelet realitzar les seves funcions.

Entrant amb més detall, un esquelet podrà realitzar les següents funcions:

- Atacar: l'esquelet realitzarà l'acció d'atacar, si algun esquelet enemic està a prop, aquest rebrà un cert dany.

- Rebre un atac: en el cas que un enemic ens ha atacat, el nostre esquelet haurà de rebre el dany. A més, serà important saber des d'on ens han atacat ja que, en el cas de perdre l'últim punt de vida, l'esquelet haurà de morir amb una animació que es correspongui a la direcció de la qual ha rebut l'atac. Així, si som atacats per l'esquena, el nostre esquelet caurà cap a davant. De manera anàloga, si rebem un ataca frontal, el nostre esquelet caurà cap enrere.



Exemple de la mort de dos esquelets, segons la direcció de l'últim atac rebut.

- Actualitzar el seu estat intern: a part de les accions pròpies de l'esquelet haurem de controlar altres accions. Així, si parem de moure el nostre esquelet, aquest haurà de seguir realitzant un seguit d'accions, com ara passar al estat de repòs, actualitzar les animacions, etc.
- Moviment: la classe esquelet no oferirà cap mecanisme per moure directament l'esquelet. Seran les classes que heretin d'aquesta les encarregades de actualitzar el seu vector de moviment, bé sigui a través de l'entrada de l'usuari o bé sigui a través de la seva pròpia intel·ligència artificial.
- Realitzar accions: aquest mètode, inicialment buit, serà el que sobreescriurem en les classes que heretin de Skeleton i serà on realitzarem, o bé el tractament de l'input, o bé la intel·ligència artificial.

5.3.2 Creació del esquelet jugador

Sobre la base de la classe Skeleton s'ha construït una nova classe SkeletonPlayer, encarregada de controlar el personatge esquelet mitjançant les comandes del jugador. La única

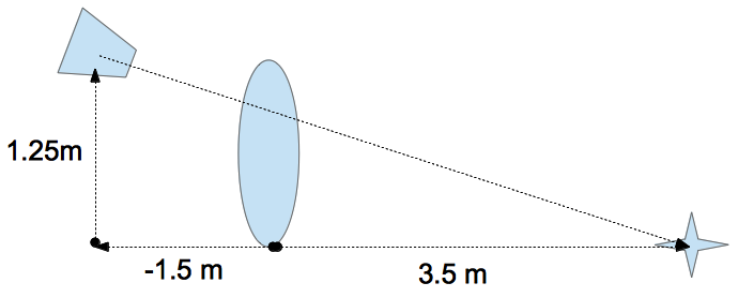
diferencia respecte a la seva classe base serà el mètode Actions. En aquest mètode tractarem l'entrada de l'usuari i actualitzarem la càmera en tercera persona de l'esquelet.

Entrada d'usuari

En aquesta primera versió del joc, l'usuari només podrà atacar. Per tant, només haurem de gestionar els botons o tecles de direcció (segons juguem amb el controlador de Xbox360 o amb el teclat), la direcció a la que mirem (amb el controlador de Xbox360 o amb el ratolí), i els botons o tecles de funció (amb el controlador de Xbox360 o amb el teclat). Realitzar aquesta gestió serà trivial gràcies als mètodes i a les classes del sistema d'entrada d'usuari.

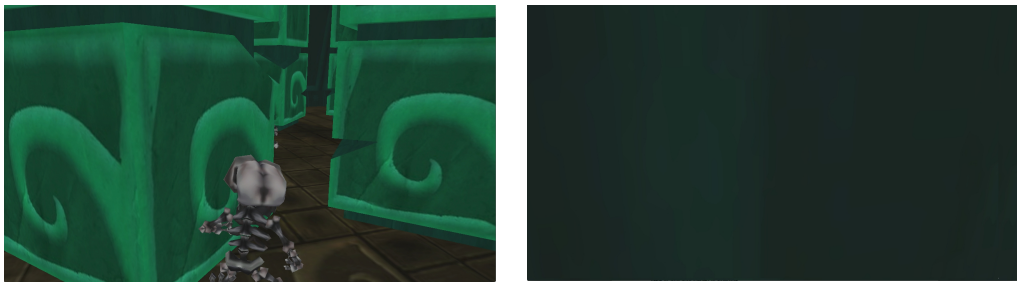
Càmera en tercera persona

Utilitzant la funcionalitat d'Ogre3D i el seu funcionament mitjançant nodes, s'ha creat la següent estructura per a la càmera 3D en tercera persona. Primerament s'ha creat el node pare, que serà l'encarregat de dibuixar el nostre esquelet a la seva posició corresponent. Després, com a fills d'aquest pare, s'han creat dos nodes, el node de la càmera, i el node de l'objectiu on mirarà la càmera. D'aquesta manera, cada cop que fem moure el nostre personatge, tant la càmera com el punt on mirarà la càmera s'actualitzaran correctament. En la següent figura podem veure de manera esquemàtica com quedaran els nostres objectes. Tindrem el nostre personatge en forma de càpsula al centre, la càmera situada darrera del nostre personatge a una altura de 1.25m i a una distància de 1.5m i l'objectiu de la càmera a una altura 0 i a una distancia de 3.5m respecte del nostre objecte.



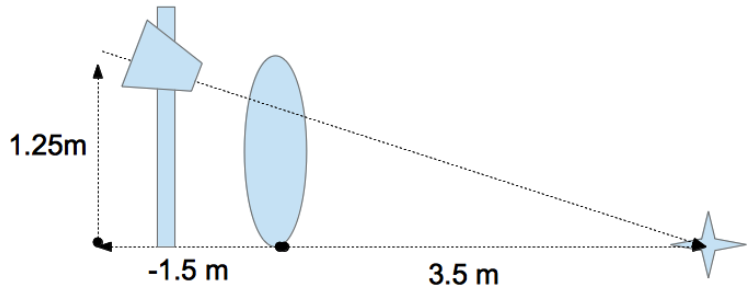
Esquema de la posició per defecte de la càmera

Aquesta aproximació ha funcionat correctament en entorns oberts però ens ha comportat problemes en espais tancats. Així, si apropem el nostre personatge a una paret, tindrem el problema que la càmera traspassarà aquesta paret i, per tant, tindrem problemes amb el què està veient el jugador, com podem veure en el següent exemple:



Exemple de problema amb la càmera

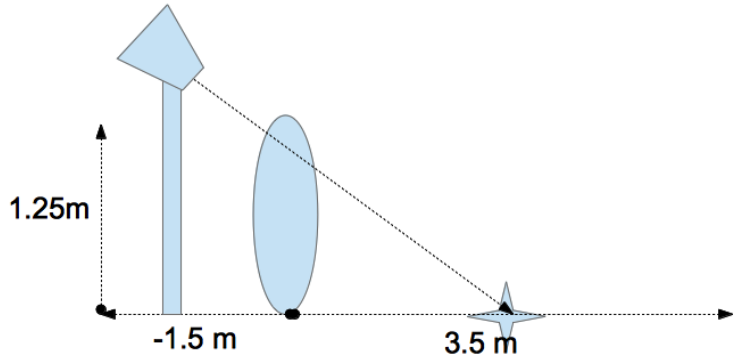
Per solucionar aquest problema s'ha optat per configurar la càmera segons la distancia a la que estem de la paret. Així, mitjançant la llibreria PhysX, llançarem un raig des de la posició del nostre esquelet en direcció a la càmera. Si aquest raig col·lisiona amb la paret situarem la càmera a la distancia exacte on s'ha produït la col·lisió, de manera que tindrem la càmera sempre per davant de la paret.



Esquema de la posició de la càmera al tenir una paret en la seva trajectòria

Aquesta, no obstant, no ha estat la solució definitiva. Si ens fixem en el resultat, tenim una càmera a una altura molt més baixa i molt propera a l'esquelet. Amb aquesta aproximació s'han produït casos en els que l'esquelet es deixava de veure per estar la càmera massa propera a ell. Si bé a molts jocs aquest efecte és el desitjat, és a dir, s'activa una transparència per tal de no tenir problemes amb un model massa proper a la càmera, en el cas d'aquest projecte he optat per buscar una altre solució que no provoqui que l'esquelet desaparegui. La solució final

opta per, en cas de trobar-nos amb una paret entre el nostre personatge i la càmera, moure la càmera a una posició més propera al personatge i, a més, moure-la a una altura més alta a l'hora que la distancia on mira la càmera es redueix. Així, tot i que tenim el problema de perdre una mica de camp de visió, solucionem el problema de perdre l'esquelet de vista.



Esquema final de la posició de la càmera al tenir una paret en la seva trajectòria

Amb aquesta solució final apareix un últim problema, la recuperació de la càmera. No podem passar del tot al res. Així, el que farem per solucionar aquest problema és posar l'altura de la càmera i la proximitat del objectiu segons la proximitat de la paret. D'aquesta manera, si ens anem allunyant de la paret, la càmera cada cop es situarà a menys altura i l'objectiu a una distancia més llunyana, fins a arribar als valors per defecte.

Exemple de com la càmera recupera la seva posició original

5.3.3 Creació esquelet enemic

La primera implementació d'un enemic en el joc ha constat només d'una mínima intel·ligència artificial. Aquesta intel·ligència es limitava a comprovar si hi havia algun enemic dins el seu radi d'acció. En cas de tenir un enemic dins el radi d'acció l'esquelet es girava per mirar-lo i immediatament es posava a seguir-lo. Per realitzar aquesta intel·ligència artificial ens hem basat en les funcionalitats que ens ofereix PhysX altre cop. Així, sabem que dos cossos es veuen si al llançar un raig des d'un dels dos en direcció a l'altre, el punt de col·lisió està més allunyat que la distancia entre els dos objectes. També sabem com hem de posicionar l'enemic (pel que fa a la seva rotació) de manera molt senzilla realitzant un producte escalar entre el la direcció cap on està mirant actualment l'enemic i la direcció on esta el seu objectiu:

1. Normalitzem el vector resultant d'obtenir la direcció cap on està mirant actualment l'enemic (en l'esquema correspon al vector A)
2. Calculem el vector de la direcció cap on l'enemic hauria de mirar mitjançant la resta entre la posició actual de l'objectiu de l'enemic i la posició actual de l'enemic.
3. Normalitzem aquest vector resultant (en l'esquema correspon al vector B)
4. Sabent que $A \cdot B = |A|*|B|*\cos(\theta)$ i que $|A| = 1$ i $|B| = 1$ tenim que $A \cdot B = \cos(\theta)$.
5. Calculem $\arccos(\theta) * 360$ i tenim l'angle en graus que haurem de rotar l'enemic per tal que aquest estigui mirant la seva presa.

Esquema i exemple d'enemic encarant un objectiu

5.3.4 Millores en el motor de joc

Un cop començat a implementar el joc, com ja hem dit, és lògic i normal que sorgeixin nous requeriments que haurien d'estar inclosos dins el motor del joc. Un dels requeriments que ha aparegut en aquesta iteració ha estat la necessitat de poder configurar escenes externament, afegint i eliminant parts de les mateixes des d'un arxiu de text. És per això que s'ha modificat el codi del creador d'escenes per carregui els objectes que haguem definit en un XML. Per fer-ho, s'ha utilitzat un dels més simples lectors de XML, tinyxml[TXML]. El seu funcionament es basa en anar llegint de manera molt simple els nodes que volem del fitxer XML i els seus atributs. El format final que han seguit les escenes ha estat el següent, comentat en base a un exemple real:

```
<Escena nombre="demotest" gravedad="-9,81f">
   <ListaCameras>
      <Camera nombre="NombreCamera">
         <Posicion x="0" y="50" z="1" />
         <LookAt x="0" y="0" z="0" />
      </Camera>
        ….
   </ListaCameras>

   <ListaObjetos>
      <Objeto nombre="Personaje" tipo="SkeletonPlayer" grupo="jugador">
                                    <Graficos    tipo="EntityWithCamera"    mesh="skeletonFINAL.mesh"
camera="NombreCamera2" cameraDistance="1.5" cameraHeight="1.25" targetDistance="3.5" />
         <Fisica tipo="CapsuleController" material="Default" altura="0.40" radio="0.45" />
         <Posicion x="0" y="5" z="0" />
         <Rotacion x="0" y="0" z="0" angulo="0"/>
      </Objeto>

      <Objeto nombre="Estancia6" tipo="" grupo="">
         <Graficos tipo="Entity" mesh="room2.mesh" />
         <Fisica tipo="Mesh" material="Default" />
         <Posicion x="-20" y="0.289" z="2" />
         <Rotacion x="0" y="1" z="0" angulo="0"/>
      </Objeto>
        …
   </ListaObjetos>
</Escena>
```

1. L'inici d'una escena el marca la definició del nom de l'escena i la gravetat associada a aquesta (en la component Y)
2. El fitxer continua definint totes les càmeres de l'escena. S'han de definir també les càmeres que formaran part dels objectes, és a dir, si volem una càmera en tercera persona, és necessari definir-la prèviament.
3. L'última part del fitxer correspon a la definició dels objectes. Cada objecte està format, al menys, per un nom, un tipus (en cas de ser buit, es considera que l'objecte serà de tipus dxObject), un grup (pot ser buit, s'utilitza per classificar els objectes en cas que volem recuperar tots els objectes d'un tipus, per exemple, enemics), un objecte gràfic (que especifica els paràmetres per crear un dxGraphicsObject, en cas de tenir tots els camps buits es considerarà un objecte sense malla gràfica), un objecte físic (que especifica els paràmetres per crear un dxPhysicsObject, ja sigui una malla estàtica, un controlador o un disparador), una posició i una rotació.

S'ha de destacar especialment el camp tipus. Si volem afegir un nou element al nostre joc serà suficient amb indicar en aquest camp de quin tipus és i sobreescriure el mètode CreateObject per tal de poder crear objectes amb aquest tipus. Aquesta funcionalitat ens dóna una gran versatilitat ja que fa que sigui extremadament senzill afegir nous elements al nostre joc.

5.4 Quarta iteració

En aquesta quarta i última iteració s'ha acabat de completar una versió demostrativa de tot el treball realitzat. En concret, s'han afegit noves classes d'enemics i s'ha creat el mapa de pantalla d'inici de joc i el primer dels escenaris. S'ha de destacar també la creació, pel que fa al motor de joc, de la classe encarregada de la gestió del so, fins ara obviada per falta de temps. En concret, s'ha creat les següent classe:

- dxSoundManager: classe encarregada de gestionar el so del joc. Com en el cas del gestor de temps, es tracta d'una classe molt simple gràcies a la facilitat d'ús de la llibreria emprada (SFML).

Podem veure com quedarà aquesta classe en el diagrama 20 de l'annex de diagrames.

El funcionament de la classe dxSoundManager serà el següent:

1. Inicialitzarem el sistema mitjançant la crida al mètode Init.
2. Afegirem tots els sons que volem carregar mitjançant el mètode LoadSound.
3. Reproduirem el so en el moment que volem mitjançant el mètode PlaySound.

Aquesta ha estat l'última iteració realitzada sobre el projecte, de manera que el projecte queda tancat en aquest punt. No obstant això, com veurem en els següents punts, encara hi ha moltes coses a millorar i a ampliar, de manera que aquesta iteració en cap cas es pot considerar un punt i final.

Capítol 6

Resultats

En aquest capítol es mostraran els resultats finals del projecte fi de carrera. S'analitzaran per separat els resultats de la creació del motor de joc dels resultats de la creació

6.1 Motor de joc

Pel que fa al motor de joc hem acabat el projecte amb un motor capaç de suportar la creació d'un joc sobre seu de manera fàcil. Tot el motor es basa en sistemes (so, gràfic, físic, entrada d'usuari) fàcilment modificables gràcies al esforç realitzat per crear els sistemes de manera el més independent possible. S'ha de destacar que el desenvolupament del motor ha coincidit amb un canvi molt important pel que fa a la llibreria de físiques Nvidia PhysX, de la seva versió 2.8 a la seva nova versió 3.0 (actualment 3.2). Aquest ha estat un canvi molt violent (han canviat per complet totes les classes) al qual ens hem pogut adaptar de manera més o menys fàcil gràcies precisament al fet de separar el màxim possible els sistemes.

Destacar del motor del joc també la funcionalitat de crear nous objectes. Gràcies a aplicar aquest mètode (tenir un mètode factoria que s'encarregui de la creació de tots els objectes) és molt senzill afegir nous objectes al projecte, ja que només ens hem de dedicar a implementar la classe en qüestió (heretant de dxObject) i dotar al motor de la funcionalitat de llegir el fitxer XML de la manera que correspongui, no ens hem de preocupar de res més.

Per acabar l'anàlisi dels resultats pel que fa al motor del joc, destacar també el suport tant de Microsoft Windows com de Mac OS X. El projecte va començar a realitzar-se en un entorn Windows per ser migrat després a l'entorn de Mac OS X. Aquest fet va provocar certs problemes, com el comentat anteriorment en la llibreria de so (no compatible en aquest segon sistema) però que ha permès que el resultat final sigui executable en ambdues plataformes, si bé el projecte final només està creat per l'entorn Mac OS X.

6.2 Joc

Pel que fa al joc pròpiament dit hem acabat el projecte amb un joc curt però demostratiu del que es podria arribar a fer dedicant més temps al projecte. Així, el joc comença en una pantalla on elegim si jugar o sortir del joc en un entorn tridimensional per seguir en el primer nivell dels laberints en cas d'entrar en la habitació que porta a la opció jugar.

Imatge del menú principal

Un cop en el primer nivell, ens trobem amb un seguit de reptes i trampes. Per citar algunes de les funcions implementades, tenim habitacions que en si mateixes son una trampa, ja que ens tanquen dins seu fins que no aconseguim derrotar a tots els monstres i recollir tots els cristalls i altres habitacions que no s'obriran fins passar per un seguit de reptes.

En quant als enemics, tenim tres tipus d'enemics en el joc, fet que li dóna una certa varietat tot i comptar només amb un nivell complet. A més els enemics compten amb una certa intel·ligència artificial, fet que li dóna certa gracia al joc.

En quant a la jugabilitat, s'ha implementat les funcions bàsiques per tal de poder sobreviure en l'entorn del laberint. Destacar la creació d'una càmera en tercera persona, una elecció molt més complicada que no pas tenir una càmera fixa o en primera persona.

Una de les escenes del joc

Part final del joc

Capítol 7

Conclusions i treball futur

Un cop acabat el projecte fi de carrera és moment de fer balanç. Observant els objectius inicials del projecte crec que s'han acomplert ja que:

- S'han analitzat les eines i llibreries disponibles i s'ha escollit d'acord a les necessitats que plantejava el projecte.
- S'ha creat un motor de joc que ha demostrat ser el suficientment complert com per a crear sobre ell un videojoc 3D.
- S'ha realitzat una implementació mitjançant una metodologia àgil, molt emprada avui en dia a la industria.
- S'ha creat un videojoc que pot servir com a carta de presentació per a demostrar coneixement en el llenguatge de programació C++.
- S'ha utilitzat Unity3D per tal de composar prèviament les escenes i comprovar si realment quedaven bé o no, estalviant temps de desenvolupament.

Si bé s'han acomplert els objectius, s'han de fer certes consideracions respecte al desenvolupament del projecte:

- Utilitzar llibreries de tercers pot portar molts problemes: durant el desenvolupament del projecte s'ha donat el cas d'un atac informàtic als servidors de Nvidia. Aquest fet va provocar que les pàgines de suport al desenvolupament fossin clausurades durant un temps, fet que em va deixar sense accés, per exemple als fòrums de consulta.
- Les eines multi-plataforma molts cops no són tant multi-plataforma com hom esperaria. Si bé en teoria la llibreria emprada per a la gestió de l'entrada d'usuari (OOIS) hauria de funcionar correctament tant en Windows com en Mac OS X, a l'hora de la veritat ens hem trobat problemes, com ara el fet de no tenir versió pura de 64bits, resolt canviant la plataforma a 32bits, o com el fet de tenir problemes amb el

controlador de Xbox360, a causa del driver no oficial emprat, que han fet impossible fer funcionar el suport a aquest controlador en la plataforma Mac OS X.

- El desenvolupament del motor del joc, si bé ha servit per aprendre amb molta profunditat el funcionament tant de la llibreria gràfica (Ogre3D) com de la llibreria física (Nvidia PhysX), ha comportat molt temps de desenvolupament i test, fet que ha fet que el joc no pogués ser tant ampli com s'hauria desitjat en el moment de començar el projecte. Aquest és un fet que ja sabíem en el moment d'optar per aquest camí però que val la pena remarcar per tenir-lo en compte en projectes futurs.
- Les eleccions finals de les eines a utilitzar han estat molt encertades. Destacar especialment el bon funcionament d'Ogre i PhysX i la facilitat amb la que s'han pogut ajuntar per realitzar un treball comú. La no elecció de llibreries externes per realitzar aquesta unió ha estat del tot encertada, ja que ha ajudat a aprofundir en el coneixement d'aquestes dues llibreries i ha estalviat molts problemes que altre gent ha hagut de patir per fer servir segons quines llibreries de tercers. Destacar també l'excel·lent funcionament de la llibreria de so i de la llibreria emprada per a llegir fitxers XML. Dues llibreries amb una facilitat d'ús increïbles.

Pel que fa al treball futur, s'han detectat un seguit d'accions que farien millorar el projecte fi de carrera, enumerats a continuació:

- Ampliar el nucli jugable, afegint més nivells al joc.
- Introduir algun tipus de llenguatge de scripting al motor del joc. Aquesta acció faria possible, per exemple, implementar la intel·ligència artificial dels enemics en aquest tipus de llenguatges i estalviar, a la llarga, temps de desenvolupament gracies a la facilitat d'edició del codi.
- Buscar una altre llibreria per canviar la llibreria d'entrada d'usuari o, si no és possible, crear una llibreria pròpia que s'encarregui d'aquesta tasca.

Bibliografia

UDK: Motor Unreal development kit, http://www.unrealengine.com/udk/

UGINE: Motor Unigine, http://unigine.com

U3D: Motor Unity3D, http://unity3d.com

IRR: Llibreria Irrlicht, http://irrlicht.sourceforge.net

IRRK: Llibreria IrrKlang, http://www.ambiera.com/irrklang/

IRRE: Llibreria IrrEdit, http://www.ambiera.com/irredit/

O3D: Llibreria Ogre3D, http://www.ogre3d.org

OET: Llibreria Ogre speed tree, http://www.torusknot.com/ogrespeedtree.html

BLLT: Llibreria Bullet physics, http://bulletphysics.org/wordpress/

WKBLLT: Wikipedia sobre bullet physics, http://en.wikipedia.org/wiki/Bullet_(software)

PHSX: Llibreria Nvidia PhysX, http://developer.nvidia.com/physx

WKPHSX: Wikipedia sobre Nvidia PhysX, http://en.wikipedia.org/wiki/PhysX

FMOD: Llibreria Fmod, http://www.fmod.org

WKFMD: Wikipedia sobre Fmod, http://en.wikipedia.org/wiki/FMOD

FSLW: Llibreria FreeSLW, http://sourceforge.net/projects/freeslw/

SMFL: Llibreria SMFL, http://www.sfml-dev.org

OMAX: Etna Ogremax, http://congresswoman

CRY01: Ricard Pillosu, Scrum para estudios de videojuegos, http://www.youtube.com/watch?&v=sWx58uWBHI4

AC2012: Certamen AppCity 2012, https://www.facebook.com/appcity2012

AEU3D: Noticia sobre el concurs Unity3D, http://applesencia.com/2012/02/crear-un-videojuego-unity3d

TXML: Llibreria tinyxml, http://www.grinninglizard.com/tinyxml/

Annex 1

Diagrames

1. Diagrames classes comunes

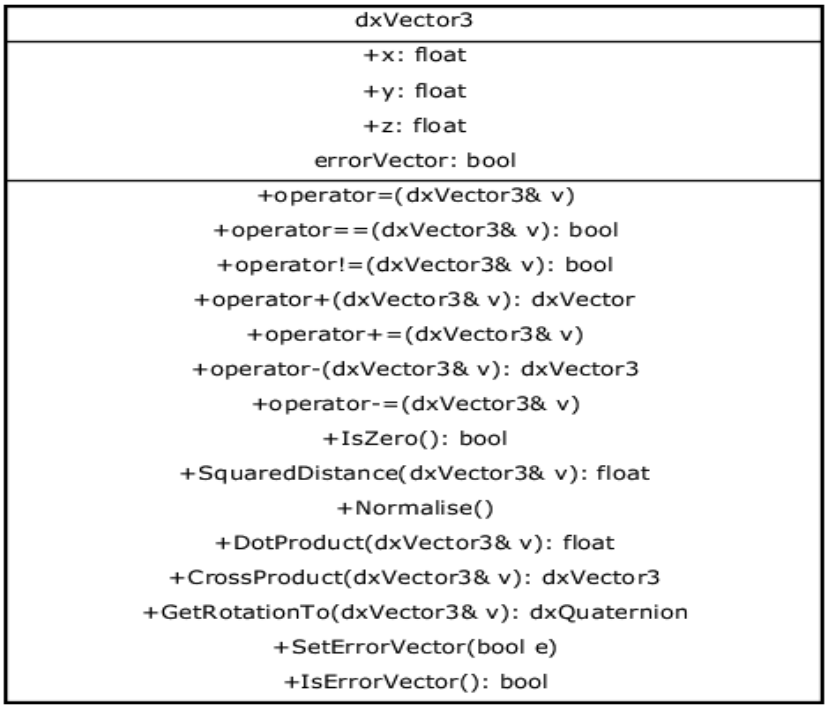


Diagrama 1: classe dxVector3

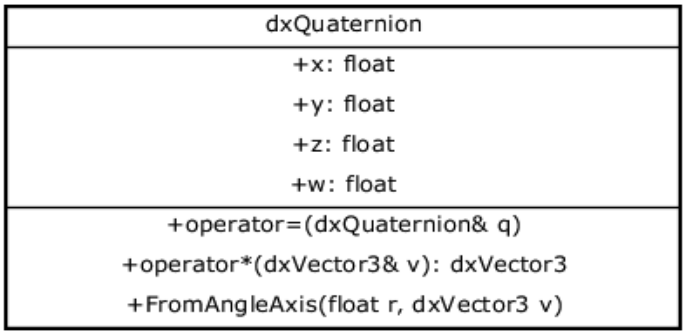


Diagrama 2: classe dxQuaternion

2. Diagrames sistema gràfic

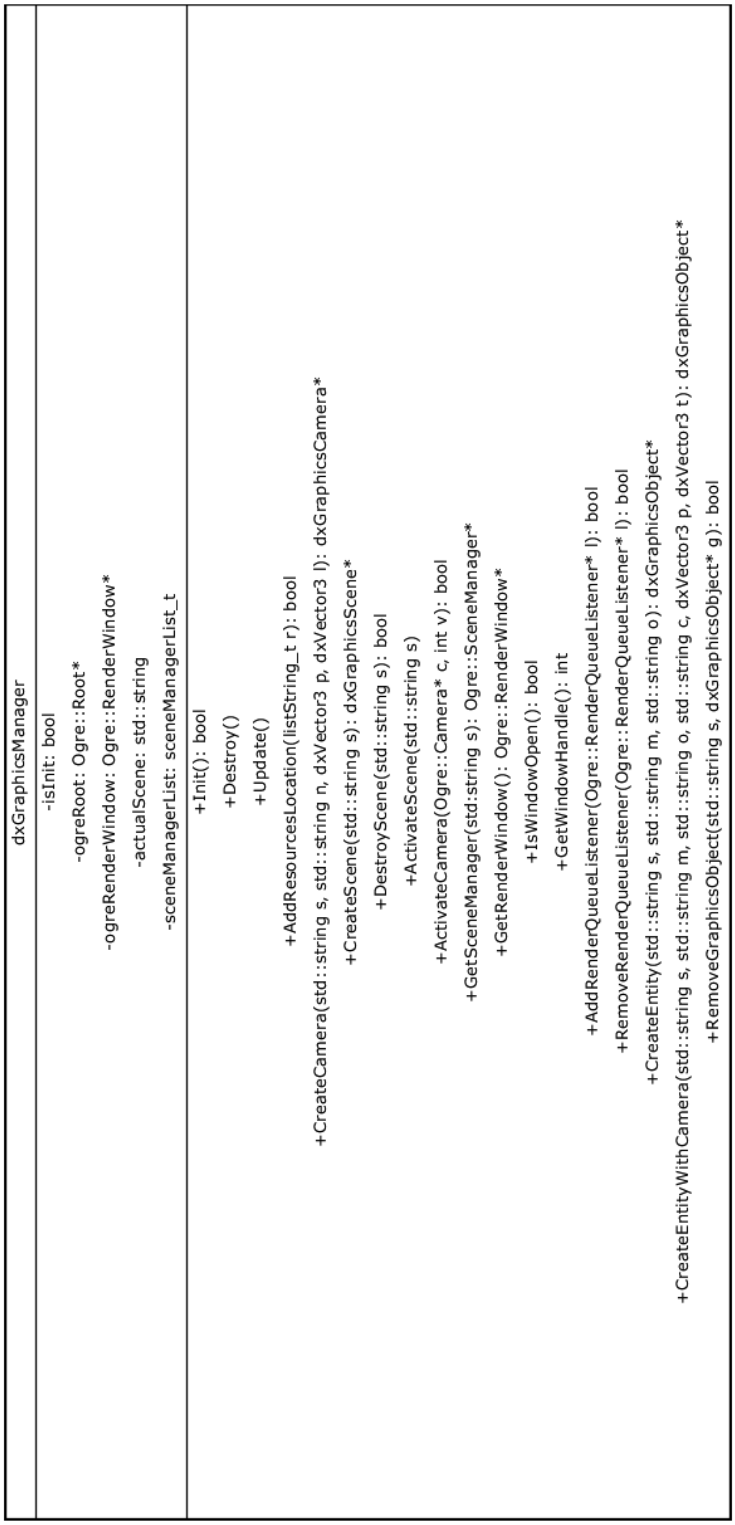


Diagrama 3: classe dxGraphicsManager

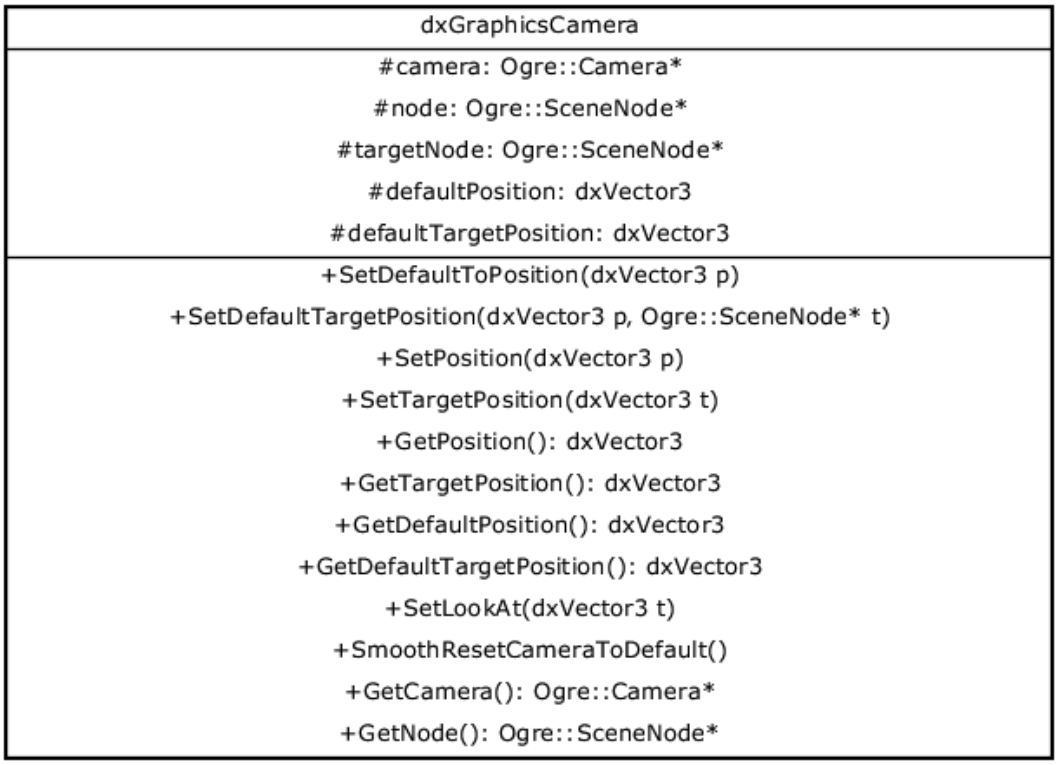


Diagrama 4: dxGraphicsCamera

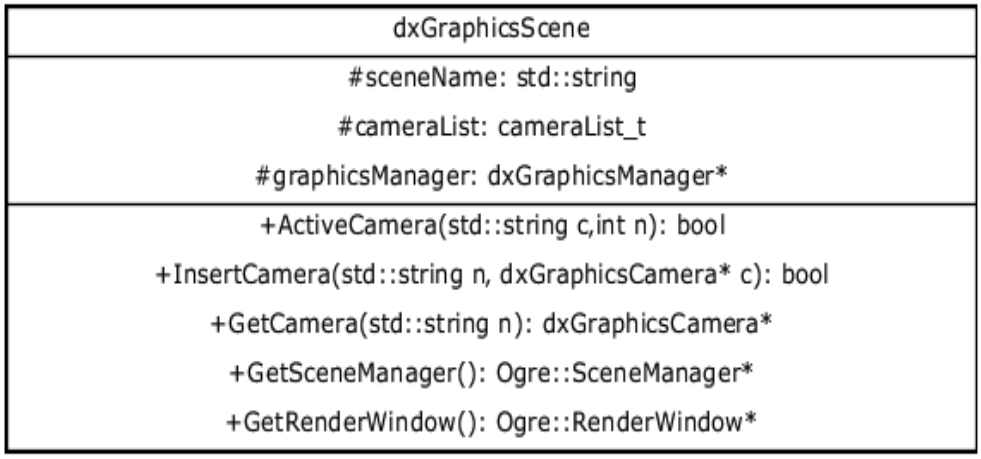


Diagrama 5: dxGraphicsScene

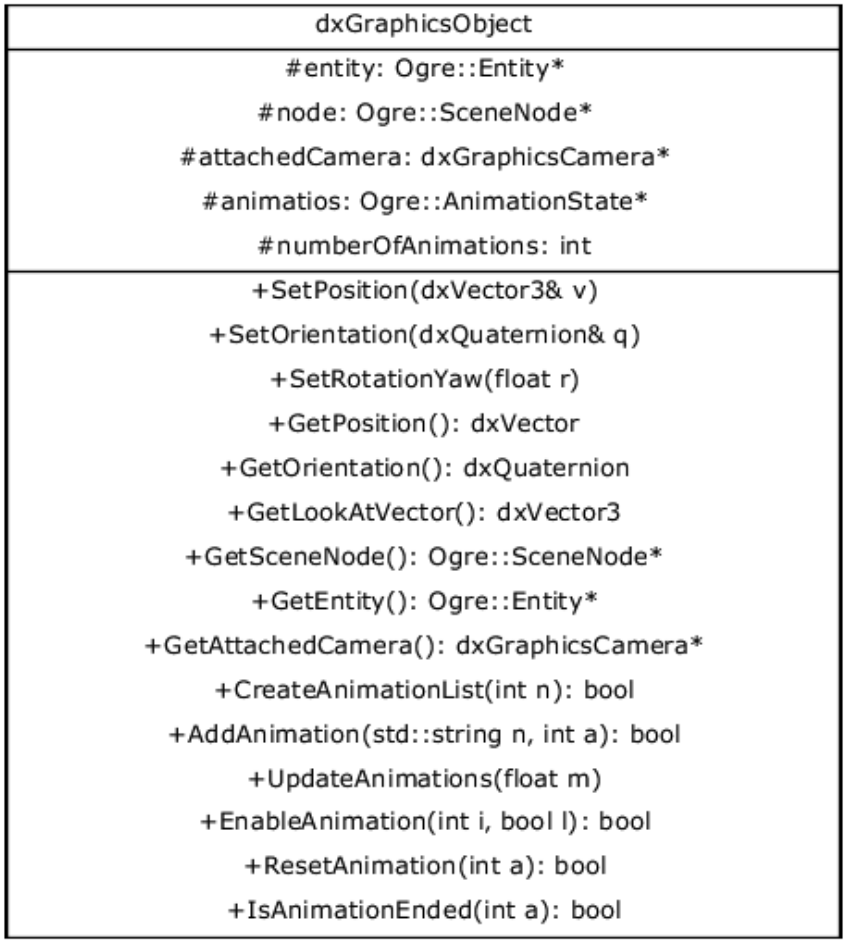


Diagrama 6: dxGraphicsObject

Diagrama 7: dxPhysicsManager

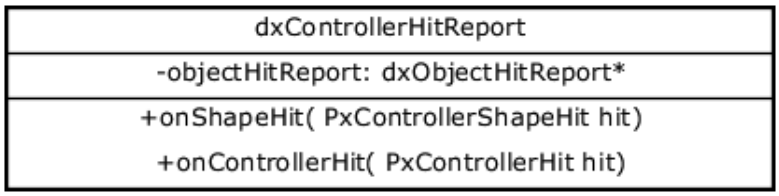


Diagrama 8: dxControllerHitReport

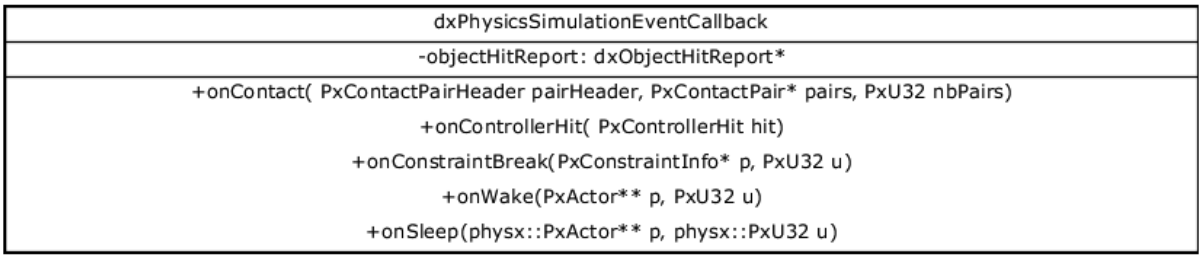


Diagrama 9: dxPhysicsSimulationEventCallback

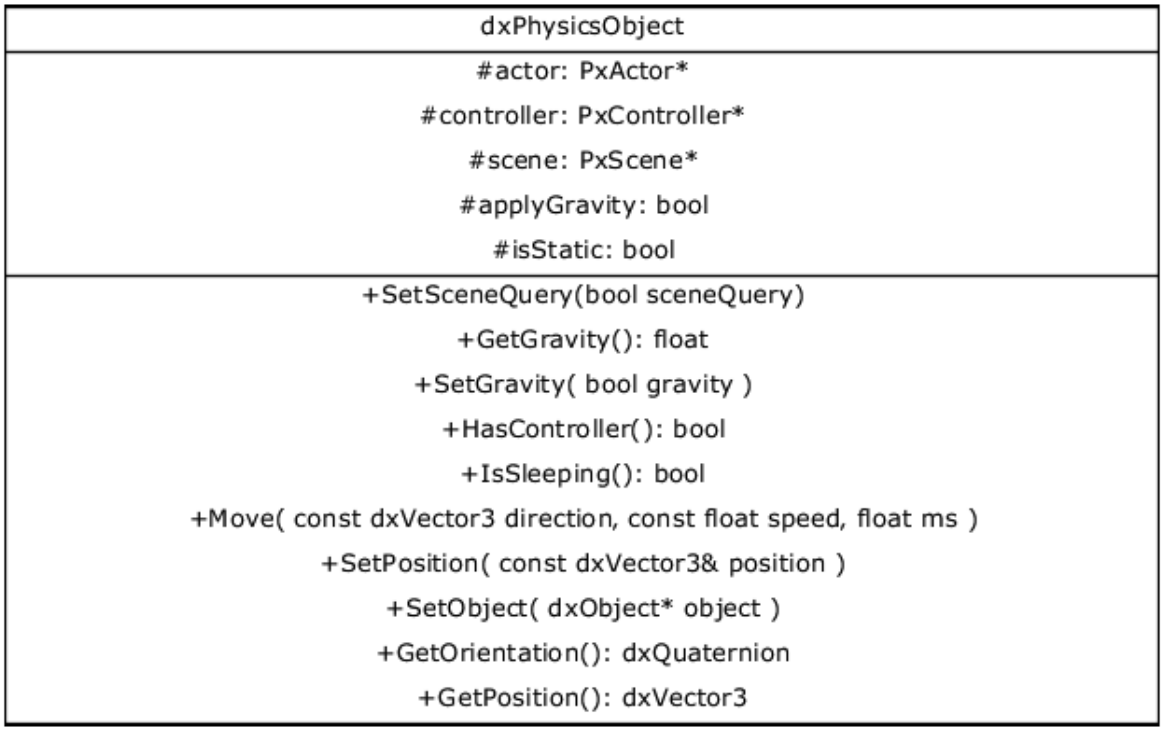


Diagrama 10: dxPhysicsObject

dxPhysicsScene
-scene: PxScene
+GetCollisionObject(dxVector3 origin, dxVector3 direction, float distance): dxObject* +GetCollisionPoint(dxVector3 origin, dxVector3 direction, float distance): dxVector3 +GetGravity(): dxVector3 +SetGravity(const dxVector3& gravity)

Diagrama 11: dxPhysicsScene

dxObject
#objectName: string #scene: dxScene* #physicsObject: dxPhysicsObject #graphicsObject: dxGraphicsObject #Synchronize()
+GetPosition(): dxVector3 +GetObjectName(): string +Update() +TriggerActivated(dxObject* ab) +CollisionDown()

Diagrama 12: dxObject

dxScene
#gravity: dxVector3 #objectTypeList: objectTypeList_t #objectList: objectList_t #graphicsScene: dxGraphicsScene* #physicsScene: dxPhysicsScene* #gameEngine: dxGameEngine*
+GetGameEngine(): dxGameEngine* +GetDebugInfo(): sceneDebugInfo_t +GetCollisionDown(const dxVector3& o, const dxVector3& d): dxObject* +GetCollisionPoint(const dxVector3& o, const dxVector3& d): dxVector3 +GetObjectsFromGroup(string g): dxObjectList +GetObject(string n): dxObject* +SetSceneGravity(dxVector3 g) +GetSceneGravity(): dxVector3 +InsertObject(dxObject* o, string g) +Update()

Diagrama 13: dxScene

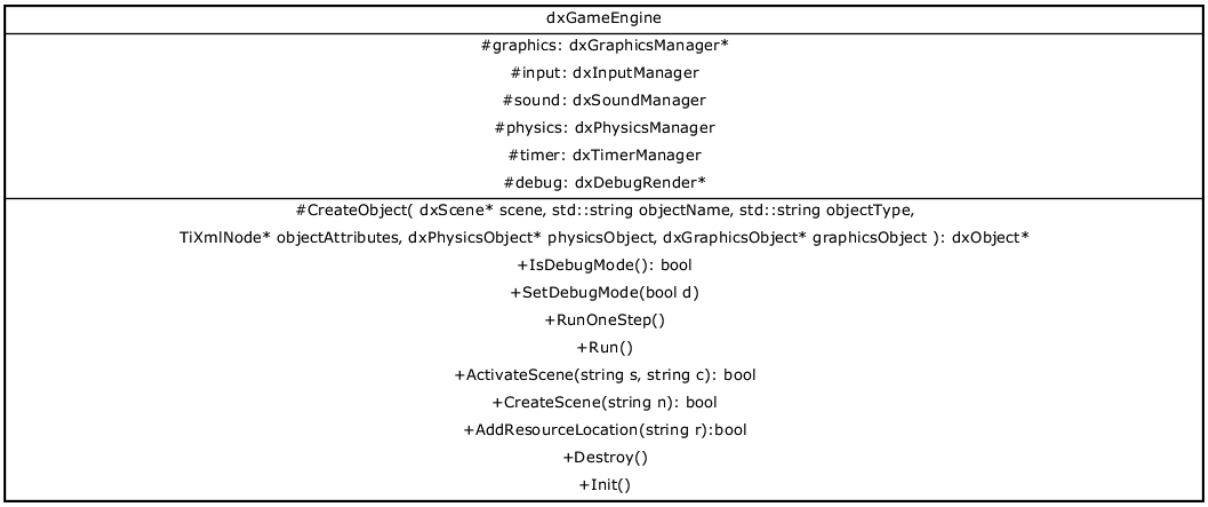


Diagrama 14: dxGameEngine

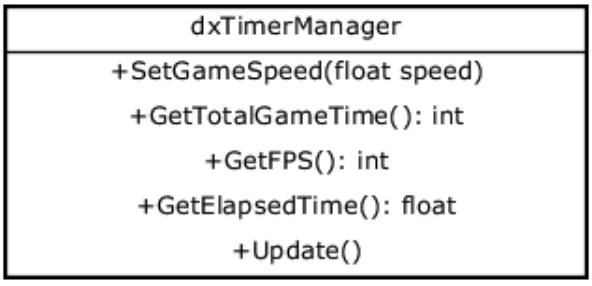


Diagrama 15: dxTimerManager

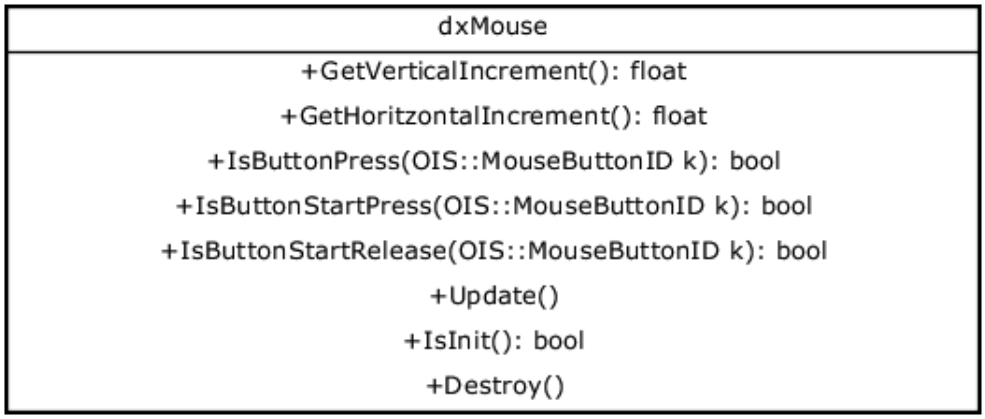


Diagrama 16: dxMouse

Diagrama 17: dxKeyboard

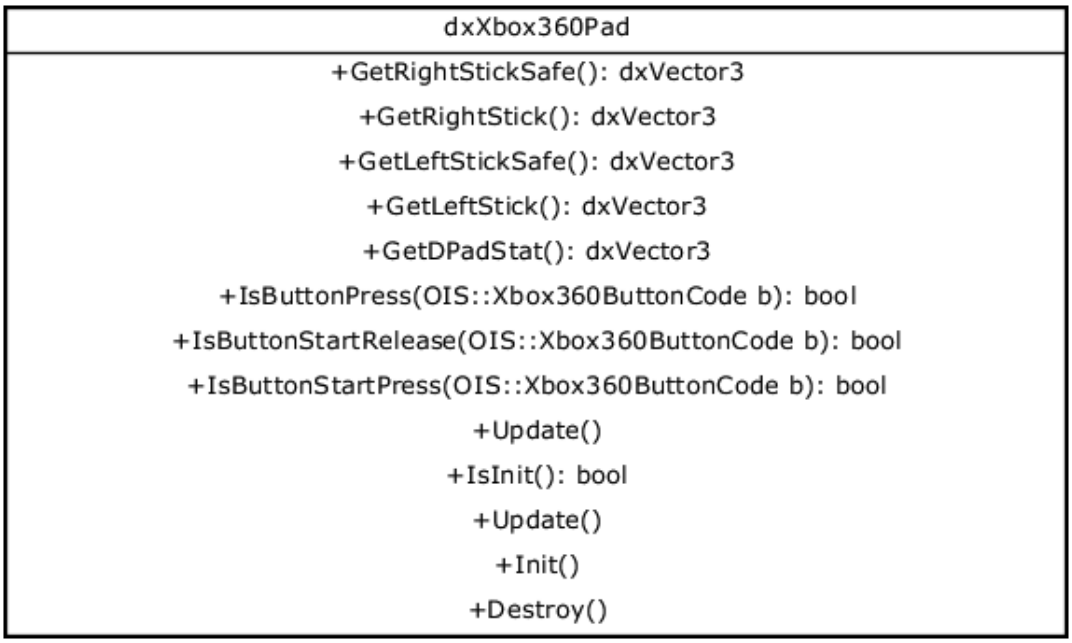


Diagrama 18: dxXbox360Pad

Diagrama 19: dxInputManager

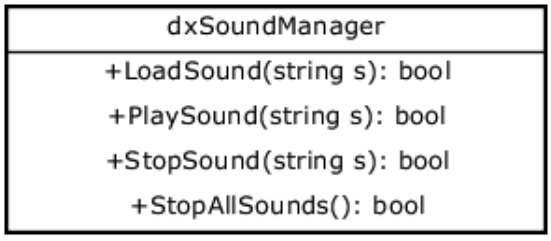


Diagrama 20: dxSoundManager

Disseny i implementació d'un videojoc 3D tracta sobre el desenvolupament d'un videojoc, des dels seus inicis amb la creació del document de disseny fins al punt final, en el moment de tenir el joc acabat. En aquesta memòria veurem tot el procés, com s'han escollit les llibreries utilitzades, quines implicacions han tingut aquestes eleccions, com s'ha dissenyat el motor de joc, quines funcionalitats aporta, quins problemes han aparegut durant el procés de desenvolupament del joc i quin ha estat el resultat final.

Diseño e implementación de un videojuego 3D trata sobre el desarrollo de un videojuego, des de sus inicios con la creación del documento de diseño hasta su punto y final, momento en el que tenemos el juego terminado. En esta memoria veremos todo este procesos, cómo se han escogido las librerías utilizadas, qué implicaciones han tenido estas elecciones, como se ha diseñado el motor del juego, qué funcionalidades ha aportado, qué problemas han aparecido durante el proceso y cuál ha estado el resultado final.

Design and implementation of a 3D videogame discusses about videogame development, from it's beginnings with the game design document to it's end point, when game have been finished. In this project you will see all of this process, what libraries has been chosen and why, how the game engine has been designed, how it works, what problems have appeared during this process and what has been the final result of the project.