DISSENY I DESENVOLUPAMENT DE VIDEOJOCS

Memòria del projecte
d'Enginyeria Informàtica
realitzat per
Joan Amat Iglesis
i dirigir per
Enric Martí Gòdia i
Jordi Arnal Montoya
Bellaterra, 25 de juny de 2009

El sotasignat, Enric Martí Gòdia

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 Joan Amat Iglesis.

I per tal que consti firma la present.

Signat:

Bellaterra, de de 2009

El sotasignat, Jordi Arnal Montoya

Professor del curs de programació de videojocs de la UAB

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Joan Amat Iglesis.

I per tal que consti firma la present.

Signat:

Bellaterra, de de 2009

Taula de continguts

1. Introducció 5
    1.1 El prototip 5
    1.2 Proposta de solució 7
    1.3 Objectius 8
2. Desenvolupament 10
    2.1 Metodologia 10
    2.2 Estructura 12
3. Resultats 17
    3.1 Joc de bitlles 18
    3.2 Conducció 19
    3.3 Estratègia 21
    3.4 Acció en primera persona 22
    3.5 Perspectives de continuitat 24
4. Conclusions 25

Referències bibliogràfiques 28

Annex 1: Manual de l'usuari 32
Annex 2: Diagrama de classes UML 35
Annex 3: Codi font dels jocs creats 37

1. Introducció

La indústria del videojoc ha crescut enormement els últims anys. Als Estats Units el volum de negoci va superar al de la indústria musical el 2005, i el 2007 al de la cinematogràfica [Ban08].

En aquest sector l'Enginyeria del Software pren un paper encara més rellevant que en la resta de productes software, no només per la proporció comercial que el sector ha adquirit sinó per la dificultat afegida d'estar treballant amb un producte artístic.

L'objectiu d'un videojoc comercial és satisfer al públic per a obtenir vendes. L'entreteniment obtingut amb ell és un concepte subjectiu i, per tant, difícil de mesurar i avaluar. La mecànica del joc serà el que faci que el joc agradi. Per tant, el seu disseny és un punt del procés amb importants conseqüències. L'èxit de públic és, en gairebé tots els casos, l'únic factor d'èxit comercial. El resultat d'aquest fenòmen és que, mancats de les mètriques i sistemàtiques que l'Enginyeria del Software requereix, la llista de jocs inacabats creixi cada any [Unseen64].

Una de les etapes inicials en la creació d'un videojoc és fer-ne el disseny, que determinarà en què consistirà la mecànica del joc. Aquesta mecànica crea les normes del joc, que és el que finalment farà que sigui divertit i que vinguin ganes de jugar-hi. Essent el concepte de diversió completament subjectiu, aquesta fase no només determina un aspecte crucial del resultat final i ho fa en una de les primeres etapes de la creació, sinó que també té un objectiu difús com és produir satisfacció al jugador.

Per a aquest problema existeix una solució clàssica de l'Enginyeria del Software; provar aviat, però veurem que aquesta solució no es pot traslladar directament al món del videojoc. No podem aïllar els mòduls que acompleixen l'objectiu de divertir o satisfer l'usuari i provar-los per separat. Cal un enfocament diferent al concepte de proves, cal poder provar el producte final en l'etapa de disseny abans de construir-lo, és a dir, prototipar.

En aquesta introducció s'exposa la creixent necessitat del sector de prototipar i s'avalua la problemàtica inherent al codi d'un sol ús que en resulta. La solució proposada al punt 1.2 s'ha implementat respectant els objectius i metodologia descrits als punts 1.3 i 2.1 respectivament.

1.1 El prototip

El prototipatge ha pres rellevància en els últims anys com a forma de millorar la fase de disseny per a aconseguir resultats més sòlids. Gràcies a un bon prototip, un dissenyador pot emprendre la creació del joc amb la certesa que el projecte no tant sols sembla bo sobre el paper sinó que l'ha jugat –o més aviat simulat– per a comprovar-ho. Això també suposa una ajuda per a especificar millor les necessitats tècniques del projecte, reduint així la probabilitat d'un canvi d'objectius tardà i repercutint directament en el risc inherent d'un projecte llarg.

Un prototip és, per tant, una senzilla simulació del joc on s'ometen els factors que no influeixen en la mecànica. En general els personatges i els objectes podran ser cubs o esferes i els

escenaris tant senzills com una habitació buida. També es podrà prescindir de tots els elements de presentació i de realisme gràfic i sonor que tant decisius són per a un videojoc.

La cultura del prototipatge, encara incipient, està irregularment estesa entre les companyies del sector. Una de les més importants, Maxis –creadora del joc d'ordinador més venut de la història, The Sims– considera el prototipatge un pilar sobre el qual no només es prova un joc sinó que també és el millor suport per a comunicar idees.

Més enllà d'una eina per al disseny, és una eina de comunicació entre treballadors. Enquadrat dins la filosofia de la programació extrema, el seu director de desenvolupament en cap Eric Todd veu el prototip com la millor forma de comunicar interactivitat, un concepte, diu, molt difícil de descriure en paraules i imatges [Wau06].

En la figura 1 s'observen captures de pantalla de l'últim joc de Maxis, Spore. A l'esquerra podem veure un dels primers prototips, un fet inusual que ens permet veure com es prescindeix dels elements decoratius i només s'hi inclouen els elements mínims necessaris per a simular el joc. En aquest cas es tracta d'un món amb vàries criatures, menjar i uns indicadors de gana, energia i salut. La comparació amb la captura de la dreta permet, amb una mica d'imaginació, traçar els canvis que va sofrir el projecte i constatar que es tracta, clarament, del mateix joc.

Figura 1. Contraposició de l'aspecte del videojoc Spore amb una instantània d'un dels seus prototips

El prototip pot variar molt en ambició i propòsit. Una companyia desenvolupava jocs per a iPhone prototipant-los amb trossos de cartrons que simulaven la pantalla [Mar08]. No només era una forma barata i ràpida, també era la l'única possible perquè Apple encara no havia fabricat cap telèfon. Els jocs havien d'estar apunt per al llançament de l'aparell.

La conclusió és que en tots els casos el mètode consisteix en crear ràpidament la tecnologia mínima necessària per a el que vol ser provat. En un gran projecte el programador ajunta codi amb la màxima rapidesa sense usar temps en documentar el programa o organitzar-lo bé, doncs un cop hagi estat provat tot el codi produït es llença. Un prototip no és la base per a la feina futura sinó un esbós

que no es reutilitza, concepte ben explicat per Julian Spillane, CEO de Frozen North, en un article de Mathew Kumar per a Gamasutra [Kum07].

Vist el funcionament real del prototipatge en la indústria s'observa una contradicció entre el que és productiu i el que és la bona programació des d'un punt de vista més acadèmic. Aquests prototips resulten ser difícilment continuables si més endavant es decideix provar quelcom similar, molt més encara si el creador no està disponible per a repassar el seu codi. El bon codi, planificat i documentat no és ràpid de crear, així doncs, es presenta la necessitat d'un compromís entre els dos extrems.

Aquest projecte estudia la implementació d'una **plataforma programable mitjançant la qual es pot construir ràpidament un videojoc de forma esquemàtica**. El benefici pretès és el de simular de manera senzilla el resultat final del joc per tal de dotar la fase de disseny d'eines concretes que produeixin resultats mesurables.

1.2 Proposta de solució

L'objectiu d'aquest projecte és trobar un compromís entre els objectius clàssics de l'Enginyeria del Software de produir codi reutilitzable i ben documentat i la naturalesa ràpida que hem vist dels prototips. La clau d'aquesta proposta rau en dividir el prototip en dues parts. Com seguidament veurem, certes funcions són comunes a quasi tots els videojocs ja que en modelen el suport tecnològic. D'altres en canvi es centren en implementar la mecànica del joc. Les primeres les podrem desenvolupar d'una forma sòlida, separant així els aspectes tècnics de la part creativa que correspon directament al disseny del joc. Aconseguirem, doncs, que la part del codi més reutilitzable estigui ben estructurada i documentada, mantenint la rapidesa i flexibilitat que el dissenyador necessita a l'altre extrem de l'aplicació.

Podem abstraure que un videojoc és un sistema que rep les instruccions del jugador per respondre'n amb les seves conseqüències. La consistència del món s'aconsegueix a mode de màquina d'estats on, donades aquestes instruccions i un estat del món, la màquina crea el nou estat del món. Per a simular un joc en temps real la màquina repeteix continuament el procés que es reprodueix a la figura 2.

L'entrada per part del jugador i el dibuix per pantalla dels resultats obtinguts són un element comú en tots els videojocs, per tant els podrem situar a la base del nostre projecte. De la mateixa manera, tots els videojocs necessiten una base de dades que contingui l'estat del món. La mecànica del joc són les normes que dicten com s'actualitza aquest estat depenent de les instruccions del jugador.

Aquests elements comuns poden ser abstrets i programats d'una forma sòlida i permanent. Una base a partir de la qual es podran construir els prototips creant només les funcions que regulen el joc en si mateix i que trobem localitzades a l'esquema.

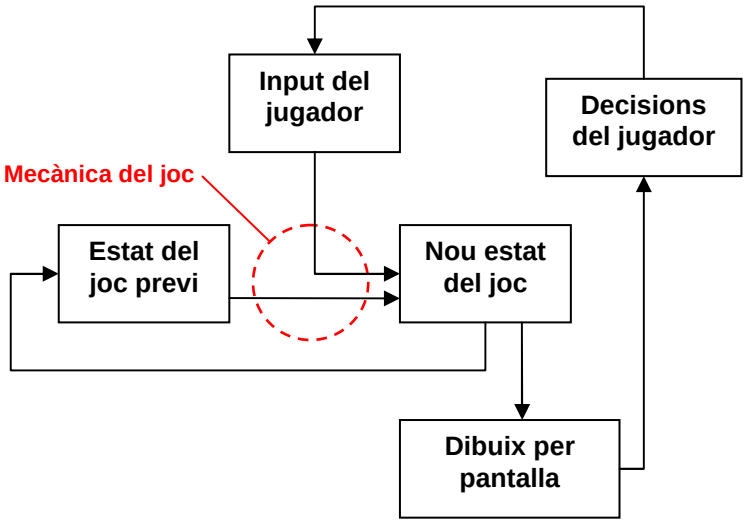


Figura 2. Localització dels càlculs de la mecànica del joc fins l'esquema comú de tots els videojocs

1.3 Objectius

El producte d'aquest projecte serà dividir aquestes dues parts de manera que en resulti una base tecnològica que sigui programable mitjançant llenguatge de script que modelarà tant sols la mecànica del joc, resolent així la dicotomia exposada als punts anteriors. El programa, per tant, és una plataforma de prototipatge de videojocs.

Serà important poder simular la mecànica d'un joc de forma ràpida, mitjançant un llenguatge d'alt nivell i sense necessitat de compilar o administrar memòria. També caldrà que la plataforma sigui prou flexible per a adaptar-se al màxim de gèneres possibles.

Aquesta plataforma de prototipatge haurà de constar de tres mòduls bàsics dels videojocs; un gestor de l'entrada de dades per part del jugador, un motor gràfic capaç de mostrar els objectes per pantalla i un motor físic que representi els objectes que hi existeixen i les seves interaccions.

En segon lloc, el funcionament del joc s'haurà de llegir des d'un llenguatge de script d'alt nivell que se situarà fora del programa principal, de manera que aquesta execució es faci en temps real. Així sacrifiquem rendiment en favor de flexibilitat i no haurem de compilar codi.

Sobre aquesta base s'incorporaran les funcionalitats que l'usuari pugui necessitar per a construir-hi la mecànica del joc. Aconseguida la solidesa, ens centrarem en els objectius de rapidesa i productivitat.

- **Gestor d'entrades:** En tractar-se de les instruccions donades per un humà, aquestes són, des del punt de vista de la màquina, molt lentes i esporàdiques. Caldrà només acumular-les en un *buffer* per a poder-ne consultar l'estat periòdicament. Les necessitats tècniques, en aquest cas, són mínimes.
- **Motor gràfic:** Gràficament no farà falta vistositat sinó un motor que escali bé. Les nostres escenes seran senzilles i no necessitaran les tecnologies innovadores que la majoria dels

creadors valoren. En canvi, haurà de ser eficient perquè tot el que guanyem en rendiment amb la senzillesa ho perdem perquè gran part del nostre codi estarà externalitzat i no compilat.

- **Motor físic:** Al tractar-se d'una part vital de la mecànica del joc, cal que les interaccions físiques siguin precises i realistes. De no ser així el joc que juguem ara podria no ser el mateix que el que es construeixi en el futur. Cal que el motor físic sigui de primer nivell, el mateix que utilitzariem en una producció definitiva.
- **Llenguatge extern:** Per a especificar la mecànica del joc utilitzarem un llenguatge script. És important que incorpori totes les operacions habituals en els llenguatges de programació i que sigui extensible mitjançant llibreries. La interacció amb el codi que el crida haurà de ser senzilla i permetre l'intercanvi d'estructures de dades entre els dos.
- **Funcionalitats a incorporar:** Caldrà que el creador tingui a la seva disposició un conjunt d'objectes bàsics –cubs i esferes– que pugui crear i manipular en l'espai mitjançant operacions algebraiques. Hauran d'estar disponibles totes les transformacions habituals en el món dels gràfics i els videojocs, de manera que caldrà conservar íntegrament la representació matemàtica del món per a poder usar funcions algebraiques i trigonomètriques. La càmera mostra allò que el jugador veu i cal poder manipular-la de múltiples maneres. Aquest és un punt fonamental per a garantir la flexibilitat, ja que els gèneres del videojoc s'acostumen a classificar segons la posició i comportament de la càmera, classificació que s'explica amb més detall al punt 2.1.2 d'aquesta memòria.

El capítol dos mostra el desenvolupament del projecte i, a part de la metodologia, també la seva estructura des de diferents punts de vista, essent un programa destinat a un doble ús per part del dissenyador.

Els resultats obtinguts es mostren al capítol tres tot intentant mostrar el producte de forma mesurable. Com ja s'ha dit, la mecànica d'un joc és difícil de comunicar; per tant certs aspectes s'intentaran mostrar a través de descripcions i de captures de pantalla.

El capítol quatre resumeix les conclusions tècniques i personals del projecte.

Al final de la memòria s'adjunten tres annexos. El primer dels quals és un manual d'ús de la plataforma creada, el segon mostra els mòduls i estructura del programa en un esquema UML i el tercer és una recopilació d'exemples de codi Lua que defineixen la mecànica de diferents jocs.

2. Desenvolupament

El desenvolupament del projecte s'ha dut a terme en dues fases diferenciades. En primer lloc calia ajuntar les diferents tecnologies per a crear un sistema capaç de lligar els tres pilars fonamentals del projecte, el motor gràfic, el físic i la vinculació amb el llenguatge extern. Després calia posar a prova l'entorn creant diferents prototips que en posessin a prova les capacitats de manera iterativa i incremental.

2.1 Metodologia

L'abast del projecte serà la implementació en l'ordinador tenint en compte tant sols les interfícies humanes típiques de la plataforma, el teclat i el ratolí. El programa, escrit en C++, incorporarà els sistemes bàsics del joc identificats anteriorment.

2.1.1 Posar en ús la tecnologia

Usarem llibreries públiques per a crear els elements necessaris; un motor gràfic realitzarà el dibuixat en pantalla i un motor físic representarà el món i gestionarà les interaccions entre objectes, un tercer mòdul s'encarregarà d'exportar funcions al llenguatge extern Lua.

- **El motor gràfic:** Hem triat el motor gràfic OGRE 3D [Ogre3D] per el seu disseny sòlid i la seva jerarquia de classes. OGRE es publica sota llicència LGPL (Lesser General Public License) i el seu bon disseny el fa molt estructurat, cosa que permet prescindir fàcilment de moltes de les capacitats gràfiques més punteres que incorpora. OGRE és un motor gràfic genèric no específic per a videojocs i no incorpora cap altre funcionalitat més enllà del dibuix per pantalla.
- **El motor físic:** PhysX [PhysX] és un sistema propietari de Nvidia –important fabricant de processadors gràfics i, degut a aquest producte, ara també fabricant de processadors físics per hardware–, el SDK (Software Development Kit) del qual té una llicència gratuïta. Es tracta d'un conjunt de llibreries de processament d'interaccions físiques usat en molts dels títols més importants per a ordinador i consola. En conjunt es tracta d'un software molt potent i ben documentat.
- **El llenguatge de script:** Un dels llenguatges de script més usat en el món dels videojocs és Lua [Lua]. És un llenguatge lleuger i funcional dissenyat amb l'objectiu de ser un llenguatge de script i, per tant, amb una bona API (Application Programming Interface) per a C. Tot i això Lua és limitat per el que fa a la interconnexió amb l'aplicació. Per a millorar aquest aspecte usarem les llibreries Luabind de Rasterbear software, que permeten exposar funcions i classes de C++ directament a Lua.

2.1.2 Implementació de les funcionalitats

Per a mesurar el compliment dels objectius de rapidesa i flexibilitat que ens hem fixat, hem planificat el prototipatge de varis jocs de complexitat creixent i de gèneres diferents.

La classificació dels gèneres dels videojocs és un tema subjecte a debat. Tot i una certa tendència inicial cap a una classificació similar als gèneres novel·lístics i cinematogràfics [Wol01], actualment la indústria i el públic identifiquen el gènere segons la relació que el jugador estableix amb els elements que controla. Els elements definidors d'un gènere seran, majoritariament, la posició de la càmera relativa als elements que el jugador controla i el tipus de control que hi estableix.

Aquesta classificació és una llista oberta que molts jocs intenten desafiar amb propostes difícilment classificables. La decisió ha estat centrar-nos en alguns dels gèneres més habituals segons criteris d'ortogonalitat per a assegurar-ne la màxima varietat.

- **Joc de puzle:** Crearem un senzill joc de bitlles consistent en tombar un conjunt d'objectes llançant-ne un altre des de la càmera, que serà immòbil però que el jugador podrà canviar-ne l'orientació lliurement amb el ratolí. Aquest joc servirà per a provar les funcionalitats bàsiques del sistema:
  - La creació d'objectes en pantalla: plans, cubs i esferes.
  - El dibuixat de les interaccions físiques: la cohesió entre els motors físic i gràfic
  - La capacitat del sistema de cridar al llenguatge extern i de rebre les seves instruccions.
- **Joc de conducció:** Crearem un joc de conducció on el jugador dirigeixi un objecte al voltant d'un circuit. El repte en aquest cas és crear una càmera de bord que segueixi l'objecte mirant en tot moment en la direcció adequada. De la mateixa manera, caldrà implementar un sistema de creació d'entorns ràpid per poder provar situacions complexes. Cal anar més enllà del pla infinit del joc anterior i poder crear ràpidament un entorn complex com el d'un circuit de carreres.
- **Joc d'estratègia:** Essent un gènere totalment diferent als anteriors, els jocs d'estratègia solen caracteritzar-se per una càmera zenital i un sistema d'unitats on es controlen varis personatges en paral·lel. Haurem de poder seleccionar objectes assenyalant-los amb el ratolí i moure'ls per el món a voluntat. Es tracta de poder modelar el món a través de paràmetres lògics més allunyats de la representació matemàtica dels vectors i les seves transformacions.
- **Joc d'acció en primera persona:** Per a simular la primera persona el jugador esdevé un objecte més del joc però aquest no es mostra en cap moment. Així doncs, caldrà que la càmera tingui l'aparença de respondre a les lleis físiques tot i no formar part del motor físic. A més, els trets del jugador impactaran en punts ben localitzats dels objectes i per tant caldrà aplicar forces locals als objectes i no en el seu centre de gravetat com feiem fins ara.

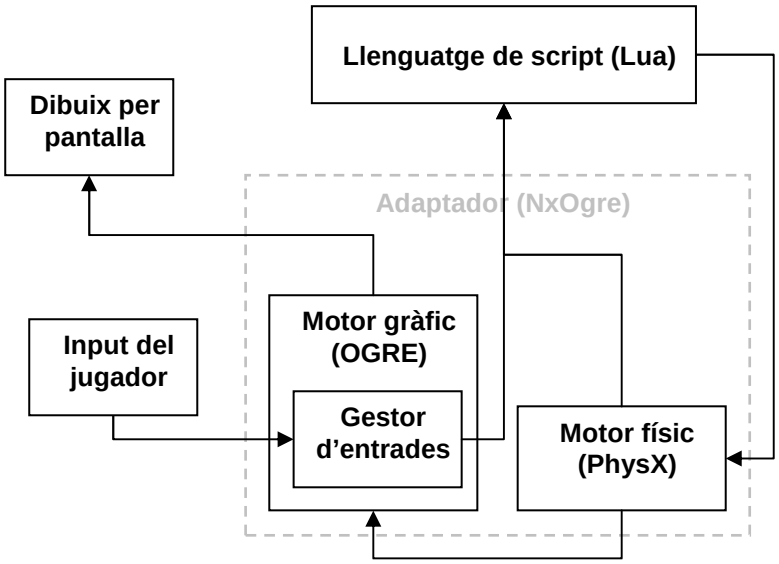


Figura 3. Detall dels mòduls que gestionen els passos de l'esquema de la figura 2

2.2 Estructura

Tractant-se d'un projecte que divideix un procés en dues parts tal com ja hem dit, n'explicarem l'estructura des dels diferents punts de vista implicats. L'estructura bàsica d'aquest es mostra al diagrama de la figura 3 que n'evidencia la connexió amb l'esquema genèric explicat a la introducció. En endavant abstraurem aquesta seqüència i la tractarem com a un bucle que calcula constantment la nova situació del joc.

El punt 2.2.2 explica el funcionament des del punt de vista de l'usuari –un dissenyador de videojocs– que usarà el programa en forma de caixa negra, el llenguatge Lua i el manual d'usuari de l'annex 1 són les seves eines de treball.

D'altra banda tenim un nucli que, segons les necessitats, el dissenyador en cap pot voler modificar. En aquest cas s'explica també el funcionament intern del programa, descrit en el punt 2.2.3 al qual li correspon l'annex 2 que en mostra en detall les classes i mètodes.

2.2.1 Funcionament general

L'adaptador que coordina el motor físic i gràfic és NxOgre. El seu objectiu és assegurar que estan perfectament sincronitzats, és a dir que no hi ha una sola realitat sinó dues d'idèntiques: el motor gràfic dibuixa els resultats de les interaccions que el motor físic ha calculat. La figura 4 mostra els moments del cicle en els que les tasques a realitzar vindran dictades per la mecànica del joc. Llavors és quan el bucle permanent de càlcul del nou estat del món s'interromprà per a sortir fora del programa per a llegir la mecànica programada en el llenguatge de script extern. Hi ha quatre ocasions en que està previst que això passi:

- **En cas d'input del jugador:** Segons el jugador premi el teclat o usi el ratolí es cridaran les funcions *MouseMoved(), MouseClicked()* o *KeyClicked()*. Aquestes funcions determinaran les conseqüències en el joc d'aquestes instruccions. Cada una de les crides portarà els arguments

pertinents per indicar quina tecla s'ha activat i si s'ha premut o deixat anar. En el cas *MouseMoved()* indicarà la quantitat de moviment l'eix *x* i *y*.

- **A cada nou frame:** Cada cop que es repeteixi el cicle, es cridarà la funció *FrameStarted()*, en la qual es materialitzaran totes les accions constants. Juntament amb aquesta crida passarem l'important paràmetre *elapsed*, que indica el temps que ha passat des de l'últim cicle, permetent així adequar les nostres transformacions per a que siguin constants al temps i no al nombre de cicles per unitat de temps.
- **A l'inici del joc:** Després de crear els elements interns de representació del món buits, es crida la funció de Lua *Startup()*. Allà s'indicaran les accions a prendre abans de començar el joc, que en general seran definir el món, gravetat, terreny, objectes inicials i posició de la càmera. Aquesta crida no porta arguments.
- **Activació de rellotge:** El control del temps és un factor comú en quasi tots els videojocs. Moltes accions s'activen per temps i quasi bé tots els videojocs tenen algun tipus de pressió en el temps per a completar els objectius. Per això s'ha inclòs aquesta funcionalitat, que permet al programador indicar que es cridi una certa funció que ell indiqui al cap d'una quantitat de temps determinada.

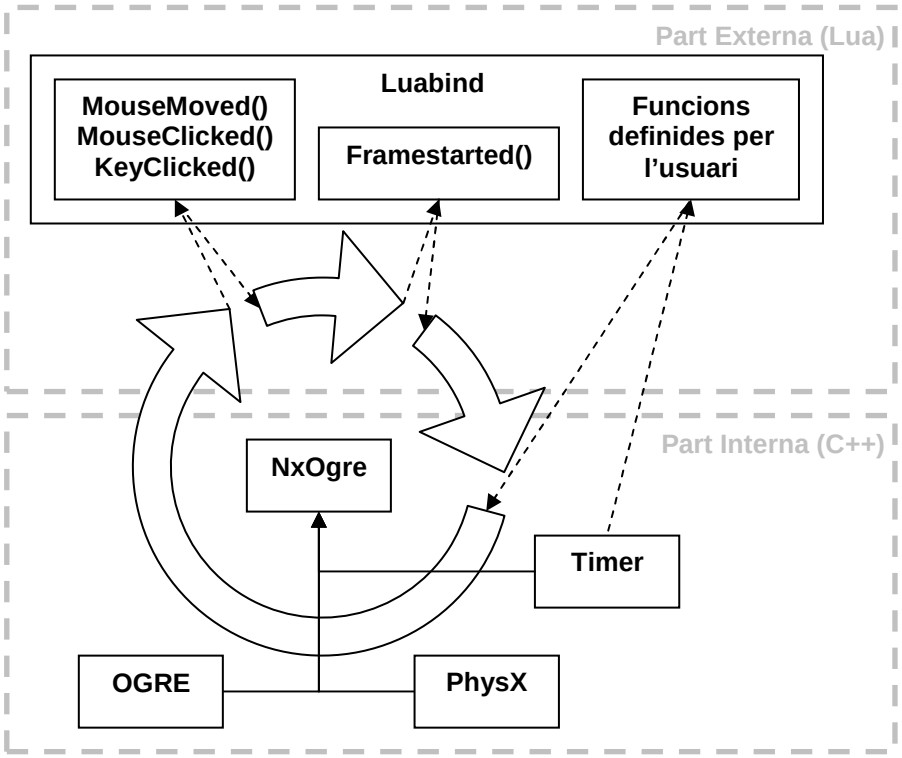


Figura 4. Interrupcions al funcionament normal del joc que en dicten la mecànica.

2.2.2 Funcionament des del punt de vista de l'usuari

Donats els punts d'entrada citats anteriorment, l'usuari escriurà el seu codi en base al diagrama de flux de la figura 5. En cada una d'aquestes situacions es programarà la resposta del joc retornant les instruccions que modelaran la mecànica del joc. Les operacions disponibles –descrites amb detall al manual d'usuari a l'Annex 1– s'engloben en sis grans grups:

- **Funcions matemàtiques:** Un vector, a part de permetre les operacions normals de suma i resta entre vectors i multiplicació i divisió escalars, es pot operar amb un altre per obtenir-ne el producte vectorial, normalitzar-lo o calcular-ne la magnitud.
    - *Vec = V(1,2,3)* és el constructor d'un vector amb coordenades x = 1, y = 2 i z = 3.
    - *Normalize(Vec)* retorna el vector anterior normalitzat.
- **Objectes:** Els elements visibles del joc són una sèrie d'objectes que l'usuari pot crear. Hi ha tres tipus d'objectes, Cub, Esfera i Pla. En base són cubs, esferes i plans, que seran la representació que el motor físic tindrà en compte, tot i que podem visualitzar-los en forma i textura qualsevol. El funcionament de tots ells és força similar al del cub:
    - *C = Cub(V(1,2,1), 10)* és un cub escalat per a tenir el doble d'alçada i massa 10.
    - *C:material("Examples/Rocky")* s'aplicarà la textura d'una pedra.
    - *C:mesh("barrel.mesh")* es dibuixarà segons la malla indicada aquí.
    - *C:crear("cub_ 1",V(0,0,0))* crea una instància del cub C en l'origen amb el nom "cub_1"
- **Les funcions físiques:** Permeten accedir i influir en la representació del món, conèixer o fixar la posició i velocitat dels objectes i aplicar força a aquests ja sigui al centre de gravetat o en un punt local o també força de rotació. També es poden aplicar tècniques de *raycasting* per a localitzar un objecte o un punt traçant una línia a partir d'una posició i una direcció, tècnica que s'usa normalment per a disparar o senyalar amb el punter. Heus aquí alguns exemples:
    - *getPosition("cub_1")* retorna la posició de l'objecte anomenat "cub_1" en forma de Vector.
    - *Force("cub_1", V(100,0,0))* aplica una força de 100 en la direcció de l'eix x.
    - *Stop("cub_1")* atura completament "cub_1", anul·lant la seva inèrcia.
    - *getNameByRay(V(0,0,0), V(1,0,0), 9999)* traça un raig de 9999 de distància partint de l'origen en la direcció de l'eix x que ens retornarà el nom del primer objecte amb qui impacti.
- **Funcions de càmera:** La càmera sol ser un dels objectes més mòbils d'un joc, per tant a més de conèixer o fixar la posició i orientació de la càmera, permet moltes de les transformacions més habituals que s'hi realitzen; moure-la respecte al món o a la seva pròpia orientació o també indicar directament a quina posició ha de mirar.
    - *setCameraPosition(V(0,0,0)* situa la càmera a l'origen
    - *CameraYaw(90)* rota la càmera 90º al voltant de l'eix z.
    - *CameraLookAt(V(0,10,0))* la càmera mirarà al punt (0,10,0), és a dir, amunt.

- **Propietats del món:** Tot i que menys tècniques, aquest grup inclou funcions importants que defineixen la gravetat o l'aspecte del terreny, que es pot definir mitjançant mapes d'alçada:
    - *Gravity(V(0,-9.8,0))* estableix una gravetat terrestre a l'escenari.
    - *Terrain("exemple", V(1500,50,1500), V(0,0,0))* situa el terreny de mida (1500,50,1500) a l'origen.
- **Altres:** També s'han afegit certes funcions que, tot i que no sempre afecten a la mecànica del joc, poden ser importants per a l'usuari. Aquestes són molt variades i van des de poder situar llums a l'escena fins a mostrar text per pantalla.
    - *Print("hola món")* mostra el text per pantalla
    - *Timer("timer1", 10, "disparar")* el programa farà una crida a la funció "disparar" (que haurem de crear en Lua) al cap de 10 segons.

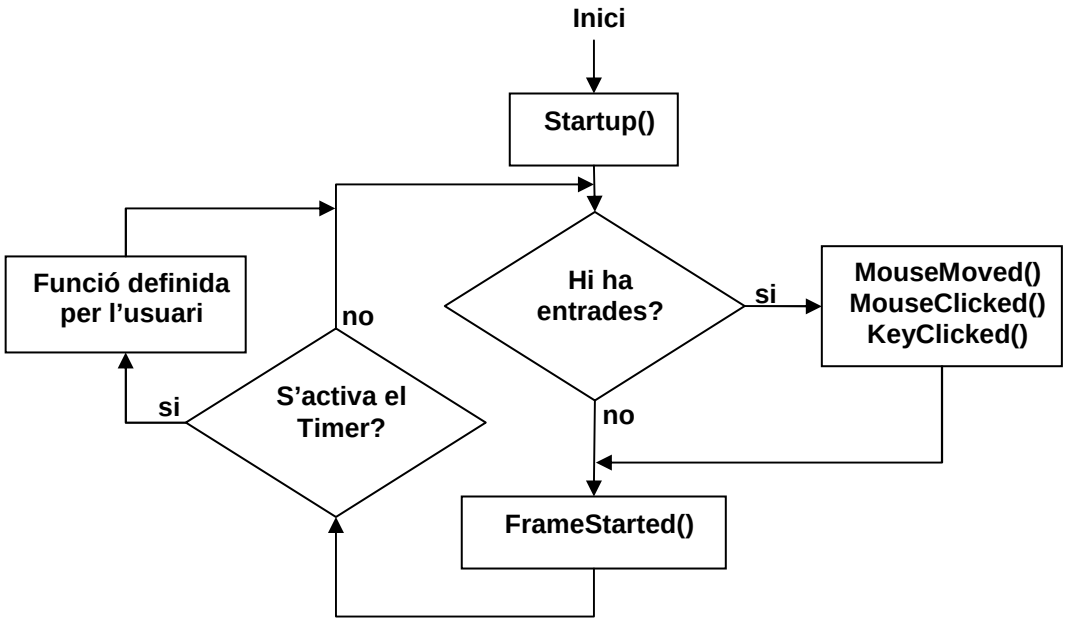


Figura 5. Diagrama de flux de les crides bàsiques que reb Lua

2.2.3 Funcionament intern

El codi que hem creat està format pels mòduls que es poden veure a la figura 6. El diagrama és una abstracció sobre l'esquema de classes UML complet de l'annex 2.

Engine és la classe base, un patró *singleton* on se succeeix el bucle principal i que implementa la unió entre OGRE i PhysX gestionada per NxOgre. En ella existeixen els elements principals del programa; *SceneManager* –l'escena gràfica que es dibuixa– i *Scene* –l'escena física que es calcula–. Aquests dos elements modelen el món i tot allò que hi és visible en forma d'arbre que agrupa i relaciona els elements. *Engine* és qui, en crear les dues escenes mencionades, fa la primera crida a Lua –la funció *startup()*– per a que el joc situï els elements inicials.

Frame és una classe heretada de OGRE que gestiona les entrades de teclat, ratolí i la majoria d'esdeveniments que es produeixen. Aquí és on tenen lloc les dues primeres crides cap a LUA exposades al punt 2.2.1, les interrupcions per part de l'usuari i les crides periòdiques després de cada cicle.

Els objectes que poden ser creats des de LUA estan descrits a la classe *Objects* i les funcions a la classe *Primitives*. Aquests són els únics punts des del qual rebrem les instruccions LUA, que accedeixen als membres *SceneManager* i *Scene* gràcies al *singleton Engine*.

Aquestes funcions de LUA es vinculen a *ScriptManager*, l'element que vincula la realitat interna i externa relacionant cada crida amb la seva respectiva funció. Aquesta classe és un excel·lent manual d'ús del programa, ja que referencia directament totes les possibilitats de cada funció i objecte.

Finalment, *Timer* i *Timermanager* són una integració dels temporitzadors a la part interna. Cada *Timer* conté la funció de Lua que haurà de ser cridada quan passi el temps establert. A part d'estalviar operacions que normalment es farien a l'exterior amb un comptador de cicles, serveix com a sistema d'alertes molt útil per a l'usuari, ja que pot indicar quina funció es cridarà en cada cas, ajudant així a organitzar el programa LUA.

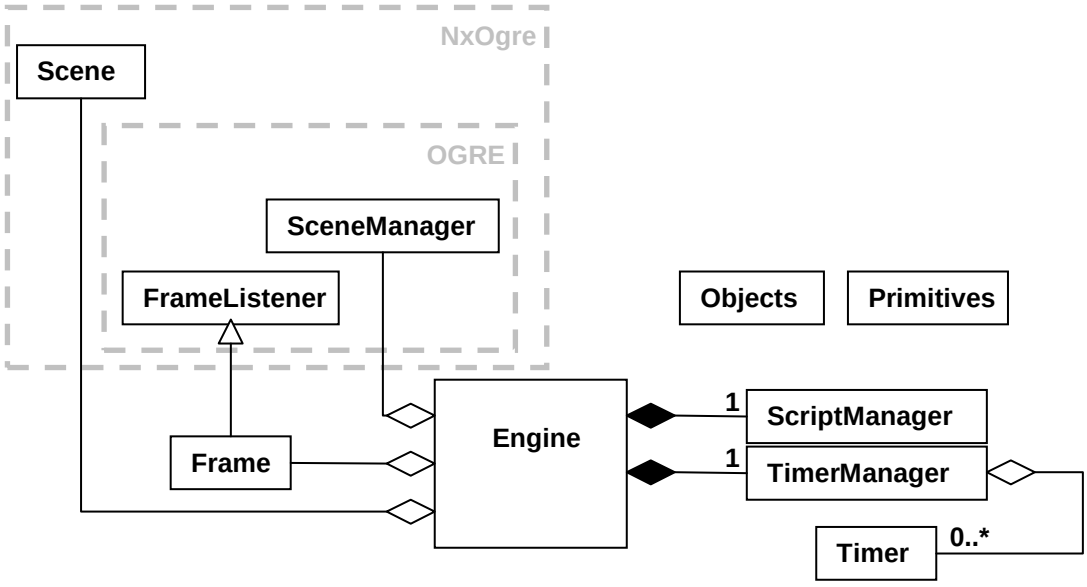


Figura 6. Diagrama UML simplificat del programa

3. Resultats

La plataforma de prototipatge construïda acompleix els objectius de rapidesa i flexibilitat tal i com mostrarem més endavant, en part gràcies a l'elecció dels seus components. OGRE permet configurar l'aspecte gràfic en gran mesura, Lua és un llenguatge sòlid amb totes les facilitats per a construir programes complexes i PhysX és un motor físic de primer nivell que funciona impecablement.

Un important complement per a aquest capítol de resultats és el contingut de l'annex 3. Allà hi ha el codi font de cada un dels codis generats. La lectura d'aquests és senzilla ja que no hi ha llargs algorismes ni funcions complicades. Un dels nostres objectius era la rapidesa en la creació i això és comprovable revisant el codi creat, que és un llistat d'instruccions amb poques sentències condicionals.

En aquest apartat de la memòria es farà evident el problema exposat al punt 1.1 de la introducció; comunicar interactivitat és difícil. El joc és una experiència i aquesta és complicada de transmetre a algú que no hi té accés. És per això que en aquest capítol ens centrarem a donar mètriques alternatives que permetin entendre bé el resultat. També les acompanyarem d'imatges i seqüències dels jocs que, tot i que difícilment transmeten el que volem, són un suport molt útil per a seguir els codis fonts de l'annex 3.

Figura 7. El joc de les bitlles

3.1 Joc de bitlles

El joc de les bitlles ha estat el banc de proves del sistema per la seva simplicitat. Aquest senzill joc permet disparar una pilota contra un conjunt d'objectes amb l'objectiu de tirar-ne el màxim possible. Com s'aprecia a la figura 7, la pilota és a la mà –quieta davant la càmera– mentre l'usuari apunta. Quan aquest prem la tecla espai la pilota surt disparada endavant.

La mecànica del joc està definida en menys de 90 línies de codi Lua. En el codi font que es pot veure a l'annex 3 d'aquesta memòria es pot apreciar la facilitat per a definir un món i una mecànica senzilla com aquesta. Mostrem a la figura 8 com es crea un món buit i s'omple amb uns quants objectes. Les quatre primeres línies especifiquen la força de la gravetat i creen un pla completament il·luminat, després es situa la càmera i finalment les últimes quatre línies creen sis bitlles i una pilota.

```
function startup()
    Gravity(V(0,-9.8,0))
    Floor()
    Pla(V(0,1,0), 500, 500):crear("")
    LightAmbient(Color(1,1,1))
    setCameraPosition(V(200,10,0))
    CameraYaw(90)
    Bitlla = Cub(V(2.5,7.5,2.5), 10)
    for i=1,6 do
        Bitlla:crear("Bitlla" .. i, PosicioInicial + Bitlles[i] * 5)
    end
    Esfera(1, 1000):crear("Bola", V(0,0,0))
end
```

Figura 8. Codi que genera un món similar al del joc de bitlles que hem creat

Per a aquest primer joc el centre de les proves va ser en tot moment la fiabilitat del motor físic. Un dels objectius més exigents del projecte requeria que aquest fos de primer nivell. El càlcul de les interaccions entre objectes és un aspecte determinant de la mecànica del joc resultant. Vàrem comparar els càlculs del motor físic amb el comportament habitual en el món real en xocs entre objectes. En tot moment el comportament és realista si bé la força de la gravetat només sembla correcte quan es té un punt de referència per a entendre l'escala de l'escena. En la imatge seqüència de la figura 9 es pot apreciar la caiguda d'una bitlla (els rectangles blancs) al ser impactada per una pilota (l'esfera blava) vista des d'un lateral.

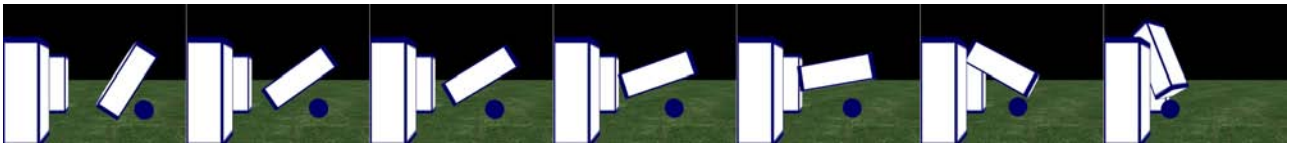


Figura 9. Seqüència que mostra com un objecte impacta en un altre

Aquesta primera creació també inclou una interfície gràfica comú a tots els prototips que es troba a la part dreta del panell verd que es pot veure a la figura 7. Allà es donen mesures del rendiment de l'aplicació i també la identificació numèrica de l'última tecla premuda. Aquest identificador numèric facilita al dissenyador la programació de la interfície del prototip.

En tractar-se aquest d'un joc orientat a ser un sistema de proves de les funcionalitats que s'anaven incorporant, no es va construir tenint en compte el temps d'elaboració. Es van efectuar proves per a comprovar el realisme de les reaccions físiques, que en tot moment tenen un comportament realista.

3.2 Conducció

El primer joc complet que vàrem crear va ser un de conducció. L'objectiu és donar una volta ràpida a un circuit. La restricció de tenir una càmera de bord que se situa en relació a un objecte –que a efectes d'aquest prototip serà una pilota– augmenta la complexitat dels càlculs algebraics realitzats, però el codi segueix essent molt curt, menys de 70 línies.

A la figura 10 es pot veure una instantània del moment de la sortida. L'ús que es dóna a la interfície gràfica serveix un doble propòsit, dóna informació rellevant al jugador però també és una eina per al dissenyador. El temps que ha transcorregut des de l'inici de la cursa o el rècord de velocitat són informacions per al jugador, que voldrà superar el seu rècord, les coordenades de posició de la pilota són, en canvi, una eina per al dissenyador.

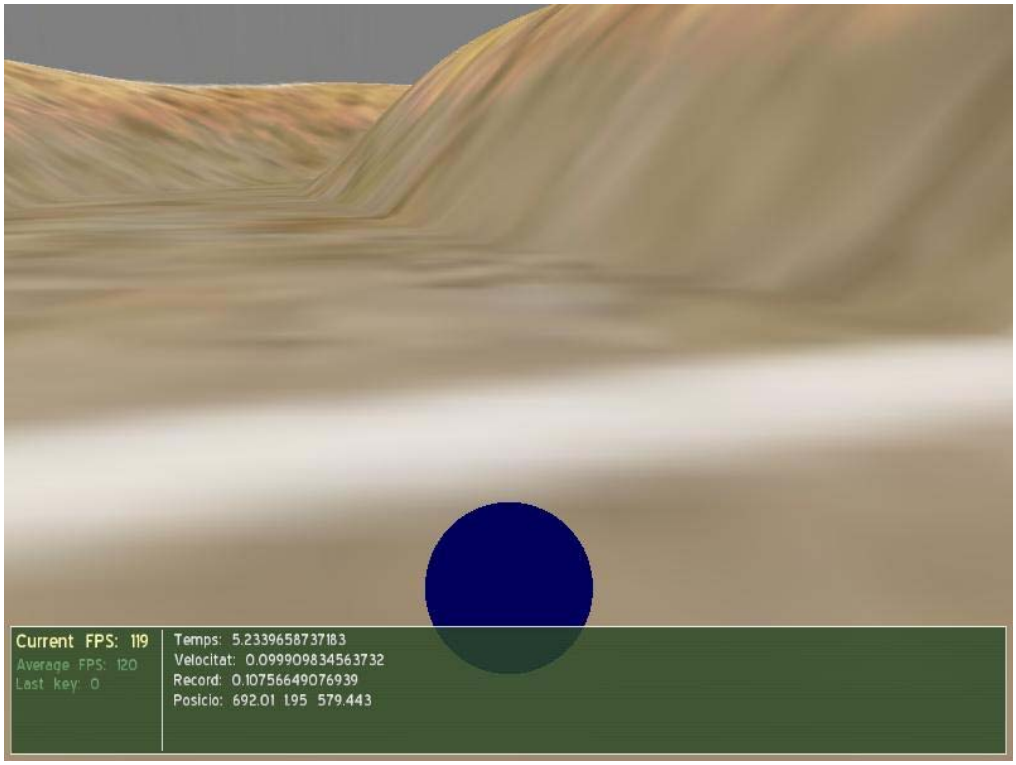


Figura 10. El joc de conducció

El codi d'aquest joc té com a element clau el càlcul de la direcció de la pilota com a conseqüència de les instruccions del jugador. Aquestes forces s'han d'aplicar en relació a la direcció de la pilota, per tant, busquem un vector perpendicular a la direcció del moviment mitjançant un producte vectorial. La variable *Right* de la figura 11 indica si estem girant cap a la dreta (prendria valor

1), a la esquerra (-1) o si no estem girant (0). Un cop trobada la força resultant s'aplica a la pilota en proporció al temps que ha passat des de l'últim càlcul (*elapsed*). Així assegurem que la incidència del jugador és relativa al temps i no depèn de la potència de l'ordinador, que en cas contrari, al calcular més cicles per segon, aplicaria més cops aquesta força.

```
ResultingForce = V(0,0,0)
if Brake == -1 and Magnitude(BallVelocity) > 5 then ResultingForce = BallDirection * Brake end
ResultingForce = ResultingForce + BallDirection * Accel
ResultingForce = ResultingForce + Cross(BallDirection, V(0,Right,0))
Force("Bola",ResultingForce * elapsed * BallForce)
```

Figura 11. Extracte del codi font Lua del joc de conducció

Per a aquest joc també hem incorporat una nova tecnologia: el mapa d'alçades. El mapa d'alçades ens ha permès crear el circuit del joc en pocs minuts. La senzillesa del mètode implementat el fa ideal per a tots els prototips d'ara endavant. Com s'aprecia a la figura 12, la imatge en blanc i negre –que representa les alçades– pot servir com a base per a després afegir-hi color –en aquest cas, lluminositat– a una textura qualsevol.

Els beneficis d'aquesta tecnologia estan completament alineats amb les necessitats del projecte i per tant constitueixen un dels seus elements més potents. Revisant les figures 10 i 12 i prenent la línia blanca de meta com a referència, es pot apreciar la correspondència entre els colors del mapa d'alçades i la textura en la presentació real del joc.

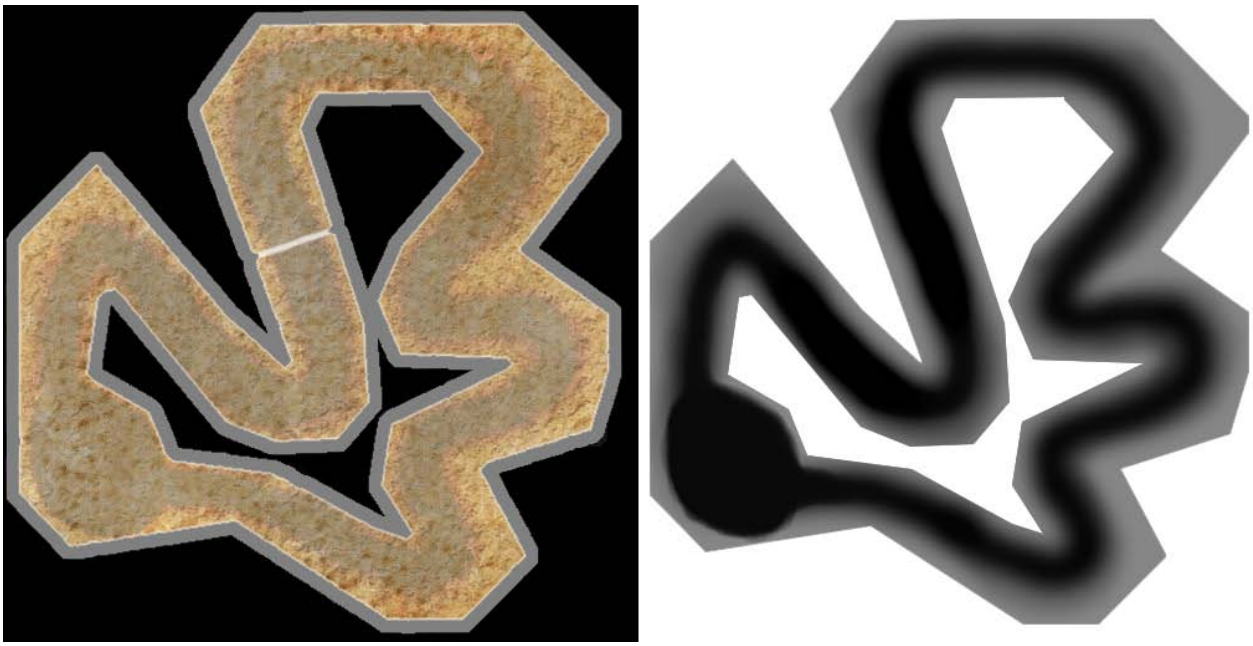


Figura 12. Mapa d'alçades (dreta) i la textura (esquerra) que conformen el circuit de carreres.

Aquest joc és en el que més clarament es pot observar l'eficiència temporal d'aquesta eina. El joc de l'exemple es va poder crear en menys de vint minuts, la majoria dels quals van estar dedicats a resoldre el seguiment de l'objecte per part de la càmera. Per a crear un joc similar partint de les mateixes eines

(OGRE i PhysX) escrit totalment en C++ es trigaria varies hores. D'haver-ho fet així, a part d'un gran increment en el rendiment de l'ordinador, haguèssim aconseguit un joc embrió que podríem seguir desenvolupant. Usant la nostra eina, en canvi, sacrifiquem aquests dos punts que hem considerat innecessaris en les assumpcions inicials per a tenir els resultats molt abans. Dit d'una altra manera, podríem fer moltes més proves en un mateix interval de temps.

3.3 Estratègia

Un joc d'estratègia és un repte per el coneixement del món que s'ha de gestionar. Lluny dels càlculs vectorials, es tracta de poder gestionar els diferents objectes per assignar-los propietats. Un punt clau del comportament d'aquests jocs és poder assenyalar un objecte al qual voldrem donar-li instruccions, per això va fer falta implementar una funció de *raycasting* mitjançant la qual es pot recuperar el nom del primer objecte en el qual impacti una línia recta traçada des d'un punt qualsevol.

Figura 13. Joc d'estratègia.

L'ús més comú en aquest gènere sol ser clicar amb el ratolí sobre un objecte per a seleccionar-lo, traçant una recta des de la càmera fins al punt que assenyala el cursor. La nova funció creada ens permet fàcilment rebre un nom traçant una línia que surt de la càmera en la direcció en la que està mirant. La figura 13 és una captura de pantalla que, si bé és poc llegible, mostra com el jugador pot seleccionar objectes, en aquest cas l'objecte anomenat *Personatge3*, i amb l'altre botó del ratolí indicar fins a quina posició vol que arribi.

La figura 14 mostra una crida a la funcionalitat de *raycasting* que, sense gairebé modificacions, també serà crucial per a l'últim gènere que hem provat, l'acció en primera persona.

```
Cosa = getNameByRay(getCameraPosition(), getCameraDirection(), 9999)
```

Figura 14. Crida a una funció de raycasting

Per tal d'implementar un joc d'estratègia és vital que cada objecte tingui un conjunt de propietats, és per això que és possible accedir a ells a través d'un nom. Cada objecte té un nom únic, per tant aquesta és la clau primària mitjançant la qual es podrà usar per a crear una base de dades amb les propietats de cada objecte.

Aquesta base de coneixement s'ha modelat amb un sistema d'etiquetes per el qual tots els objectes del món tenen un nom únic. Les funcions sobre objectes especifiquen el seu objectiu amb el seu nom. Aquesta és la base sobre la qual es pot construir un joc d'estratègia. La tecnologia implementada funciona però, en aquest cas, es queda curta. Les estructures d'informació necessàries per a un joc d'aquestes característiques haurien d'estar situades totalment en l'entorn Lua. La generació i control d'estructures de dades no és un punt fort d'aquest llenguatge. Per a crear un prototip mínimament complex caldria crear noves funcions en la part interior del programa que donessin a Lua les estructures i els mètodes per a que l'usuari tant sols hagués d'escriure o llegir-ne les dades, però no crear-ne el model.

3.4 Acció en primera persona

L'acció en primera persona és el gènere al qual pertanyien els primers jocs en tres dimensions que van ser creats. Aquest gènere simula la nostra immersió en el joc, usant el ratolí per a mirar cap als voltants. Gairebé l'única forma d'interacció amb l'entorn és l'arma que empunyem i aquesta dispara partint de la càmera, de manera que se sol assenyalar el centre de la pantalla amb una creueta per a indicar cap a on aniran els trets.

Per al nostre projecte això implica dues conseqüències. Com que la càmera és el personatge, cal que es comporti com una part del joc i no com un simple observador, és a dir, no pot flotar en l'espai com fèiem fins ara. En segon lloc, cal poder localitzar els impactes dels dispars amb precisió i aplicar les forces conseqüentment.

Portar a terme aquest joc demana la implementació d'un sistema de *raycasting* precís per a determinar punts d'impacte de raigs tal i com fèiem en els joc d'estratègia. Aquest cop, però, volem saber-ne les coordenades. Això permetrà situar els dispars del jugador, però també crear la il·lusió de que la càmera es mou com una persona, traçant un raig vertical que impacti al terra i ens permeti situar-la en tot moment enganxada al terra i no flotant. De la mateixa manera podrem impedir que aquesta travessi parets o simular la gravetat en cas de que el jugador efectuï un salt.

En el nostre joc, visible a la figura 15, els dispars del jugador es converteixen en impactes amb posició i direcció concretes que es poden transformar en conseqüències lògiques o físiques. En el

primer cas, igual que amb el joc d'estratègia, podem accedir al nom de l'objecte impactat: en el segon, hem implementat la possibilitat d'aplicar forces locals, de manera que aquest resulti impactat en un punt concret i reaccioni de manera adequada a la força aplicada.

Figura 15. Joc d'acció en primera persona

Aquest és el joc més elaborat del projecte, per ser l'últim i per tractar-se també del gènere més exigent dels tractats fins ara. El jugador ha de disparar repetides vegades a un objecte que apareix repetidament, talment com en una competició de tir al plat. L'objectiu és el d'impulsar aquest objecte el més lluny possible en un temps determinat.

El codi d'aquest joc és curt i planer. Està format tant sols per 80 línies de codi i va ser creat en una hora. La figura 16 mostra com es regula el comportament de la càmera. A cada cicle es traça un raig vertical que consulta a quina distància es troba el terra per sota nostre i col·loca la càmera 3 unitats per damunt.

```
Terra = getPosByRay(getCameraPosition() + V(0,10,0), V(0,-1,0), 9999)
setCameraPosition(Terra + V(0,3,0))
```

Figura 16. Detall del sistema de localització de la càmera en el joc d'acció

3.5 Perspectives de continuïtat

El resultat d'aquest projecte és una bona eina per a crear prototips de curt i mig abast. Per a dissenyar un joc comercial de primer nivell és possible necessitar provar-ne també la tecnologia, no només la mecànica com hem fet fins ara. Les eines creades en aquest projecte, per tant, no serien suficients. Per a les primeres proves de disseny de projectes menys ambiciosos, en canvi, aquest programa respon adequadament a les necessitats del dissenyador.

A part de poder simular un joc i ser usat com a forma de prototipatge, la rapidesa i facilitat de crear jocs permet utilitzar aquesta eina des d'una altra vessant. El nostre programa és un banc de proves i es podria utilitzar com a tal, no sols per a dissenyar un joc específic, sinó també com a escenari per a endinsar-se en la recerca de nous jocs. En certa manera, seria l'eina que articularia una sessió de brainstorming. Podent realitzar una simulació senzilla del joc en pocs minuts superarem la difícil barrera de comunicar les nostres idees abstractes a una altra persona. Així acabem amb la dificultat de representar interactivitat mitjançant paraules o dibuixos, usant el prototipatge com un mitjà de comunicació d'idees.

En el futur més immediat treballarem per a convertir l'eina en un bon *portfoli* de cara a millorar el nostre *currículum vitae*. El programa no tant sols és un punt més d'aquest, sinó una plataforma per a crear jocs a partir d'ara. En ella s'hi poden reflectir bones idees que un programador individual difícilment pot convertir en un bon joc, però que gràcies a aquest prototip es poden jugar. Poder-les jugar és el factor determinant per a poder predir que una mecànica generarà un bon joc, sense haver de fer una assumpció al respecte.

Concretant aquestes intencions, en primer lloc caldrà implementar un sistema de dades que permeti dotar als objectes d'un conjunt de valors. Això permetrà gestionar l'estat del joc fàcilment, de manera que les dades que defineixen un personatge siguin una característica de l'objecte que el representa consultable amb una senzilla funció. Aquesta és una de les millores proposades al punt 4 de la memòria, que s'ha fet evident en crear el joc d'estratègia i que creiem que és la mancança més gran d'aquest projecte.

Un cop millorat en aquest sentit, es podrà treballar en nous jocs. Tot i que començarem elaborant una recerca per a la que aquests prototips ràpids seran molt útils, el treball realitzat fins ara ja assenyala cap a una variant del joc d'acció creat en aquest projecte final de carrera.

4. Conclusions

- S'ha implementat un sistema de prototipatge que separa correctament els factors comuns als videojocs dels que en defineixen la mecànica.
- La mecànica del joc es programa en llenguatge de script dedicat en tots els casos a especificar la resposta generada als inputs de l'usuari i no a gestionar com ho farà.
- Els elements comuns d'entrada, càlculs físics i sortida per pantalla funcionen correctament amb un rendiment acceptable per a qualsevol ordinador d'avui en dia.
- S'han creat varis jocs de gèneres diferents que mostren la versatilitat del projecte per a satisfer les necessitats de la gran majoria de prototips. Per bé que és difícil assegurar que qualsevol joc pot ser creat eficientment a partir d'aquest conjunt d'instruccions, hem demostrat que el sistema és ràpid per a la gran majoria.
- S'han assolit els objectius de rapidesa necessaris per a conservar les necessitats del prototipatge. Hem mesurat el temps de creació de cada joc observant així la rapidesa d'implementar les mecàniques bàsiques dels principals gèneres del videojoc.
- Una de les fases més llargues de la creació d'aquesta plataforma ha estat una iteració contínua sobre els sistemes bàsics dels videojocs per analitzar-ne els elements que en volíem abstreure. Aquest aprofundiment ha ajudat a emfatitzar un dels pilars del projecte; certs elements són tant comuns que sovint esdevenen redundants. El sistema de càlculs gràfics i físics utilitzats aquí és vàlid per a gairebé qualsevol joc. Aquesta revisió del procés de creació d'un videojoc és molt útil per a algú que pretén desenvolupar videojocs com és el meu cas.
- L'elecció de OGRE i PhysX com a components ha estat determinant per la qualitat del producte final. La solidesa del primer i la seva excel·lent documentació on-line el fan una molt bona elecció per a tot tipus de projectes gràfics, no només jocs. PhysX és un software professional i propietari, per tant gaudeix d'un suport immillorable per al desenvolupador. Tot i que la impossibilitat d'accedir al codi font pot ser un inconvenient en certs casos, no ho ha estat per a aquest projecte.
- Una decisió posterior, unir aquests dos elements mitjançant l'adaptador NxOgre ha resultat menys encertada. Per bé que NxOgre facilita la unió d'aquests dos elements i permet dedicar menys temps als aspectes tecnològics –qüestió important per al nostre projecte– passat el temps aquest esdevé un llast per a l'aplicació. NxOgre és un adaptador encara en desenvolupament, l'equip del qual no garanteix la regularitat i el manteniment necessaris a llarg termini. Al tractar-se d'un projecte de codi lliure no ens podrem trobar amb que aquest component quedi obsolet i immodificable, però si que ens fa replantejar la utilitat d'usar-lo si de bon principi contemplem reescriure'n el codi com una possibilitat.
- La meva recomanació seria emprar el temps necessari per a ajuntar OGRE i PhysX amb els mitjans propis i prescindir d'un adaptador. Per bé que allarga la fase inicial d'un projecte i

endarrereix l'aparició de resultats, serà una decisió correcta a llarg termini quan es vulgui validar la solidesa del programa.

- Lua, d'altra banda, està esdevenint un dels llenguatges de script més usats en la indústria. Com a jugador he pogut constatar que la presència d'aquest s'estén i que aviat es podrà considerar un estàndard *de facto* tant dins del món del videojoc com fora. La qualitat d'aquest component, en canvi, la poso més en dubte que no pas la dels dos anteriors. Lua no és un producte comercial i tot i que té un bon manual per a l'usuari, manca d'una documentació extensa per l'implementador que el vol usar en les seves aplicacions. Algunes de les seves mancances, que hem solucionat usant Luabind, han reduït encara més la documentació disponible. En general Lua és poc transparent i genera blocs de codi poc modularitzables dins del programa i que poc tenen a veure amb sintaxi o organització amb el llenguatge C.
- Malgrat això i contràriament al cas de NxOgre, creiem que Lua és la millor alternativa per a llenguatge de script. La seva popularitat en garanteix una millora gradual en la seva documentació i també una millora en la usabilitat del nostre programa en tractar-se d'un llenguatge conegut per molta gent. La implementació d'un llenguatge de script diferent probablement resultaria en minses millores al preu d'allunyar molts possibles usuaris del producte.
- En segon terme, he pogut acostar-me al món del disseny dels videojocs, que és la meva passió. La valoració dels reptes que proposa un joc aporta un coneixement sobre el funcionament dels mecanismes que duen als objectius finals de generar diversió i interès en l'usuari. El prototipatge és la millor manera per a accedir a aquests coneixements a través de la pràctica.
- Creant els jocs he pogut adonar-me de la dificultat de posar davant del jugador un repte suficient per a que pugui ser jugat una bona estona i que alhora generi l'interès per a dedicar aquella estona a resoldre'l. Dit d'altra manera, el joc ha de divertir des del principi però també ha de mantenir l'interès del jugador viu per a que vulgui vèncer el repte i accedir-ne a més. Aquesta tasca es demostra difícil en la pràctica, cosa que posa de relleu la utilitat de poder dissenyar els jocs sobre prototips i no sobre la imaginació d'un creador.

El disseny del projecte contempla la possibilitat d'expandir el set d'instruccions per acomodar la tasca per la que s'usi aquesta eina. Les millores aquí contemplades tenen en compte les diverses necessitats dels possibles usuaris:

- **Jocs per a consoles portables:** Un dels mercats més actius i on és més fàcil d'entrar sense grans pressupostos és el dels jocs per a dispositius portables. Des del punt de vista del disseny, tenen dues grans limitacions: una interfície de pocs botons i una pantalla petita. Una empresa amb aquesta dedicació estalviaria temps creant llibreries Lua que adaptessin el nostre sistema d'entrades a un de més senzill on es reduïssin les crides als diferents elements de la interfície.
- **Homogeneïtzar les interfícies:** La majoria dels jocs pertanyen fortament a un gènere i es pot dir que tots ells comparteixen una interfície bàsica. A vegades el ratolí mou la càmera i a

vegades les fletxes del teclat s'usen per a moure el personatge. Aquests són elements sovint comuns que no hem volgut incorporar al sistema base per a acomplir amb els objectius de flexibilitat, ja que calia poder prototipar també jocs que no s'emmarquessin en els gèneres clàssics. En fer-se una feina de creació sobre un gènere en concret, aquestes accions es podrien incorporar per a estalviar redundàncies.

- **Grans projectes:** Les decisions de disseny preses després d'una primera fase de prototipatge podrien ser incorporades en una llibreria Lua. A partir d'aquí es podria començar una fase de prototipatge que construís a partir del punt on es trobi el desenvolupament, així s'allargaria la vida dels prototips i es podrien portar més lluny les proves amb prototips.
- **Introducció de so:** Els videojocs rítmics i musicals són un gènere creixent en els últims anys. Aquest projecte els ha deixat de banda per no tractar-se d'un dels gèneres clàssics del videojoc, però no queda cap dubte que és un gènere molt popular i en alça. Introduir sons i músiques per a incorporar jugabilitat a partir, no només del que es veu a la pantalla sinó també el que se sent, és una de les continuacions més interessants per a aquest projecte.

Referències bibliogràfiques

[Ban08] Eric Bangeman, "Growth of gaming in 2007 far outpaces movies, music"
Ars technica, 24 de gener de 2008 (http://www.arstechnica.com)

[Unseen64] Unseen64, "Beta, unreleased & unseen videogames!"
(http://www.unseen64.net)

[Wau06] Eric-Jon Waugh, "GDC: Spore: Pre-Production Through Prototyping"
Gamasutra, 29 de març de 2006 (http://www.gamasutra.com)

[Mar08] Rich Marmura, "Paper Prototyping: 5 Facts for Designing in Low-Tech"
Game career guide, 10 de juliol de 2008 (http://www.gamecareerguide.com)

[Kum07] Mathew Kumar, "Future Play 2007: Indie Dev Frozen North On Pushing The Prototype"
Gamasutra, 16 de novembre de 2007

[Ogre3D] Motor 3D flexible i orientat a escenes escrit en C++
(http://www.ogre3d.org)

[PhysX] Motor físic en temps real propietari de Nvidia i desenvolupat originalment per Ageia
(http://developer.nvidia.com/object/physx.html)

[Lua] Llenguathe de script lleuger i incrustable
(http://www.lua.org)

[Wol01] Mark J. P. Wolf, "The Medium of the Video Game"
Capítol 6, 2001

ANNEXOS

Annex 1: Manual de l'usuari

Per a descriure la mecànica d'un joc utilitzant el nostre programa es parteix d'un arxiu anomenat *joc.lua* situat a la mateixa carpeta que l'executable. L'aspecte en blanc d'aquest arxiu ha de ser el següent per a satisfer les crides bàsiques del sistema sense respondre-les.

```
function startup()
end
function MouseClicked(id, released)
end
function KeyClicked(id, released)
end
function MouseMoved(x, y)
end
function FrameStarted(elapsed)
end
```

1.1 Crides principals

Les següents funcions són crides que el sistema farà en certs moments i per tant han d'existir necessariament. Cap d'elles té valor de retorn, per tant per prescindir d'una n'hi ha prou en deixar-les en blanc.

Startup()

Aquesta funció es crida un únic cop al començar el joc.

MouseClicked(id, released)

Cada cop que l'usuari premi un botó del ratolí es cridarà aquesta funció. L'argument *id* indica quin botó ha estat activat i *released* si ha estat premut o deixat anar.

Id [number]: 0 botó esquerre
1 botó dret

Released [number]: 0 premut
1 deixat anar

KeyClicked(id, released)

Cada cop que l'usuari premi una tecla del teclat es cridarà aquesta funció. L'argument *id* indica quina tecla ha estat activada i *released* si ha estat premuda o deixada anar.

Id [number]: Per a permetre la identificació fàcil del número de les tecles aquest es mostra en la interfície per defecte tal i com està descrit al punt 3 d'aquest annex.

Released [number]: 0 premut
1 deixat anar

MouseMoved(x, y)

Cada cop que l'usuari mogui el ratolí es cridarà aquesta funció amb el moviment en l'eix horitzontal i vertical indicat als arguments *x* i *y* respectivament.

X [number]
Y [number]

FrameStarted(elapsed)

Aquesta funció es crida al començar cada cicle del bucle del joc, l'argument *elapsed* indica la quantitat de temps en mil·lisegons transcorregut des de l'últim cop que s'ha cridat aquesta mateixa funció.

Elapsed [number]

1.2 Tipus

Hem creat dos tipus bàsics d'argupació de dades anomenats V*ector* i el *Color* en honor a la seva funció habitual, si bé el primer és una agrupació de tres enters i el segon de quatre *numbers*.

Vector

Constructor

- V(x, y, z)

X [number]: Accessible amb el membre x (ex. V(1, 2, 3).x serà 1)
Y [number]: Accessible amb el membre y
Z [number]: Accessible amb el membre z

Admet les operacions de suma (+), resta (-), multiplicació (*) i divisió (/) amb altres *Vector* i multiplicació i divisió amb escalars.

Color

Constructor

- Color(red, green, blue)
- Color(red, green, blue, alpha)

Admet les operacions de suma (+), resta (-) amb altres *Color* i multiplicació (*) i divisió (/) amb escalars. El valor per defecte d'*alpha* és 1.

1.3 Objectes

Els objectes bàsics que representaran els elements del joc són els següents:

Cub

L'objecte tindrà el comportament d'un cub per al motor físic si bé es podrà usar una malla qualsevol per a representar-lo gràficament

Constructor

- Cub(escala, massa)
  - Escala [Vector]: L'escalat en les tres coordenades del vector.
  - Massa [number]: La massa de l'objecte

Mètodes

- material(textura) Especifica quina textura aplicar.
  - Textura [string]: Textura a aplicar a l'objecte segons les convencions d'OGRE
- mesh(malla) Especifica quina malla dibuixar. Per defecte, un cub.
  - Malla [string]: Malla que es dibuixarà
- crear(nom, posició) Crea l'objecte en qüestió
  - Nom [string]: Dóna nom a l'objecte. En blanc se'n generarà un a l'atzar.
  - Posició [Vector] Posició on apareixerà l'objecte

Esfera

L'objecte tindrà el comportament d'una esfera per al motor físic, si bé es podrà usar una malla qualsevol per a representar-lo gràficament

Constructor

- Esfera(escala, massa)
  - Escala [number]: Escalat de l'esfera.
  - Massa [number]: Massa de l'objecte

Mètodes

- material(textura) Especifica quina textura aplicar.
  - Textura [string]: Textura a aplicar a l'objecte.

- mesh(malla) Especifica quina malla dibuixar. Per defecte, una esfera.
  Malla [string]: Malla que es dibuixarà
- crear(nom, posició) Crea l'objecte en qüestió
  Nom [string]: Dóna nom a l'objecte. En blanc se'n generarà un a l'atzar.
  Posició [Vector] Posició on apareixerà l'objecte

Pla

El pla existeix tant sols com una representació gràfica, no física, amb el centre a l'origen. Per al seu ús com a terra es recomana usar-lo conjuntament amb la funció Floor().

Constructor

- Pla(normal, amplada, alçada)
  Normal [Vector]: La normal del pla n'especifica la orientació.
  Amplada [number]: Mida del pla.
  Alçada [number]: Mida del pla.

Mètodes

- material(textura) Especifica quina textura aplicar.
  Textura [string]: Textura a aplicar a l'objecte.
- crear(nom) Crea l'objecte en qüestió
  Nom [string]: Dóna nom a l'objecte. En blanc se'n generarà un a l'atzar.

1.5 Funcions primitives

Físiques: Representació del món

- [Vector] getPosition([string] nom)
  Retorna la posició de l'objecte amb l'etiqueta *nom*.
- setPosition([string] nom, [Vector] posició)
  Col·loca l'objecte *nom* a la posició indicada.
- [Vector] getVelocity([string] nom)
  Retorna la velocitat de l'objecte amb l'etiqueta *nom*.
- setVelocity([string] nom, [Vector] velocitat)
  Dóna a l'objecte *nom* la velocitat i direcció indicades segons direcció i magnitud del Vector *velocitat*.
- Stop([string] nom)
  Atura completament l'objecte *nom*.
- Force([string] nom, [Vector] força)
  Aplica la força sobre l'objecte nom segons direcció i magnitud de *força* en el seu centre de massa.
- LocalForce([string] nom, [Vector] posició, [vector] força)
  Aplica la força local indicada per *força* a l'objecte *nom* en el punt *posició*.
- MoveTo([string] nom, [Vector] posició, [number] força)
  Aplica la força indicada a l'objecte *nom* per a moure'l cap a *posició*.
- Spin([string] nom, [Vector] força)
  Aplica força rotacional a l'objecte *nom* en torn al seu centre de massa amb direcció i magnitud indicat per *força*.
- [string] getNameByRay([Vector] posició, [Vector] raig, [number] distància)
  Retorna el nom del primer objecte impactat d'un raig des de *posició* en la direcció que indica *raig* amb un abast màxim de *distància*.
- [Vector] getPosByRay([Vector] posició, [Vector] raig, [number] distància)
  Retorna la posició d'impacte amb el primer objecte que trobi un raig des de *posició* en la direcció que indica *raig* amb un abast màxim de *distància*.

Càmera: Punt de vista

- [Vector] getCameraDirection()
  Retorna la direcció en la que apunta la càmera.
- [Vector] getCameraPosition()
  Retorna la posició que ocupa la càmera.
- setCameraPosition([Vector] posició)
  Situa la càmera en la posició indicada per *posció*.
- CameraTranslate([Vector] direcció, [number] coordenades)
  Mou la càmera en la direcció indicada per *direcció*. El número *coordenades* especifica el marc de referència.
  Coordenades [number]: 0 relatiu a la posició de la càmera
  1 relatiu al món
- CameraYaw([number] moviment)
  Rota la càmera al voltant de l'eix y la quantitat indicada per *moviment*, en graus.
- CameraPitch([number] moviment)
  Rota la càmera al voltant de l'eix x la quantitat indicada per *moviment*, en graus.
- CameraRoll([number] moviment)
  Rota la càmera al voltant de l'eix z la quantitat indicada per *moviment*, en graus.
- CameraLookAt([Vector] posició)
  Rota la càmera per a que miri a la *posició* indicada.

Aspecte gràfic

- LightAmbient([Color] color)
  Fixa una llum constant per a totes les superfícies independentment de la seva orientació.
- LightAdd([Vector] posició, [Color] color, [string] nom)
  Crea una llum puntual que il·lumina en totes direccions
- SkyBox([string] cel)
  Crea un embolcall amb l'aparença de cel segons el format d'OGRE.

Mostrar text per pantalla

- Print([string] text)
  Escriu el text indicat al panell verd de la interfície gràfica.
- [string] PrintVector([Vector] vector)
  Retorna els valors del vector en format string.

Matemàtiques

- [Vector] Cross([Vector] a, [Vector] b)
  Retorna el producte vectorial dels vectors *a* i *b*.
- [Vector] Normalize([Vector] a)
  Retorna el vector *a* normalitzat.
- [number] Magnitude([Vector] a)
  Retorna el mòdul del vector *a*.

Propietats del món

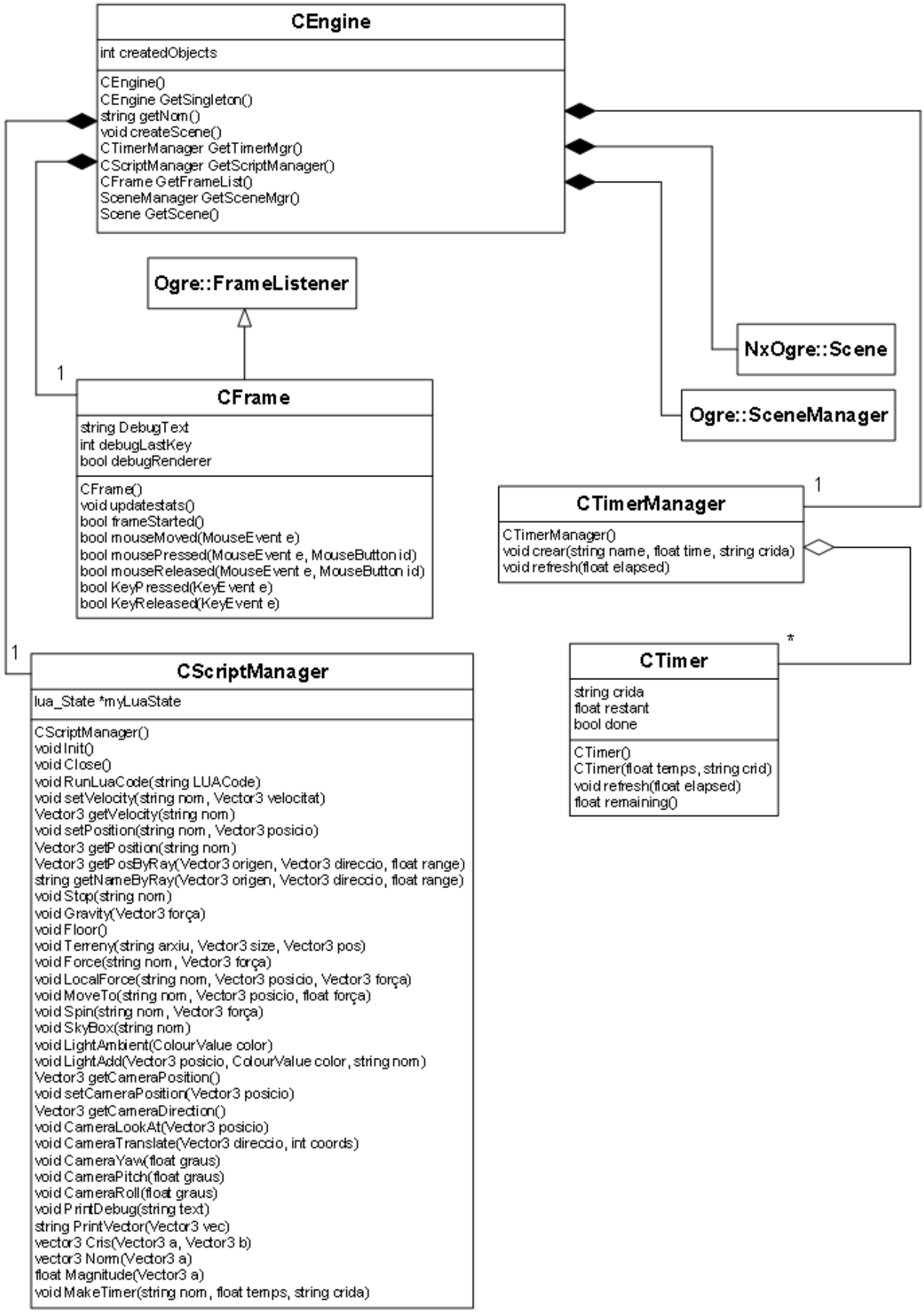
- Gravity([Vector] gravetat)
  Aplica una força constant a tots els objectes amb força i direcció indicades per el Vector *gravetat*.

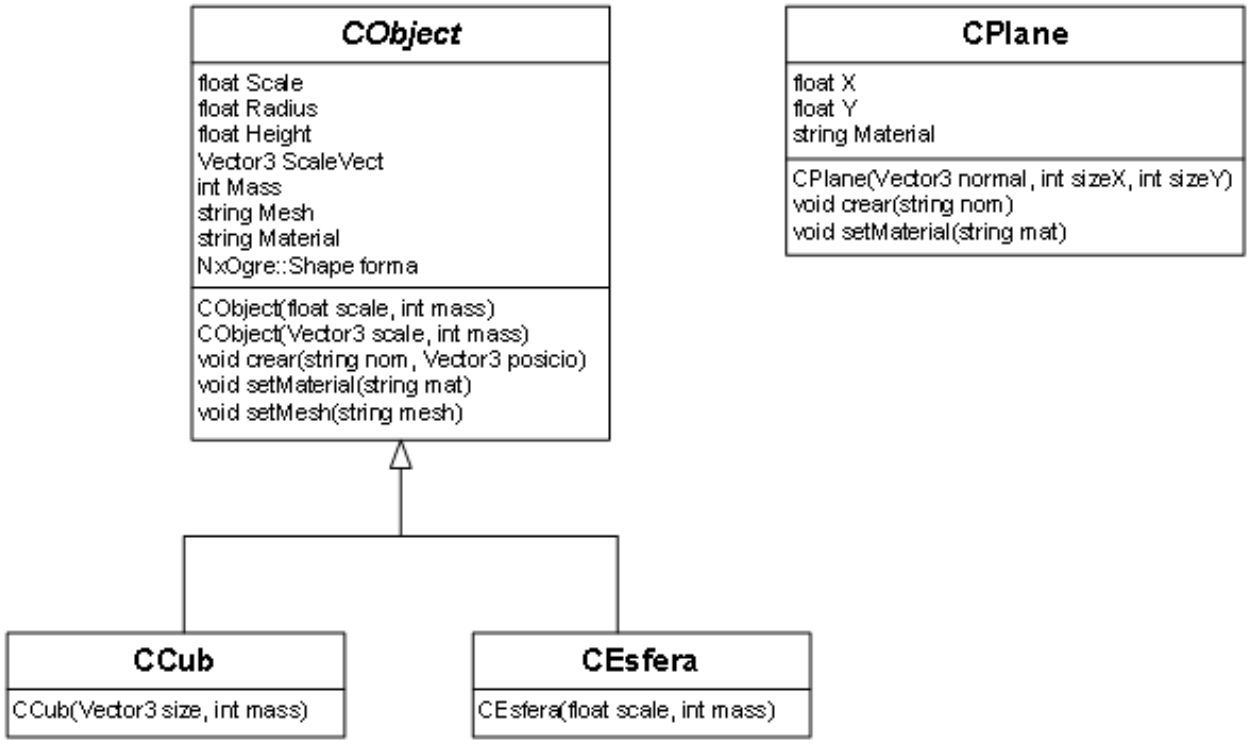
- Floor()
  Crea una superfície invisible i infinita en el pla y = 0.
- Terreny([string] arxiu, [Vector] mida, [Vector] posició)
  Crea el mapa d'alçades gràfic indicat a l'*arxiu* (.cfg) segons el format d'OGRE i el mapa d'alçades físic (.xhf) per a NxOgre.

Temporitzadors

- MakeTimer([string] nom, [number] temps, [string] crida)
  Indica al programa que cridi a la funció *crida* d'aqui a *temps* segons.

Annex 2: Diagrama de classes UML




CObject
float Scale
float Radius
float Height
Vector3 ScaleVect
int Mass
string Mesh
string Material
NxOgre::Shape forma
CObject(float scale, int mass)
CObject(Vector3 scale, int mass)
void crear(string nom, Vector3 posicio)
void setMaterial(string mat)
void setMesh(string mesh)
CPlane
float X
float Y
string Material
CPlane(Vector3 normal, int sizeX, int sizeY)
void crear(string nom)
void setMaterial(string mat)
CCub
CCub(Vector3 size, int mass)
CEsfera
CEsfera(float scale, int mass)

Annex 3: Codi font dels jocs creats

3.1 Joc de bitlles

```
CameraDirection = V(0,0,0)
CameraRotationX = 0
CameraRotationY = 0
CameraMovSpeed = 50
CameraRotSpeed = -20
PosicioInicial = V(-50,10,0)
Bitlles = { V(0,0,0), V(-4,0,-2), V(-4,0,2), V(-8,0,-4), V(-8,0,0), V(-8,0,4) }
disparat = 0

function startup()
	Gravity(V(0,-9.8,0))
	SkyBox("Core/Evening")
	Floor()
	Plano = Pla(V(0,1,0), 500, 500)
	Plano:material("Examples/GrassFloor")
	Plano:crear("")
	LightAmbient(Color(1,1,1))
	setCameraPosition(V(200,10,0))
	CameraYaw(90)
	Bitlla = Cub(V(2.5,7.5,2.5), 10)
	Bitlla:material("Examples/Bitlla")
	for i=1,6 do
		Bitlla:crear("Bitlla" .. i, PosicioInicial + Bitlles[i] * 5)
	end
	E = Esfera(0.8, 1000)
	E:material("Examples/Bolo")
	E:crear("Bola", V(0,0,0))
end

function MouseClicked(id, released)
	if id == 0 then
		if released == 0 then
			RotaDirection = Cross(getCameraDirection(), V(0,-1,0))
		else
			RotaDirection = V(0,0,0)
		end
	elseif id == 1 then
		if released == 0 then
			RotaDirection = Cross(getCameraDirection(), V(0,1,0))
		else
			RotaDirection = V(0,0,0)
		end
	end
end

function KeyClicked(id, released)
	if id == 203 then
		if released == 0 then CameraDirection.z = CameraDirection.z + CameraMovSpeed
		else CameraDirection.z = CameraDirection.z - CameraMovSpeed
		end
	elseif id == 205 then
		if released == 0 then CameraDirection.z = CameraDirection.z - CameraMovSpeed
		else CameraDirection.z = CameraDirection.z + CameraMovSpeed
		end
	elseif id == 57 then
	if released == 0 then
			if disparat == 0 then
				Stop("Bola")
				Force("Bola", getCameraDirection() * 80000)
				disparat = 1
				Timer("", 7, "resetDispar");
			end
		end
	end
end

function resetDispar()
	disparat = 0
	Stop("Bola")
end
```

```
function MouseMoved(x, y)
	CameraRotationX = CameraRotationX + x
	CameraRotationY = CameraRotationY + y
end

function FrameStarted(elapsed)
	if disparat == 0 then
		setPosition("Bola", getCameraPosition() + getCameraDirection() * 15)
		Print("Apunt per a disparar.")
	else
		Force("Bola",RotaDirection * elapsed * 6000)
		Print("Pilota en joc.")
	end
	CameraTranslate(CameraDirection * elapsed, 1)
	CameraYaw(CameraRotationX * CameraRotSpeed * elapsed)
	CameraPitch(CameraRotationY * CameraRotSpeed * elapsed)
	CameraRotationX = 0
	CameraRotationY = 0
	Posicio = getCameraPosition()
	if Posicio.z > 100 then
		Posicio.z = 100
		setCameraPosition(Posicio)
	elseif Posicio.z < -100 then
		Posicio.z = -100
		setCameraPosition(Posicio)
	end
end
```

3.2 Joc de conducció

```
Accel = 0
Brake = 0
Right = 0
MaxVel = 0
BallForce = 20000
Obstacles = { V(120,5,1000), V(200,5,950), V(220,5,1100), V(150,5,970), V(250,5,1050),
V(300,5,970), V(340,5,1120) }
ContadorTemps = 0

function startup()
	Gravity(V(0,-9.8,0))
	SkyBox("Core/Blue")
	Terrain("circuit2", V(1500,50,1500), V(0,0,0))
	LightAmbient(Color(1,0.9,0.9))
	Esfera(1, 50):crear("Bola", V(692,1,580))
	Force("Bola",V(0,0,-100))
	Obstacle = Cub(V(12,36,12), 5)
	Obstacle:material("Examples/RustySteel")
	for i=1,7 do
		Obstacle:crear("Obstacle" .. i, Obstacles[i])
	end
end

function MouseClicked(id, released)
end

function KeyClicked(id, released)
	if id == 200 then
		if released == 0 then Accel = 1
		else Accel = 0
		end
	elseif id == 208 then
		if released == 0 then Brake = -1
		else Brake = 0
		end
	elseif id == 203 then
		if released == 0 then Right = Right - 1
		else Right = Right + 1
		end
	elseif id == 205 then
		if released == 0 then Right = Right + 1
		else Right = Right - 1
```

```
                end
        end
end

function MouseMoved(x, y)
end

function FrameStarted(elapsed)
        ContadorTemps = ContadorTemps + elapsed
        BallVelocity = getVelocity("Bola")
        BallDirection = Normalize(BallVelocity)
        BallDirection.y = 0

        ResultingForce = V(0,0,0)
        if Brake == -1 and Magnitude(BallVelocity) > 5 then ResultingForce = BallDirection * Brake
end
        ResultingForce = ResultingForce + BallDirection * Accel
        ResultingForce = ResultingForce + Cross(BallDirection, V(0,Right,0))
        Force("Bola",ResultingForce * elapsed * BallForce)

        Vel = Magnitude(BallVelocity)
        Pos = getPosition("Bola")
        if Vel > MaxVel then MaxVel = Vel end
        Print("Temps: " .. ContadorTemps .. "\nVelocitat: " .. Vel .. "\nRecord: " .. MaxVel ..
"\nPosicio: " .. PrintVector(Pos) .. "\n")
        setCameraPosition(Pos + V(0,10,0) + BallDirection * -20)
        CameraLookAt(Pos + V(0,5,0) )
End
```

3.3 Joc d'estratègia

```
CameraDirection = V(0,0,0)
CameraMovSpeed = 80
CameraRotSpeed = -0.2
Cosa = "Ground"
acq = 0
follow = 0
HitLoc = V(0,0,0)
Personatges = { V(150,15,0), V(-50,15,0), V(0,15,-50), V(50,15,0), V(-50,15,100) }

function startup()
        Gravity(V(0,-9.8,0))
        SkyBox("Core/Evening")
        Floor()
        Plano = Pla(V(0,1,0), 1300, 1300)
        Plano:material("Examples/GrassFloor")
        Plano:crear("")
        LightAmbient(Color(1,1,1))
        setCameraPosition(V(0,400,0))
        CameraPitch(-90)
        Obs = Cub(V(10,7,15), 1)
        for i=1,5 do
                Obs:crear("Personatge" .. i, Personatges[i])
        end
end

function MouseClicked(id, released)
        if released == 0 then
                if id == 0 then
                        Cosa = getNameByRay(getCameraPosition(), getCameraDirection(), 9999)
                        acq = 1
                elseif id == 1 then
                        follow = 1
                end
        else
                if id == 1 then
                        follow = 0
                        if Cosa ~= "Ground" then
                                Stop(Cosa)
                        end
                end
        end
end
```

```
function KeyClicked(id, released)
	if id == 200 then -- UPARROW
		if released == 0 then CameraDirection.z = CameraDirection.z - 1
		else CameraDirection.z = CameraDirection.z + 1
		end
	elseif id == 208 then -- DOWNARROW
		if released == 0 then CameraDirection.z = CameraDirection.z + 1
		else CameraDirection.z = CameraDirection.z - 1
		end
	elseif id == 203 then -- LEFTARROW
		if released == 0 then CameraDirection.x = CameraDirection.x - 1
		else CameraDirection.x = CameraDirection.x + 1
		end
	elseif id == 205 then -- RIGHTARROW
		if released == 0 then CameraDirection.x = CameraDirection.x + 1
		else CameraDirection.x = CameraDirection.x - 1
		end
	end
end

function MouseMoved(x, y)
	CameraYaw(x * CameraRotSpeed)
	CameraPitch(y * CameraRotSpeed)
end

function FrameStarted(elapsed)
	CameraTranslate(CameraDirection * elapsed * CameraMovSpeed, 0)
	if follow == 1 then
		HitLoc = getPosByRay(getCameraPosition(), getCameraDirection(), 9999)
		if(acq == 1) then
			Distancia = getPosition(Cosa) - HitLoc
			forsa = Magnitude(Distancia)
			if forsa > 50 then
				forsa = 50
			end
			MoveTo(Cosa, HitLoc, forsa)
		end
	end
	Posicio = getCameraPosition()
	Posicio.y = 400
	setCameraPosition(Posicio)
	Print("Focus: " .. Cosa .. "\nPosicio: " .. PrintVector(HitLoc))
End
```

3.4 Joc d'acció en primera persona

```
CameraDirection = V(0,0,0)
CameraMovSpeed = 19
CameraRotSpeed = -0.2
PosicioInicial = V(750,50,750)
Enemics = { V(850,75,750), V(700,75,750), V(750,75,700), V(800,75,750), V(700,75,850) }
HitLoc = V(0,0,0)
DistanciaRecord = 0

function startup()
	Gravity(V(0,-9.8,0))
	SkyBox("Core/Cloudy")
	Terrain("exemple", V(1500,50,1500), V(0,0,0))
	LightAmbient(Color(0.75,0.75,0.75))
	setCameraPosition(PosicioInicial)
	Cub(V(2,2,2), 1.5):crear("Enemic", PosicioInicial)
	Timer("", 5, "SurtEnemic")
end

function SurtEnemic()
	Stop("Enemic")
	setPosition("Enemic", PosicioInicial)
	Force("Enemic", V(10,3,0) * 1000)
	Timer("", 3, "SurtEnemic")
	Print("")
end

function MouseClicked(id, released)
	if released == 0 then
```

```
		if id == 0 then
			Cosa = getNameByRay(getCameraPosition(), getCameraDirection(), 9999)
			HitLoc = getPosByRay(getCameraPosition(), getCameraDirection(), 9999)
			if Cosa == "Enemic" then
				LocalForce(Cosa, HitLoc, getCameraDirection() * 50000)
				Print(Cosa .. " ha estat impactat!")
			end
		end
	end
end

function KeyClicked(id, released)
	if id == 200 then
		if released == 0 then CameraDirection.z = CameraDirection.z - CameraMovSpeed
		else CameraDirection.z = CameraDirection.z + CameraMovSpeed
		end
	elseif id == 208 then
		if released == 0 then CameraDirection.z = CameraDirection.z + CameraMovSpeed
		else CameraDirection.z = CameraDirection.z - CameraMovSpeed
		end
	elseif id == 203 then
		if released == 0 then CameraDirection.x = CameraDirection.x - CameraMovSpeed
		else CameraDirection.x = CameraDirection.x + CameraMovSpeed
		end
	elseif id == 205 then
		if released == 0 then CameraDirection.x = CameraDirection.x + CameraMovSpeed
		else CameraDirection.x = CameraDirection.x - CameraMovSpeed
		end
	end
end

function MouseMoved(x, y)
	CameraYaw(x * CameraRotSpeed)
	CameraPitch(y * CameraRotSpeed)
end

function FrameStarted(elapsed)
	CameraTranslate(CameraDirection * elapsed, 0)
	Terra = getPosByRay(getCameraPosition() + V(0,10,0), V(0,-1,0), 9999)
	setCameraPosition(Terra + V(0,3,0))
	Distancia = Magnitude(getPosition("Enemic") - getCameraPosition())
	if Distancia > DistanciaRecord then
		DistanciaRecord = Distancia
	end
	Print("Distancia: " .. Distancia .. "\nRecord: " .. DistanciaRecord)
end
```

Estudi i implementació d'una plataforma de prototipatge de videojocs mitjançant la qual es pot crear un videojoc elemental, descartant aspectes decoratius o accessoris. Aquesta eina pretén millorar l'etapa de disseny d'un videojoc avançant el moment en que aquest es podrà jugar. Això permetrà prendre decisions importants en base a proves i experiències mesurables. S'ha implementat un sistema programable en llenguatge de script que estalvia a l'usuari treballar en els aspectes tecnològics i li permet centrar-se en crear la mecànica del joc que vol ser provat.

Estudio e implementación de una plataforma de prototipaje de videojuegos mediante la cual se puede crear un videojuego elemental, descartando aspectos decorativos o accesorios. Esta herramienta pretende mejorar la etapa de diseño de un videojuego adelantando el momento en el que este se podrá jugar. Esto permitirá tomar decisiones importantes en base a pruebas y experiencias mesurables. Se ha implementado un sistema programable en lenguaje de script que ahorra al usuario trabajar en los aspectos tecnológicos y le permite centrarse en crear la mecánica del juego que quiere ser probado.

Study and implementation of a videogame prototyping platform through which an elemental videogame can be created, discarding decorative or accessory aspects. This tool intends to improve the design phase of a videogame bringing forward the moment it will be playable. This will empower important decision taking based of proofs and experiences that can be measured. A system programmable through script language saves the user from working in technological aspects and allows it to focus on creating the game mechanics to be tested.