Universitat Autònoma
de Barcelona

DEMO TÈCNICA D'UN VIDEOJOC

Memòria del projecte

d'Enginyeria Tècnica en

Informàtica de Sistemes

realitzat per

Alfons Mallol Garcia

i dirigit per

Marc Talló Sendra

Escola Universitària d'Informàtica

Sabadell, *Juny* de *2010*

El/la sotasignant, *Marc Talló Sendra*,

professor/a de l'Escola Universitària d'Informàtica de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria
ha estat realitzat sota la seva direcció
per en *Alfons Mallol Garcia*

I per a que consti firma la present.
Sabadell, *Juny* de *2010*

Signat: *Marc Talló Sendra*

Resum

És un fet que cada cop l'industria dels videojocs pren més força en el mercat internacional fent-se un bon lloc en l'àmbit de l'entreteniment personal. Cada dia surten més productes de formats molt variats destinats a tota mena de públic. El requeriment de professionals que sàpiguen com crear aquests videojocs va en augment a la vegada que els perfils d'aquests també es van diversificant.

El projecte que aquí tractem és un dels videojocs creats durant el primer màster de creació de videojocs de la UAB. El videojoc en qüestió es tracta d'una casa encantada controlada pel jugador on haurà d'espantar als seus habitants fins fer-los fora. Aquest projecte està portat a terme per un grup de quatre persones, dues artistes i dos programadors (un dels quals soc jo), format un equip interdisciplinar.

Aquesta memòria tracta sobre el punt de vista del programador sense deixar de banda la feina de les artistes. S'ha creat tot el necessari partint des de cero fins assolir el producte que actualment es veu. Tot i que continuarà sent millorat durant el futur. També es parla dels problemes i avantatges de treballar en equip i com s'ha anat fent encaixar tot plegat.

Vista general del menjador amb les dues víctimes, l'Emma i el David.

(Aquesta pàgina s'ha deixat intencionadament en blanc.)

Índex

(Aquesta pàgina s'ha deixat intencionadament en blanc.)

1. Introducció

1.1. Presentació

El principal objectiu d'aquest projecte es crear una versió demostrativa d'un videojoc creant inclús tot el seu motor, també conegut com a demo tècnica. La durada d'aquesta és d'entre cinc i deu minuts de joc. Aquest projecte també forma part del primer màster de creació de videojocs de la UAB sent un dels dos que s'han creat i seran presentats de cara el curs vinent. Durant el màster els professors ens van donar les eines i orientacions necessàries però sempre ens han deixat total llibertat per implementar el videojoc segons els nostres gustos i criteris.

El joc que s'ha implementat consisteix en que el jugador porta una casa encantada on entra a viure un sèrie de gent, començant per una parella jove. L'objectiu és fer-los fora a base d'espantar-los de diferents maneres emprant diferents trucs que hi ha amagats per tota la casa, des de moure mobles a llençar objectes sense oblidar-se de fantasmes i molts sorolls variats. El jugador té un límit de temps, cinc minuts per se exactes, que representa el que falta perquè surti el sol, si no ho aconsegueix, perdrà. També perdrà si els mata de massa espantar-los.

El projecte s'ha dividit en dues parts molt clares. Per una banda tenim la creació del motor que ha de ser exportable a diferents plataformes i reutilitzable per a futurs projectes. Aquest motor s'encarrega de tots els aspectes del videojoc, des del control de gràfics passant pel so i la física sense oblidar el control d'entrada com el teclat i el ratolí. És cert que es podria haver optat per un motor comercial ja creat però un dels objectius d'aquest projecte és crear i controlar la producció des de l'inici fins al final.

Per l'altra banda tenim el joc en sí, que es una implementació especifica del motor. Aquesta part s'encarrega de dir que i quan s'ha de fer i que passa quan el jugador fa alguna acció. El motor s'encarrega de com funcionen les coses i el videojoc el que, quan i, sobre tot, que són les coses.

1.2. Estat de l'art

Si parlem de motors de joc n'hi ha molts i de molts diversos però en destacaria dos en particular, el *Unity3D* i el *UDK*. Aquests dos són els més famosos que hi ha en el mercat.

El *Unity3D*, o simplement *Unity*, es tracta d'un motor comercial amb una versió gratuïta que porta implementat tot el necessari per crear un videojoc. També té

una amplia capacitat multiplataforma, qualsevol programa creat amb l'*Unity* es pot executar amb Windows, MAC, Web, iPhone, iPad, PS3, Xbox 360 i Wii sempre i quan tinguis en compte les limitacions tècniques de cada plataforma.

L'altre motor a destacar és el *Unreal Development Kit*, altrament conegut com a *UDK*, també és un motor comercial i multiplataforma però més limitada ja que només funciona per Windows, PS3 i Xbox 360. Igual que l'*Unity* també porta implementat tot el necessari per crear un videojoc però el seu preu depèn de l'ús que se'n faci i les ventes que tinguis però si el que es vol és crear un prototip o una demo no has de pagar res.

Ara bé, si parlem del joc en sí, la casa encantada, ens trobem que no hi ha res semblant en el mercat actual. Temps enrere sí que es van fer jocs similars però no igual ja que les accions es realitzaven a través de fantasmes i altres activitats paranormals.

1.3. Objectius

Els objectius del projecte han estat molt clars des del primer dia:

- Crear i desenvolupar un motor de joc amb capacitat de ser multiplataforma i reutilitzable per a futurs videojocs.
- Crear i desenvolupar una demo tècnica en base al motor creat i que aquesta també representi un projecte sòlid d'un futur videojoc complert i que sigui divertit i entretingut.
- Aprendre i familiaritzar-se tots els processos de la creació d'un videojoc així com treballar en grup, fixar-se terminis i repartiment de tasques.

La consecució d'aquest objectius és la clau per l'èxit del projecte al tractar-se d'una feina en grup compost per perfils molt diversos, molt marcadament n'hi ha dos, els artistes i els programadors.

1.4. Motivacions

Sempre he tingut clares les meves tendències a l'industria dels videojocs. Considero un repte molt important la programació d'aquests tipus d'aplicacions ja que s'ha de ser molt meticulós i detallista. Sempre buscant com estalviar fraccions de segon per poder millorar el rendiment. Intentant estalviar memòria per agilitzar els temps de

carga i moltes coses més com són les configuracions i implementacions de la mecànica de joc, on també hi entra la IA.

A un nivell més concret, un videojoc és un projecte d'una gran envergadura. Aquest que ens ocupa esta format per moltes classes i objectes que han d'interactuar entre elles a més d'utilitzar varies capes per mantenir la capacitat d'adaptació tant del motor com del videojoc en sí. Això també és un repte important a superar, sobre tot si tenim en compte que ha de funcionar tot com una seda. Com sempre he dit "no és el mateix que un botó del Word trigui un segon més del necessari que el d'un videojoc", això podria representar la diferencia entre que es pugui jugar i sigui divertit a que no sigui res d'això.

Aquest projecte representa la meva carta de presentació al món laboral tant per mi com pels meus companys del grup. Sense oblidar que també és la demostració de la formació rebuda durant tota la carrera.

1.5. Estructura de la memòria

Per tal de facilitar la comprensió de la memòria s'ha estructurar de la següent forma:

- Estudi de viabilitat: Es determina la viabilitat del projecte en base dels objectius determinats, l'estudi actual del sector i les seves demandes, les especificacions (tant funcionals i no funcionals com les tècniques), la planificació (tant a nivell d'hores invertides com a nivell econòmic) i els riscos externs.
- Fonaments teòrics: Explicació dels conceptes tecnològics per entendre els desenvolupament del projecte.
- Anàlisis: Descripció del disseny i funcionament del projecte i com es duran a terme les probes.
- Implementació: Descripció de les diferents fases d'implementació durant el desenvolupament.
- Proves: Realització de les proves necessàries per comprovar que el videojoc resultant funciona correctament i també que sigui entretingut.
- Conclusions: Descripció de les conclusions finals que es poden extreure del projecte així com les complicacions que s'han anat trobant durant el seu transcurs.
- Bibliografia: S'especifiquen les fonts d'informació emprades pel correcte desenvolupament del projecte.

- Apèndix: es detalla un glossari de termes emprats durant la memòria i el document de disseny inicial que es va utilitzar per començar el desenvolupament del projecte.

2. Estudi de viabilitat

2.1. Introducció

Aquí s'explica els aspectes fonamentals del projecte determinant els objectius als quals es vol arribar, l'estat de l'art, les especificacions, els perfils del sistema, els recursos necessaris, la planificació del treball, els riscos i, finalment, les conclusions.

2.2. Objectius

Per poder portar a bon port el projecte s'han de complir els següents objectius:

- Creació d'un motor de joc que englobi tota la funcionalitat relacionada amb interfície humana, gràfics en 2D per la GUI i 3D pel el transcurs del joc, física aplicable, so i música, control d'errors ja siguin per part del programador o de l'artista o del jugador.
- Creació d'un visualitzador perquè els artistes gràfics puguin provar els seus models 3D sobre el motor sense necessitat d'esperar que el joc o el motor estiguin en una fase avançada i puguin crear sense trencar el ritme de treball.
- Eines de proves del motor per minimitzar els nombres de proves que es faran en el videojoc.
- Creació de models 3D de baix poligonatge per ser usats en el videojoc. També s'han de crear la seva versió amb molt baix poligonatge, és a dir, simples caixes, per les proves del joc.
- Creació de l'escenari de joc implementant les càmeres necessàries pel seu visionat.
- Creació de l'art per la GUI creant les icones necessàries per la correcte interacció amb l'usuari.
- Implementar la mecànica de joc fent casar tots els models 3D amb la GUI i aconseguir una primera instancia de la versió jugable.

2.3. Estat de l'art

S'ha buscat altres videojocs sobre cases encantades que espantin els seus habitants però en l'actualitat no hi ha res i els més assemblat que s'ha trobat són alguns videojocs en que el jugador porta fantasmes o altres entitats paranormals per espantar. En aquest aspecte no hi ha res semblant.

Però si que hi ha sobre el motors de joc. El mercat n'hi ha de diferents menes, els més importants actualment son el *Unity3D* i el *UDK*. De totes formes, l'elecció d'un d'aquest motors en lloc de crear el nostre només afectaria a un dels objectius però crearia la necessitat d'aprendre'n l'ús del que triéssim. A més a més, quedaríem lligats a les seves limitacions sense tenir l'opció d'explorar altres vies com, per exemple, exportar a altres plataformes que no suportessin.

- **Unity3D**, o simplement *Unity*, es tracta d'un motor comercial amb una versió gratuïta que porta implementat tot el necessari per crear un videojoc. També té una amplia capacitat multiplataforma, qualsevol programa creat amb l'*Unity* es pot executar amb Windows, MAC, Web, iPhone, iPad, PS3, Xbox 360 i Wii sempre i quan tinguis en compte les limitacions tècniques de cada plataforma.
- **UDK**, abreviació de *Unreal Development Kit*, altrament conegut com a *Unreal Engine*, també és un motor comercial i multiplataforma però més limitada ja que només funciona per Windows, PS3 i Xbox 360. Igual que l'*Unity* també porta implementat tot el necessari per crear un videojoc però el seu preu depèn de l'ús que se'n faci i les ventes que tinguis però si el es vol és crear un prototip o una demo no has de pagar res.

2.4. Especificacions

2.4.1. Funcionals

- Capacitat per entretenir a l'usuari.
- Demostració de la capacitat dels integrants del grup de crear un videojoc.
- Poder explicar una idea de joc de forma ràpida i concisa.

2.4.2. No funcionals

- Qualitat gràfica adaptada als nivells exigits actualment.
- Qualitat sonora que empri els estàndards de so actuals en 3D.
- Poder ser exportat a diferents plataformes.
- Tenir la capacitat de localitzar-se a diferents idiomes.
- Fer-lo el més accessible possible tenint en compte possibles minusvalideses dels usuaris.

2.5. Perfils

Normalment, en un videojoc només hi ha un tipus de perfil d'usuari però en podríem tenir dos si tenim en compte el requeriments no funcionals:

- Jugador: és qui engegarà el joc i passarà una bona estona espantant a la parella que només volia viure tranquil·la.
- Adaptador: és qui s'encarregarà d'adaptar el videojoc a les necessitats especifiques d'algun tipus de jugador sense la necessitat que intervingui algú de l'equip de desenvolupament.

2.6. Recursos

El requisits de hardware són iguals per tothom però el software que es necessita varia segons si és pel jugador, pel programador o per l'artista. Ens cenyim a la versió per Windows ja que ha estat la implementada.

2.6.1. Hardware

- Intel Pentium 4 o similar.
- Targeta gràfica compatible amb Shaders 3.0.
- Targeta de so 3D.
- Memòria RAM: 512 MB
- Espai al disc dur: 1 GB

2.6.2. Software del programador

El requisits de software pel programador són:

- *Windows XP*
- *DirectX 9.0 SDK*
- *PhysX SDK*
- *Visual Studio 2005*
- *3DSMax 2010*

2.6.3. Software de l'artista 3D

El requisits de software per l'artista 3D són:

- *Windows XP*
- *DirectX 9.0 o superior*

- *PhysX*
- *3DSMax 2010*
- *ZBrush*

2.6.4. Software del jugador

El requisits per poder gaudir de l'aplicació són:

- *Windows XP*
- *DirectX 9.0*
- *PhysX*

2.6.5. Recursos humans

Pel desenvolupament del projecte es defineixen els següents perfils:

- Dissenyador de joc: s'encarrega de crear les mecàniques de joc.
- Programador: implementa el software necessari per crear el motor i el videojoc amb la seva mecànica.
- Artista: crear l'art visual del videojoc, tant en 2D com en 3D.

2.7. Planificació

2.7.1. Model de desenvolupament

Per el desenvolupament del projecte s'ha usat un model lineal amb les següents fases:

a) Document de disseny: escrit amb les especificacions de jugabilitat, aspectes gràfics i mecànica de joc entre altres punts a tenir en compte a la creació del videojoc.
b) Establir la base del motor de joc i crear un visualitzador i un exportador pels artistes 3D.
c) Crear escenari de joc a nivell artístic.
d) Desenvolupament complet del motor de joc.
e) Crear els objectes mòbils i animats a nivell artístic.
f) Implementació de la mecànica de joc.
g) Implementació de la interfície gràfica.
h) Proves de joc.

2.7.2. Planificació temporal

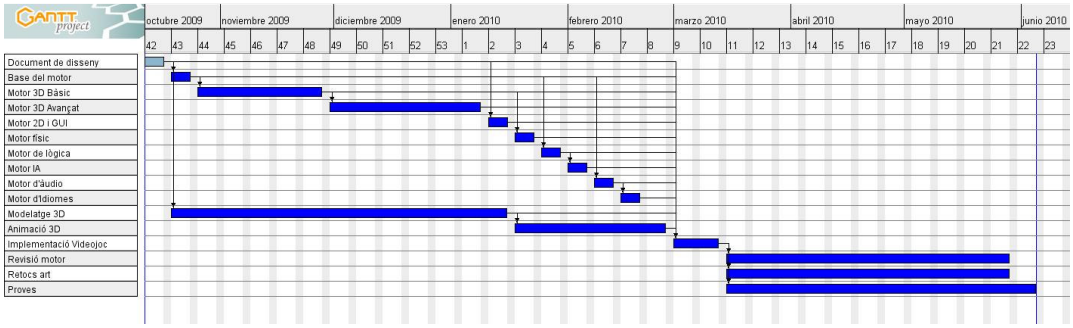
Per controlar correctament el temps i coordinar els esforços de tot el grup per evitar que algú del grup no pugui aprofitar el temps.

A la següent taula s'especifiquen les diferents tasques que té el projecte així com el seu perfil responsable, destacar que les feines de disseny i fer les proves és una tasca del grup:

Tasca	Perfil	Duració (en hores)
Document de disseny	Grup	24
Base del motor de joc	Programador	36
Motor 3D	Programador	185
Motor 2D i GUI	Programador	36
Motor físic	Programador	36
Motor de lògica i scripting	Programador	24
Motor IA	Programador	24
Motor d'àudio	Programador	12
Modelatge 3D	Artista	500
Animació 3D	Artista	150
Modelatge 2D	Artista	30
Implementació videojoc	Programador	280
Proves	Grup	180
Total grup		**204**
Total programador		**633**
Total artista		**680**
TOTAL		**1.5173034**

Taula 1: Tasques, perfils i duració del projecte

El que ens dona el següent diagrama de Gantt



Il·lustració 1: Diagrama Gantt del projecte.

La meva feina en particular estava molt lligada a l'altre programador durant el desenvolupament inicial del motor. Per norma general, cada un s'encarregava d'una part del codi que després havíem d'ajuntar. Quan un feia l'exportador, l'altra feia l'importador, quan un preparava el motor 3D per funcionar amb el hardware, l'altra feia el *shaders*. Al arriba a etapes més avançades cadascú es cuidava de l'apartat que havia tocat.

2.7.3. Planificació de costos

En aquest apartat s'estimen les despeses econòmiques contemplades en els sous dels diferents responsables, les despeses de hardware emprat i el software necessari.

A la taula següent es mostra la despesa del projecte en funció del sou dels diferents responsables.:

Perfil	Cost/hora	Hores	Cost (en €)
Artista	12	680	8.160
Programador	12	633	7.596
TOTAL			**15.756**

Despeses dels responsables del projecte

A continuació es mostren les despeses relacionades amb el hardware:

Material	Cost (en €)
PC de desenvolupament (x4)	1.600 (400 unitat)
TOTAL	**1.600**

Despeses de hardware

En la taula següent es detallen les despeses del software:

Material	Cost (en euros)
Sistema operatiu	200
Eines del programador	150
Eines de l'artista	5.500
Eines ofimàtiques	200
TOTAL	**6.050**

Despeses de software

I, finalment, s'especifiquen les despeses finals:

Material	Cost (en €)
Personal	15756
Hardware	1600
Software	6050
Consumo de electricitat	300
Internet (ADSL)	180
TOTAL	**23886**

Despeses totals

2.8. Riscos

S'han de tenir en compte els riscos que existeixen a tots els nivells tant tècnics com de costos i de temps:

- Coneixements tècnics: existeix la possibilitat que es vulgui implementar alguna cosa que sigui o molt costosa a nivell de programació o a nivell d'art i s'hagi de remodelar el projecte.
- Factor econòmic: si el projecte no es aprovat per convertir-se en un videojoc complert representaria una gran pèrdua de diners.
- Factor de temps: un retard en alguna de les tasques implica que les demes tasques també s'enderiaran.

2.8.1. Alternatives

Per reduir el riscos associats es mostren les següents opcions:

- Emprar un motor comercial. Reduiria el temps necessari per crear el nostre però ens limitaria el mercat.
- Comprar art ja creat. Reduiria el temps de creació d'art 3D i 2D però no s'adaptaria al joc.

2.9. Valoració

Després de finalitzar l'estudi de viabilitat es consideren les següent valoracions:

- Planificar clarament les fases i les tasques a dur a terme cada membre del grup és imprescindible per poder coordinar-se i no perdre temps.
- Conèixer les tècniques i diferents llenguatges que es poden emprar ajudà a agilitzar i millorar el producte final.

2.10. Conclusions

Arribats aquest punt podem extreure les següents conclusions:

- El projecte ofereix la possibilitat de presentar un producte innovador així com una bona mostra de la capacitat del grup involucrat en el seu desenvolupament.
- Es pot considera que el projecte és viable considerant la llibertat d'acció i de mercat que tenim.

3. Fonaments teòrics

3.1. Introducció

Abans de tot hem d'entendre quines han estat les eines emprades pel desenvolupament del software d'aquest projecte. Primer s'expliquen les llibreries externes i després els llenguatges de programació.

Pel bon desenvolupament del projecte es copien les llibreries externes en un directori apart anomenat 3dParty. Aquí es copien totes excepte el DirectX i el PhysX que s'han d'instal·lar a l'equip de desenvolupament.

3.2. DirectX 9.0

Es un conjunt d'APIs creades per Microsoft per facilitar totes les tasques relacionades amb els aspectes multimèdia, especialment per videojocs. Funciona per les diferents versions de Windows tenint retrocompatibilitat. Una de les principals avantatges d'emprar DirectX ens que ens podem despreocupar sobre quin hardware estem funcionant sempre i quan sigui compatible amb la versió de DirectX que estem utilitzant però donat que tots els PCs actuals son compatibles amb Windows es un problema que es pot ometre, sobre tot si tenim en compte que la versió 9.0 té més de set anys.

Es va triar la versió 9.0 perquè va ser l'última que va sortir per Windows XP. Les versions posteriors ja no son compatibles amb aquest sistema operatiu i necessiten o Windows Vista per la versió 10 o Windows 7 per la versió 11.

Aquest conjunt d'APIs ens permet interactuar de forma fàcil i senzilla amb els recursos del PC. Destaquem dues d'elles, totes emprades en aquest projecte:

- Direct3D: Ens permet interactuar amb la targeta enviant-li tota la informació de polígons i textures així com els shaders que volem emprar.
- DirectInput: Podem llegir les entrades de teclat, ratolí o gamepad de forma estàndard sense tenir en compte el seu fabricant.

S'ha d'instal·lar el SDK del DirectX i després incloure els directoris i llibreries addicionals per poder treballar amb ell.

3.3. PhysX

Es tracta d'una llibreria de física d'Nvidia. Creada inicialment per Aegia per funcionar amb un hardware específic anomenat PPU (Physic Proces Unit) que permetia càlculs de física a temps real massa complexes per una CPU. Aquesta empresa va ser comprada per Nvidia, fabricant de hardware gràfic, i va integrar aquesta tecnologia a les seves GPUs.

Aquestes llibreries ens permeten fer càlculs físics des de la detecció de col·lisions fins a rebots d'objectes impactant entre ells, passant per moviment de roba o un cotxe en marxa simulant la rotació de les seves rodes i com afecte el terreny a la seva suspensió. Pel projecte necessitàvem una bona forma de trobar les col·lisions entre objectes a més de les coses que volaven i després queien.

Per poder utilitzar *PhysX* hem d'instal·lar el programari de desenvolupament, el SDK. Un cop instal·lades hem d'incloure els directoris i les llibreries addicionals que es troben a la carpeta on s'ha instal·lat el *PhysX*. També s'han de copiar les llibreries *NxCharacter.dll*, *NxCooking.dll* i *PhysXLoader.dll* en el directori d'execució.

3.4. Cal3D

Llibreria Open Source dedicada a l'animació esqueletal. Tot i que el DirectX ens dona les eines necessàries per dibuixar un model 3D no en tenim per fer-lo animat com, per exemple, un home caminant, per això s'ha de recorre a una solució externa. Cal3D és una llibreria independent que funciona tant amb OpenGL com en DirectX. També permet la interpolació d'animacions fent els canvis més suaus.

La tècnica d'animació esqueletal consisteix en forma un esquelet on cada vèrtex del model 3D farà referència. Després es mou l'esquelet i els vèrtexs es posicionen seguint el patró establert. El moviment de l'esquelet es fa via software, és a dir, el posiciona la CPU mentre que la posició del vèrtexs es fa via hardware, és a dir, la targeta gràfica deforma el model segons la posició de l'esquelet.

Per poder incorporar el Cal3D al projecte cal incorporar-la tota sencera en la solució com un projecte més ja que s'ha de compilar. Després s'ha d'encapsular com les restes de llibreries externes. També s'han de copiar les DLLs *cal3d.dll* i *cal3d_d.dll* al directori d'execució.

3.5. Bass

Llibreria de so externa multiplataforma, molt fàcil d'usar. S'encarrega de gestionar els recursos de so on només li has d'especificar quins sons hi haurà i quan sonen. Suporta tant so 3D com estèreo o mono, el que ens permet, per exemple, fer escoltar una passes que s'allunyen (un so 3D) mentre es sent com plou (un so mono, ambiental) i de fons una música per l'escena (un so estèreo, també ambiental).

Per poder treballar amb el Bass s'ha d'incloure el directori a on es faci l'encapsulament i copiar la llibreria *bass.dll* al directori d'execució.

3.6. Libxml

Llibreria per la lectura i escriptura de fitxers XML. En un videojoc hi ha molts paràmetres que estan escrits en fitxers XML per no haver de posar els valors dins el codi ja que cada cop que es volgués canviar alguna cosa obligaria a recompilar el projecte amb la seva conseqüent pèrdua de temps. Al tenir-ho en fitxers XML no només ens estalvia la recompilació sinó que també ho podem fer en calent, sense necessitat de tancar i tornar a obrir el programa.

Per poder-la utilitzar s'ha d'introduir el directori d'inclusió addicional (Propiedades de configuración -> C/C++ -> General -> Directorios de inclusión adicionales) on estan la llibreria Libxml i copiar els arxius *iconv.dll* i *libxml2.dll* al directori d'execució. Després es fa l'encapsulament que es vulgui crear.

3.7. Lua, LuaBind i Boost

Aquestes tres llibreries juntes ens permeten usar el llenguatge d'*scripting* Lua en el nostre videojoc amb capacitat per tractar tot tipus d'informació. El *Lua* bàsic no ens permet emprar classes però juntament amb el *LuaBind* sí que ho podem fer però aquest necessita el conjunt de llibreries *Boost* per poder funcionar.

Com que es tracta d'un llenguatge extensible podem incorporar les nostres pròpies funcions. A continuació poso el codi per registrar les funcionalitats del *SoundManager* per ser usades posteriorment en codi Lua.

ResgisterSoundManager.h

```
#pragma once

#ifndef _REGISTER_SOUND_MANAGER_H_
#define _REGISTER_SOUND_MANAGER_H_

#include "Scripting/ScriptBase.h"

class CRegisterSoundManager : CScriptBase
{
  public:
    CRegisterSoundManager();
    ~CRegisterSoundManager();
    void Register();
};

#endif
```

RegisterSoundManager.cpp

```
#include "RegisterSoundManager.h"

#include "luabind/luabind.hpp"
#include "SoundManager.h"
#include "Scripting/ScriptManager.h"
#include "Core.h"

using namespace luabind;

CRegisterSoundManager::CRegisterSoundManager()
{
}

CRegisterSoundManager::~CRegisterSoundManager()
{
}

CSoundManager * GetSoundManager()
{
  return CCore::GetSingleton().GetSoundManager();
}

void CRegisterSoundManager::Register()
{
  REGISTER_LUA_FUNCTION("get_sound_manager",&GetSoundManager);

  module(LUA_STATE) [
    class_<CSoundManager>("CSoundManager")
      .def("reload", &CSoundManager::ReLoad)
      .def("play_fade_in", &CSoundManager::PlayFadeIn)
      .def("play_sample", &CSoundManager::PlaySample)
      .def("play_3d_sample", &CSoundManager::Play3DSample)
      .def("set_channel_3d", &CSoundManager::SetChannel3D)
      .def("stop_channel", &CSoundManager::StopChannel)
      .def("set_3d_position", &CSoundManager::Set3DPosition)
      .def("set_volume", &CSoundManager::SetVolume)
```

```
      .def("get_volume", &CSoundManager::GetVolume)
      .def("start", &CSoundManager::Start)
      .def("pause", &CSoundManager::Pause)
      .def("get_error_code", &CSoundManager::GetErrorCode)
  ];
}
```

La capçalera hereta d'una classe genèrica que té el destructor virtual i el mètode *Register()* virtual pur i només necessita incloure el *ScriptBase.h*. Es important deixar el fitxer .h tant net com es pugui per agilitzar el temps de compilació.

El mètode *Register()* és més important ja que s'encarrega de que els mètodes de C/C++ de la classe a tractar puguin ser cridats des de *Lua*. També podem crear nous mètodes que no estiguin a la classe original, una de les més comuns és la que ens permet demanar el seu punter d'una forma simple.

El fitxer .cpp es veu obligat d'incloure, a part de la seva capçalera, el *Luabind*, la classe o classes que registrarem i l'*ScriptManager*, a més d'altres fitxers que es necessitin, en aquest cas el *Core.h* per poder accedir als punters.

Un cop tenim creats els arxius per fer l'enregistrament només falta cridar-los amb les següent instrucció que es crida en el *Init* del *Core.cpp*:

```
m_pScriptManager->AddScript((CScriptBase*)new CRegisterSoundManager());
```

On *m_pScriptManager* és el punter de referència al manager de tot l'*script* que hi ha en el videojoc.

Un exemple d'aquesta implementació està en el fitxer *intro.lua* on s'utilitza molt el manager de sons per engegar i parar els diàlegs. A continuació poso la funció f_intro_02():

```
function f_intro_02()
      get_sound_manager():stop_channel(voiceChannel)

      get_scene_manager():change_room_and_camera("Nen", 1)
      emma:switch_to_animation(parlar,0.2)
      voiceChannel = get_sound_manager():play_sample("Emma1",1.0)
end
```

Aquesta funció para l'anterior veu (en el cas que encara estigués sonant), després canvia d'habitació i càmera, activa l'animació de parlar de l'Emma (una de les víctimes) i, finalment, fa sonar el diàleg Emma1. Destacar que *emma* es una variable local que fa referència una de les víctimes.

Per poder incorporar totes les llibreries Lua s'ha d'incorporà les llibreries *Lua* i *Luabind* com a projectes dins la solució. A més, a la llibreria *Luabind* se li ha de posar com a directori addicional d'inclusió el *Lua* i les llibreries *Boost*.

3.8. Llenguatges de programació

Pel desenvolupament del projecte s'han usat diferents llenguatges per diferents tasques. Alguns d'aquest llenguatges són molt específics, com el HLSL, o molt més genèrics, com el C++.

3.8.1. C/C++

La base del projecte. Tot i que es podria haver triat altres llenguatges orientats a objecte però el C++ ens qui ens permet millor control sobre tot el recursos el qual ens deixa optimitzar molt més que qualsevol altre llenguatge. Recordar que l'optimització és molt important, un retard de una dècima de segon pot significar una important pèrdua de qualitat del producte final.

3.8.2. HLSL

Són les inicials de *High Level Shader Language*, llenguatge per programar la targeta gràfica. Es divideix en dos processos el *vertex shader* i el *pixel shader*.

El *vertex shader* actua sobre el vèrtex del model 3D que s'ha de dibuixar i ens permet fer varies opcions. Aquí és quan els models 3D animats s'adapten a l'esquelet.

El *pixel shader* actua sobre el píxels que es dibuixaran del model 3D. Aquí també s'apliquen molt dels efectes més comuns com la brillantor especular o l'efecte de relleu *bump mapping*.

També permet combinar tècniques amb diferents passades el que ens permetria fer aures, *cell shading* o onades entre altres efectes.

3.8.3. Lua

Si el fitxers XML ens permeten canviar paràmetres sense necessitat de tocar el codi i tornar a compilar i executar, el Lua ens permet fer el mateix amb part de la

programació. Sense necessitat de canviar res del codi podem canviar gran part de la funcionalitat, especialment la de videojoc

Gracies al Lua combinat amb XML, per exemple, podem crear els botons de d'interfície de l'usuari. Podem moure els botons i la seva funcionalitat en calent reduint significativament el temps necessari en les proves.

3.8.4. MaxScript

Es un llenguatge de programació emprat per el 3D Studio Max. Amb aquest llenguatge vam crear l'exportador de models 3D per adaptar-lo al motor que s'havia creat. Permet la lectura i l'escriptura de fitxers així com la lectura de tots els paràmetres del model o escena 3D a tractar.

(Aquesta pàgina s'ha deixat intencionadament en blanc.)

4. Anàlisis

4.1. Introducció

Aquí es detalla l'anàlisi i disseny del sistema. Es verifica el compliment dels requeriments funcional i no funcionals. També es detallen els casos d'us.

4.2. Requeriments funcionals

Verifiquem els requeriments funcional estipulats:

- Capacitat per entretenir a l'usuari.

La mecànica de joc implementada assegura que el jugador no es podrà trobar encallat en ningun lloc i sempre podrà continuar jugant.

També es procura que d'interfície d'usuari sigui amena i fàcil de comprendre. Qualsevol jugador hauria de ser capaç de posar-se davant del joc i a l'agafar el ratolí pogué començar a interactuar amb la casa i els seus elements.

L'estètica del joc és d'estil dibuix animat amb els personatges però mantenint un cert realisme amb la casa. Aquesta combinació ens permet fer més creïbles les reaccions exagerades de les víctimes alhora que el jugador es troba dintre d'una casa de pel·lícula de terror clàssica.

- Demostració de la capacitat dels integrants del grup de crear un videojoc.

Al tractar-se d'una demo tècnica és molt important deixar clar que el producte resultant no és "una joc més" i cal remarcar la innovació implícita a part de demostrar les capacitats professionals que s'espera d'un projecte d'aquest tipus.

Per tal de dur a terme això es busca una idea no gaire utilitzada en l'actual mercat de videojocs. A més, es crea un motor de joc totalment nou amb les optimitzacions necessàries per aquest projecte mentre es conserva la capacitat d'adaptació a altres plataformes.

- Poder explicar una idea de joc de forma ràpida i concisa.

Al ser una demostració d'un producte, s'ha de poder veure aquest producte de forma ràpida i concisa. El joc no dura més de cinc minuts i des de el primer moment s'ha de poder jugar. Tot això fa que es pugui veure l'idea general des del primer minut.

4.3. Requeriments no funcionals

Verifiquem els requeriments no funcionals estipulats:

- Qualitat gràfica adaptada als nivells exigits actualment.

Tot i que es tracta d'una demo i que no hi ha pressupost per fer models 3D molt detallats, s'ha de crear tant un motor gràfic com un models 3D suficientment detallats amb suficient qualitat perquè no semblin d'una època passada.

Amb les eines adequades juntament amb una bona programació es pot aconseguir que els models animats puguin visualitzar-se juntament amb els models estàtics de l'escenari de forma correcte, amb les llums corresponents i les ombres adequades.

- Qualitat sonora que empri els estàndards de so actuals en 3D.

El so actual en el videojocs té una component 3D amb la seva corresponent atenuació per la distancia. També s'ha de permetre el so estèreo per la música i sons d'ambient i també mono per altres sons ambient i veus.

- Poder ser exportat a diferents plataformes.

Crear el propi exportador de models juntament amb l'ús de textures estàndards és quelcom que s'ha de tenir en compte per poder fer l'adaptació a múltiples plataformes.

Tanmateix, la programació de tot el projecte ha de tenir les suficients capes i encapsulacions per tal que el canvi de plataforma comporti el mínim de canvi de codi necessari.

- Tenir la capacitat de localitzar-se a diferents idiomes.

Tots els escrits han de poder ser traduïts i adaptats a altres idiomes. L'ús de fitxers XML per cada idioma juntament amb la capacitat de canviar l'opció de quin fitxer es carrega permet canviar d'idioma fàcilment sense necessitat de tocar el codi.

- Fer-lo el més accessible possible tenint en compte possibles minusvalideses dels usuaris.

Els sons tenen les seves respectives onomatopeies que estan juntament amb els arxius XML d'idiomes. Això el fa més accessible a la gent amb problemes d'oïda.

El joc ha de poder ser usat amb una mà, per tant, una interfície que permeti tot el control només amb el ratolí o amb una petita part del teclat s'adaptaria perfectament a la gent que li falti alguna mà o alguns dits.

4.4. Estructura d'un Manager

S'han utilitzat molts managers pel desenvolupament del projecte. Aquests ens permeten estalviar molta memòria i agilitzar els càlculs reduint així els costos computacionals i optimitzar el projecte. D'altra banda també ens deixa el codi molt més entenedor facilitant així les possibles futures adaptacions que s'hagin de realitzar.

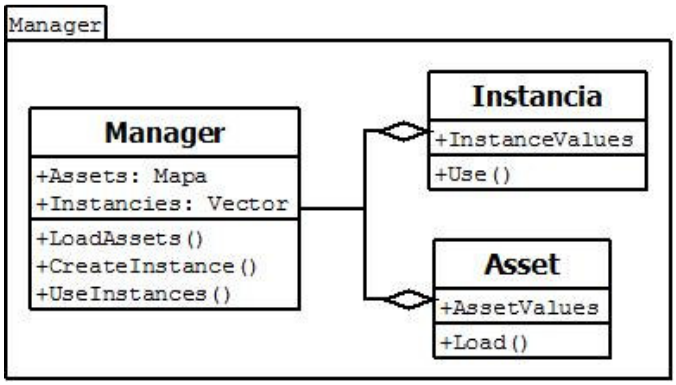


Diagrama base d'un manager.

El manager llegeix els fitxers, coneguts com a *assets*, i en carrega en memòria una sola copia independentment de les vegades que es necessiti durant el transcurs del programa. Aquestes copies son guardades en un mapa per el seu fàcil accés. Quins fitxers són ho diu un fitxer XML el qual ens permet canviar amb facilitat que volem carregar i que no.

Un cop carregades les dades dels *assets*, creem les seves instancies segons la informació que podem llegir en un XML, normalment és el mateix que els dels *assets* però amb *tags* diferents, i les registrem dins d'un vector ja que aquest ens permet una millor forma de recorre les dades.

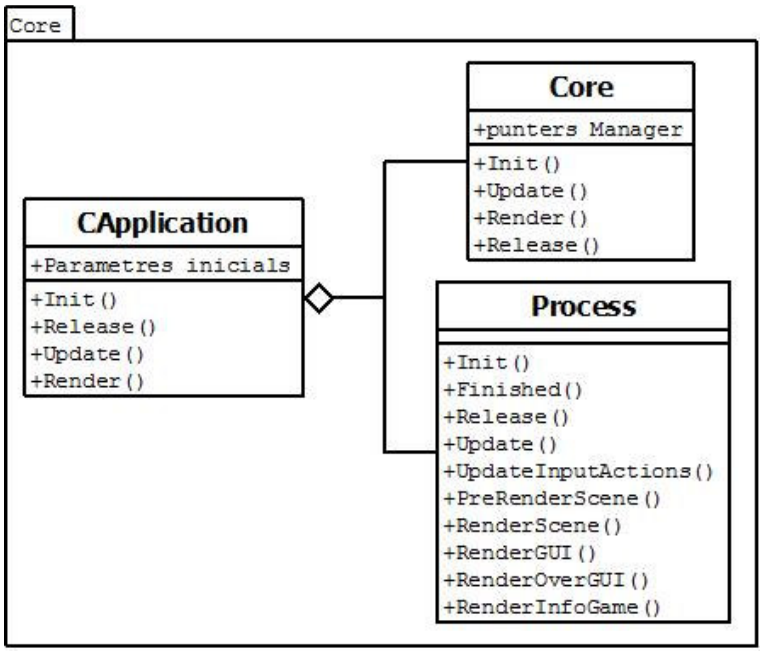
Aquest procés es realitza durant la fase de inicialització de la aplicació, per ser més exactes, en el Init() del Core. Aquest guarda la informació del Manager i es visible des de qualsevol punt de l'aplicació.

Tot això ens permet accedir a tota la informació des de qualsevol punt i en temps d'execució canviar-la si és necessari així com actualitzar-la en el seu moment. Aquest punt és de vital importància per poder incloure tota la funcionalitat de l'*scripting* en el projecte.

No obstant, això només és l'estructura bàsica d'un manager i, segons el tipus de dades a tractar, pot variar lleugerament. N'hi ha que no tenen *assets* i també varia el punt on es crida l'actualització del manager o, simplement, no té crida.

4.5. Estructura del motor i del videojoc

Ambdues coses estan lligades, el motor per sí sol no fa res i el videojoc necessita el motor per funcionar. No es pot entendre l'un sense l'altre però el videojoc és un mòdul separat del motor que pot ser canviat sense necessitat de tocar res del motor.



Il·lustració 2: Diagrama del cor del motor.

El funcionament bàsic es compon en un bucle que es repeteix fins al finalitzar l'execució que executa dos passos bàsic, l'*Update* i el *Render*. Per anar bé, s'haurien d'executar un mínim de 60 vegades per segon cada un per mantenir un flux bo d'animació i d'imatge així com un correcte funcionament i coordinació dels diferents components del joc. Tot i que hi ha molts joc punters que funcionen amb 30 imatges per segon.

4.5.1. Udpate

En aquest pas es porta a terme tota la lògica del joc a més d'actualitzar els estats dels diferents components. Dins seu podem destacar tres grans apartats:

- **Actualització de components**: alguns components s'actualitzaran independentment del que passi en el joc. Un clar exemple és la física on si alguna cosa cau, continuarà caient. També ho trobem casos similars en tot el que controla el pas del temps, per exemple els sons que durant un temps limitat o textos que apareixen durant una estona. En aquest apartat es tracten coses molt senzilles.
- **Lògica del joc**: segons el que estigui succeint s'activaran o desactivaran diferents parts del programa. Si s'ha activat una acció, aquí és on aquesta dirà el que s'ha de fer, des de moure alguna objecte fins a activar diferents sons passant per l'actualització de variables de control. Si algú ha canviat d'estat, aquí es on s'ha de comprovar. És l'apartat més complexa ja que és el que controla tots els canvis del videojoc.
- **Lectura de l'entrada de la interfície humana**: en aquest punt llegim el teclat i ratolí especificant quina acció s'activa i s'aplicarà a la lògica del joc.

Destacar que l'ordre en que s'executin aquests tres apartats es irrellevant sempre i quan s'executin tots tres. Això és deu a que estan dins d'un cicle que es repeteix moltes vegades per segon.

En molts casos i a favor de la optimització, aquests apartats es mesclen per poder millorar el rendiment.

4.5.2. Render

En aquest pas es dibuixa per pantalla. Sempre es té en compte que veu la càmera per no dibuixar res que quedi fora de visió millorant així el rendiment. La posició de cada element es determina en l'*Update*.

- **Pre-render**: Algunes tècniques de dibuixat necessiten informació addicional com un mapa de profunditat. Aquí fem tots els càlculs necessaris per aquestes tècniques, una d'aquestes és el dibuixat d'ombres dinàmiques.
- **Beging render**: En aquest punt inicialitzem les matrius, establim color de fons i preparem el hardware per enviar-li la informació. Es fa després del pre-render ja que aquest utilitza una informació diferent. A partir d'aquest punt tots utilitzen les mateixes dades.

- **Render scene**: És on hi ha la principal carrega de renderizat. Aquí es dibuixen els diferents models 3D segons la seva posició establerta durant el pas de l'Update i se'ls hi aplica els efectes de cada un d'ells. També és on es modifiquen els models animats.
- **Render GUI**: Dibuixem la interfície gràfica de l'usuari, per norma general són tot gràfics 2D que representen botons, comptadors i altres elements com guarniments de la interfície.
- **Render over GUI**: s'utilitza per si s'ha de dibuixar algun model 3D sobre la interfície de l'usuari. Els casos més típics es un dibuix animat de la cara del personatge.
- **Render system info**: Per norma general, l'escriptura de text pla indicant alguns estats del joc o informació addicional que s'ha de visualitzar per sobre de tot. Un cas típic es veure quantes imatges per segon tenim o la posició del cursor o la del personatge.
- **Render log**: Hi ha un log amb tota la informació del que s'ha anat fent durant el joc, de vegades es necessari consultar-lo i aquest es dibuixa per sobre de tot. En aquest punt també va la consola on també es necessari veure-la perfectament per poder escriure amb comoditat.

A diferencia del pas *Update*, aquí l'ordre es summament important. El primer que es dibuixi quedarà tapat pel que vingui a continuació. Per exemple, si el *render scene* anés últim ho taparia tot.

També s'ha de procurar optimitzar el més possible ja que és en el punt on es tracten més volum de dades. Poden haver-hi desenes de milers de polígons en pantalla juntament amb les seves textures més tot els gràfics 2D i les fonts del text.

5. Desenvolupament

5.1. Introducció

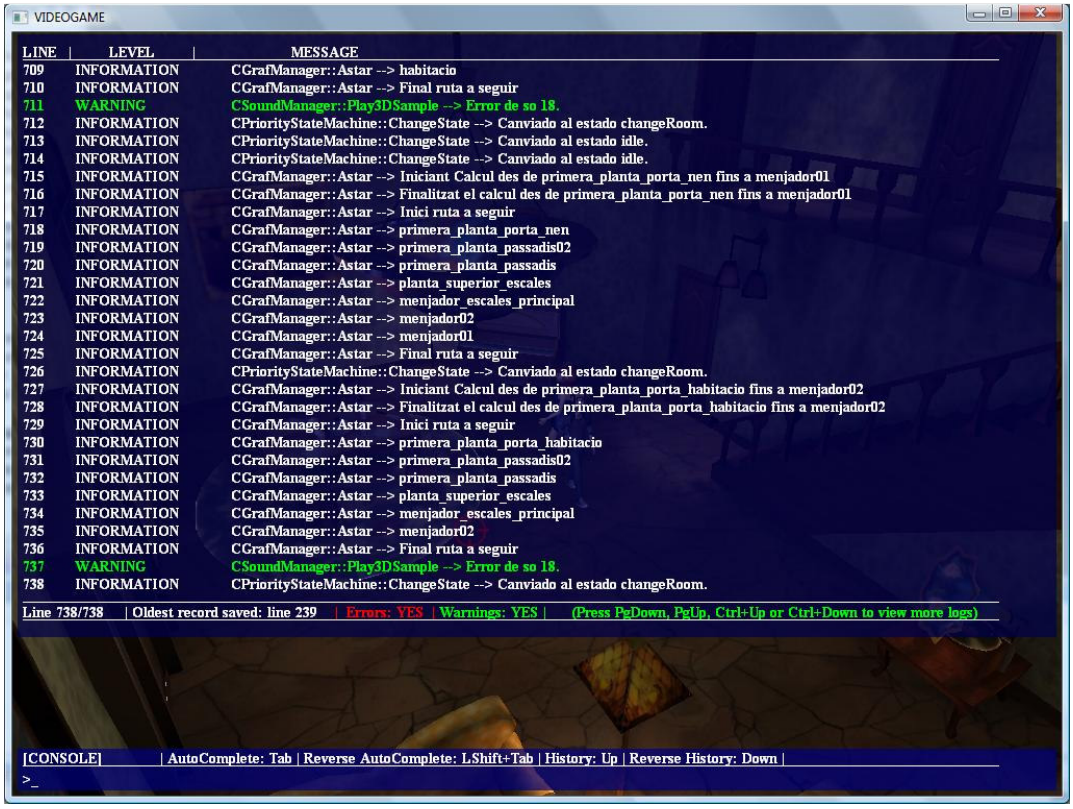
Aquí descrivim com s'ha portat a terme la implementació del projecte. Ens centrarem amb l'apartat de programació però també farem menció al procés productiu d'art.

5.2. Base del motor

En aquest punt es determina l'estructura del motor, quins components tindrà i com encapsularem les dades per poder convertir-lo en multiplataforma. Aquestes dades utilitzen *templates* per poder treballar. Un dels principals problemes amb les dades esta en el posicionament 3D ja que cada llibreria externa utilitza el seu propi sistema. Algunes son *right handed* amb la Y indicant altura mentre que altres son *left handed* amb la Z indicant l'altura o, simplement, canvien la Y per la Z. Per evitar problemes es crea un vector de tres posicions per determinar l'espai 3D. S'ha optat per una representació *right handed* amb la Y indicant l'altura tal i com utilitza el DirectX.

També s'implementen les eines que necessitarem pel desenvolupament del projecte que detallo a continuació:

- **MemLeak**: Com que treballem amb C++ no tenim *garbage collector* i ens veiem obligats a controlar tots els errors de memòria. Aquesta eina ens indica on s'ha produït la reserva de memòria que no ha estat alliberada. Així podem procedir a crear el seu destructor de forma adequada.
- **Logger**: Ens permet registrar el passos que segueix durant l'execució i registrar els errors que s'hi produeix. No obstant, no ho fa de manera automàtica i li hem d'anar indicant on ha de fer les entrades al log i que hi ha de posar. Tot i així, és d'una gran ajuda per controlar errors que van sorgint a mida que el projecte va creixent.
- **XML**: Aquesta eina la necessitem per llegir els fitxers XML amb els paràmetres i altres valors que necessitarem durant tot el videojoc. Aquí és on emprem la llibreria externa Libxml.



Logger i Consola desplegats.

A part d'aquestes eines més important també hi ha definit interfícies de patrons usats en altres punts del projecte com ara el Singleton o un manager de mapes.

5.3. Motor 3D bàsic, exportador i visualitzador

En aquest punt comencem a visualitzar alguna cosa per pantalla. El primer pas es dibuixar un triangle, després un quadrat i finalment una caixa, tot des de codi. Un cop aconseguit això ho anem complicant.

El següent pas es poder crear un objecte 3D amb una eina d'edició gràfica com el 3DSMax. Aquí és quan hem de crear un exportador per agafar una caixa simple amb una textura simple i que es vegi amb el nostre motor.

L'exportador es crea amb el MaxScript i ens crea un fitxer binari. Acte seguit, es copia aquest fitxer al directori de treball del projecte. Tanmateix, per agilitzar l'exportació s'ha de crear un plug-in de Max per la lectura del model 3D. Aquest ens permetrà baixà el temps d'exportació de casi deu minuts a deu segons, sense exagerà.

Un cop sabem que exportem hem de crear l'importador de malles estàtiques del nostre motor. Aquest es fa amb C++ creant una manager de malles estàtiques que conté objectes que són les malles estàtiques amb la informació llegida del fitxer binari. Després s'ha de crear un manager d'instàncies de malles estàtiques amb objectes que representen on es dibuixaran les malles estàtiques que hem llegit anteriorment. Fent-ho d'aquesta manera podem dibuixar el mateix model 3D en varies posicions sense necessitat de tenir-lo més d'una vegada a la memòria.

Un cop arribats aquest punt creem el visualitzador. Es tracta d'un programa que utilitza el nostre motor on es podran provar els models exportats amb l'exportador anteriorment creat.

Un cop esta llest, se'ls hi passa als artistes el visualitzador, l'script de l'exportador i el plug-in del 3DSMax perquè puguin provar els seus models en el motor que s'està creant. Per norma general, això comportarà molts i molts problemes ja que sempre hi haurà alguna limitació que no s'ha previst o l'artista ha fet alguna cosa massa complexa.

5.4. Motor 3D avançat, *shaders* i models animats

En aquesta tapa continuem treballant amb el motor gràfic. El primer que farem es treballar amb *shaders* i, per tant, passarem tota la feina de renderitzat al hardware. Cal destacar que s'ha treballat amb la versió 3.0 del HLSL i que, per tant, la targeta gràfica ha de ser compatible amb *shaders* 3.0.

El primer *shader* que creem és el més bàsic que ens permetrà dibuixar un objecte per pantalla sense tenir en compte la il·luminació. El resultat és un model 3D ben texturitzat i uniformement il·luminat.

El segon *shader* a crear és el que inclou il·luminació. Aquesta afecte durant el procés de *pixel shader* deixant el procés de *vertex shader* igual que l'anterior. Durant aquest projecte treballarem amb quatre tipus de llums creant la nostre il·luminació dinàmica:

- Direccional: emula la llum solar representant només un vector de direcció de la llum i no te en compte ni la distancia ni la posició d'aquesta. Això ens donarà una il·luminació uniforme per una banda mentre que l'altre quedarà obscura.
- Omni: una llum omnidireccional com podria ser un foc, una explosió o una bombeta que penja. Aquest tipus de llum ve marcada per dos distancies, la

major, que indica fins on il·lumina, i una de menor que indica on comença l'atenuació. Es va decidir emprà un atenuació lineal, no és la més realista però és la més senzilla de calcular i no hi ha gaire diferencia.

- Spot: es tracta d'una llum cònica amb dos radis, el major indica que il·lumina i el menor on comença l'atenuació. Existeix de posar també distancies d'atenuació com a la Ommni però no s'ha fet per estalviar costos de càlcul.
- Ambiental: és una llum uniforme que s'aplica a tots els píxels del model il·luminat, la seva principal funció consisteix en que no quedi res a la foscor absoluta deixant un mínim de llum residual evitant deixar cares negres.

S'ha de tenir en compte que les llums no generen ombres, aquestes es fan d'una manera diferent o, simplement, s'emulen fent una rodona a terra. També és important tenir en compte que les llums travessen les parets i que s'han de controlar.

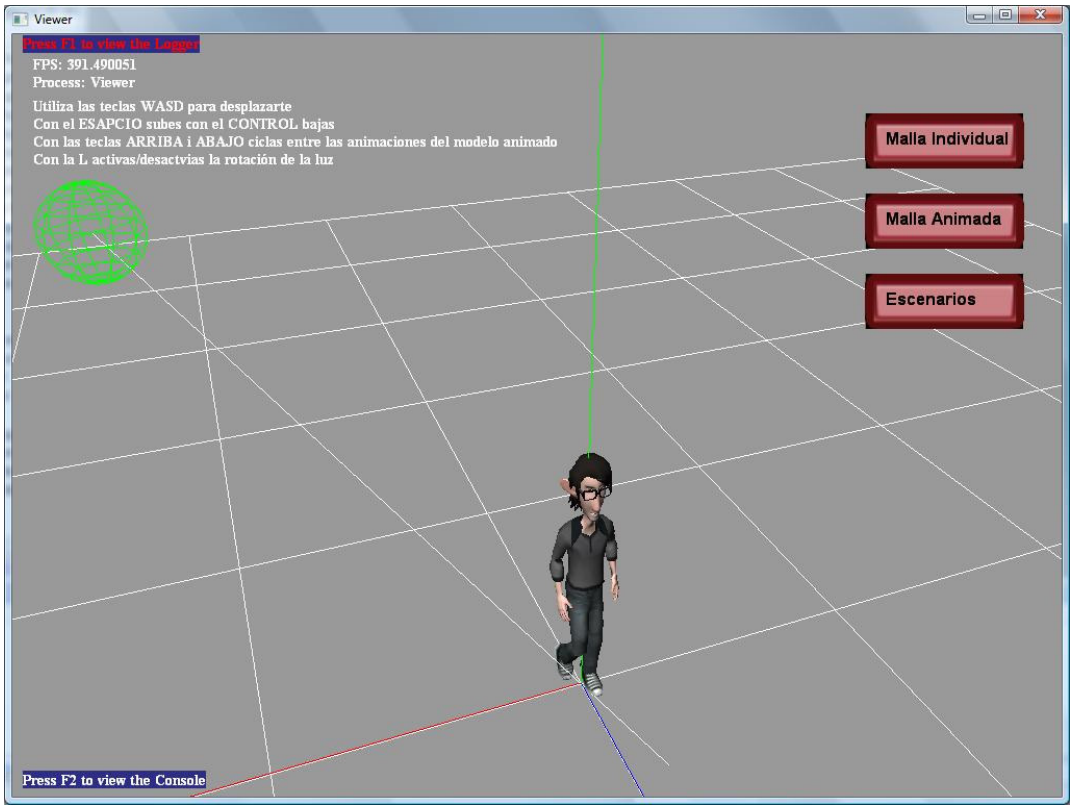
El següent pas es treballar amb la brillantor especular. És una tècnica simple que consisteix en cremar els píxels segons la posició de la càmera i de la llum creant un efecte de reflex en l'objecte.

Un cop tenim llums podem passar a tractar models 3D amb més d'una textura per aconseguir millors efectes. El primer que farem és la tècnica del *lightmapping* o mapa de llums. En aquest cas tenim dues textures, la de difús (la bàsica de l'objecte) i una altra que indica com està il·luminat el model. Aquesta tècnica te dues avantatges, la primera és que no cal calcular d'il·luminació per aquell objecte ja que ens ve donà pel *lighmap* i la segona és que s'aconsegueix una il·luminació més realista. Tot i que també té els seus defectes, el primer, de caire tècnic, és que estem carregant una segona textura a la memòria i el segon, de caire artístic, un objecte que tingui il·luminació dinàmica al costa d'un amb la il·luminació fixada es veurà alguna diferencia. Aquesta tècnica és molt útil per tots els objectes fixes com parets i terres estalviant-nos grans costos de computació gràfica.

La següent tècnica a implementar és el *bump mapping* o *normal mapping*. Consisteix en alterà les normals dels píxels que s'han de dibuixar donant un efecte de relleu sense necessitat de crear nous vèrtexs. Al tenir la normal alterada cada píxel visible actuarà de forma diferent segons la posició de la llum creant ombres a la superfície. És una tècnica ideal per quan es volen crear petits detalls als objectes com rugositats o sanefes en relleu. Però aquest truc es veu fàcilment quan la càmera s'acosta molt a l'objecte.

Una altra tècnica és la del *environmental map* o mapa ambiental que s'utilitza per crear objectes que reflexen el seu entorn. La segona textura és una imatge d'un cub on té imprès el que es veuria respecte la posició del model a dibuixà. Quan es treballa a nivell de pixel es combina el difús amb el resultat de calcular un vector de rebot des de la càmera fins al cub i agafant el píxel on s'impactà. Per norma general queda molt bé però aquest objecte ha de ser immòbil o la trampa es veurà amb molta facilitat. Útil per superfícies molt brillants com les boles d'un arbre de Nadal o la superfície d'un llac tot i que no relaxaria a qui estigues mirant-lo.

Arribats en aquest punt podem començar a combinar els efectes i triar quins volem i quins no. Un cop feta aquesta selecció es modifica l'exportador i l'importador de malles estàtiques i s'actualitza el visualitzador perquè els artistes puguin provar els models amb les noves textures.



Versió actual del visualitzador, amb el David carregat amb l'animació de caminar activada i una llum del tipus *spot* que parteix de l'esfera verda.

El següent pas es treballar amb els models animats. S'opta per l'animació per ossos usant el Cal3D. Aquesta llibreria externa incorpora el seu exportador i l'importador però s'ha d'adaptar al motor gràfic i crear l'encapsulament adequat.

L'animació per os consisteix en una sèrie de matrius que representen els punts de referència de la deformació del model on cada vèrtex té el seu pes sobre cada os.

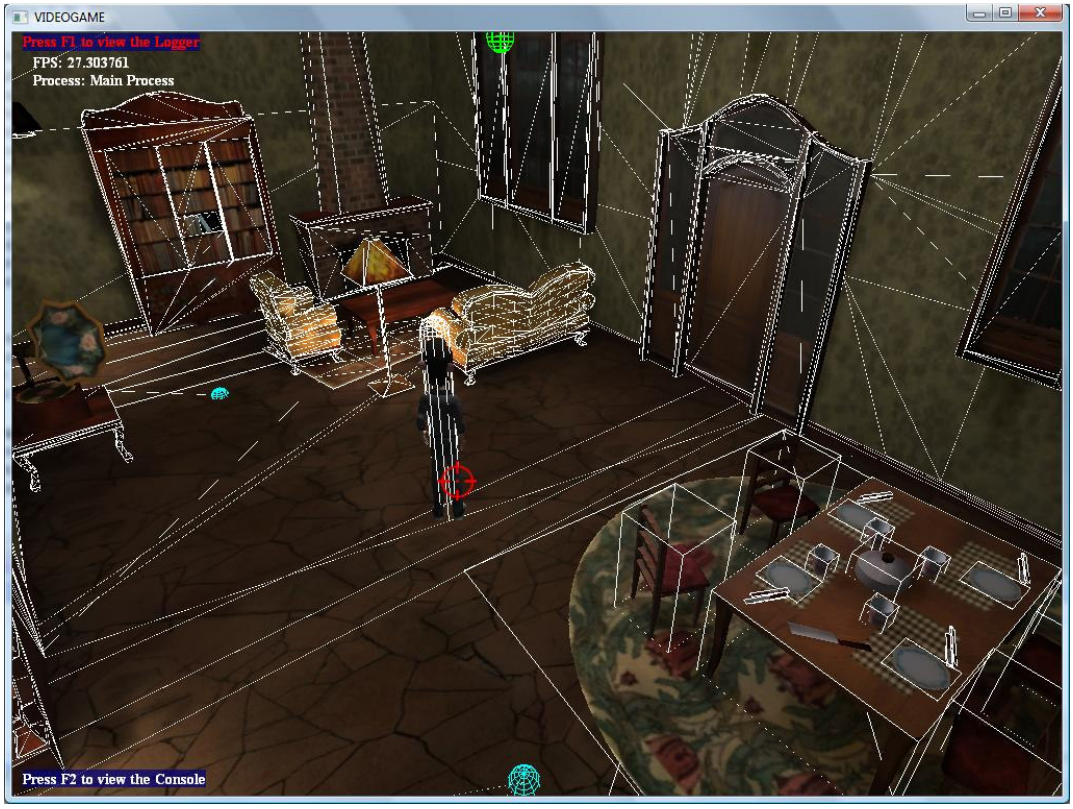
Són un conjunt de càlculs molt costosos per la CPU i per això s'optà per la tècnica de l'*skinning* que consisteix en enviar-li la informació a la GPU i que ella calculi les deformacions pertinents. La targeta gràfica esta molt més preparada per treballar amb matrius i l'augment de rendiment es força notable, com unes cinc vegades més ràpid.

5.5. Motor físic

La principal funció del motor físic és saber què toca a què. També es fan altres funcions però el càlcul de col·lisions és summament important ja que en permet establir noves accions i canvis d'estat durant el joc. Per crear el nostre motor físic es va optar per el *PhysX* d'Nvidia que té múltiples funcions integrades.

La feina en aquest punt és, essencialment, fer un bon encapsulament de totes les funcions que el *PhysX* incorpora, sinó de totes, les més bàsiques i les que utilitzarem. Entre aquestes encapsulacions hi ha la creació del manager de física que és qui determina que li passa a cada objecte.

Les representacions físiques dels models 3D no són exactes sinó que és realitzant aproximacions en forma d'esfera, cub, cilindre o capsula. Per norma general els objectes són cubs ja sigui una taula, un llit o un pollastre de goma, tot té una caixa contenidora. No obstant, altres objectes s'adapten millor a les altres formes, per exemple una pilota, que es una esfera, o una roda, que es un cilindre. Les capsules són normalment usades per personatges amb algunes característiques especials per ells, per exemple, la capsula sempre esta dreta, no pot tombar-se. També existeix la *cooking mesh* que aquesta s'adapta a la perfecció al model 3D que es vulgui tractar però té moltes limitacions, es aconsellable usar-la només per l'escenari fixa.



El menjador amb la seva representació física, destacar la capsula del David, les caixes de la taula i la *cooking mesh* de les parets i mobles fixos.

Per controlar les col·lisions els objectes es separen en grup i després es comprova si s'han tocat o no i s'aplica la física necessària. Per exemple, si llencem una pilota a una caixa, la pilota rebotarà i la caixa es mourà sense necessitat que nosaltres li diguem que ho ha de fer. Hi ha dues excepcions, els controladors de personatges o *character controlers* que només és mouen si se'ls hi especifica i les *cooking mesh* que només poden ser impactades.

5.6. Motor de lògica i scripting

La lògica i l'*scripting* van molt lligats. En aquest punt incorporem en el nostre motor el llenguatge Lua i la seva variant per treballar amb classes, el *Luabind*. Una part de la lògica del joc va implementada a l'*update* del videojoc, especialment les accions associades a l'entrada de teclat o ratolí mentre que l'altra s'executa a través d'instruccions Lua.

Un cop fet això podem començar a registrar funcions per poder ser cridades externament pel Lua. Això en serà de gran ajuda per configurar el comportament del joc.

5.7. Motor d'IA

Per poder tenir IA mínimament creïble s'han creat dos tipus de mecàniques d'estats. Una estàndard amb els seus canvis d'estats segons l'entrada que rep i l'estat en que es troba, i una altra basada en prioritats on segons l'entrada es considera un nou estat segons la seva prioritat i interromp l'actual.

El primer es usat per tots amb el que l'usuari pot interactuar i ens permet, per exemple, saber quan una taula esta volant o caient. El segon l'utilitzen les víctimes on segons el tipus d'ensurt que rebin canvien d'estat o segueixen iguals.

5.8. Motor d'àudio

Per l'àudio s'opta per la llibreria externa BASS. Aquesta ens permet controlar quins sons estan carregats en memòria i quins estan sonant. També suporta els so 3D i els cicles en els sons.

Els formats d'àudio que utilitzem son el WAV per sons normals i MP3 per la música. La utilització de WAV és deguda a la millor resposta en el moment d'activació del so aconseguint millor coordinació entre el que es veu i el que s'escolta. Amb MP3 tenim un petit desfaçament però si la música comença un segon més tard tampoc es problema, a més, MP3 aconsegueix millor compressió del so ocupant menys en memòria.

5.9. Motor d'idioma

Aquest mòdul s'encarrega de posar el textos i també de controlar els diàlegs que hi ha durant el joc.

Per una banda tenim el manager d'idioma que carrega totes les línees de text, ja siguin de botons, dels diàlegs o simples onomatopeies. Aquesta informació s'extreu d'un fitxer XML on també inclou informació sobre la velocitat de lectura, fonts i estils de visualització i també variables pel text.

Emma parlant durant la introducció.

Per l'altra banda tenim el control de diàlegs. El mànager s'encarrega de passar les línees d'una en una segons la velocitat de lectura o premen una tecla predeterminada. Aquest mànager també executa funcions Lua per controlar els canvis de càmera, les veus que s'escolten, sons de fons o canvis de posició dels models.

5.10. Implementació del videojoc

Arribats en aquest punt s'implementa el videojoc en sí. Li direm al motor quins arxius XML ha de llegir per carregar els models 3D i on els ha de col·locar, quins arxius sonors s'utilitzaran, quines tecles s'empren i a quines accions estan associades.

També programen la IA de les víctimes i les màquines d'estats del objectes i es van creant les diferents mecàniques d'estat pels models 3D amb els quals es pot interactuar.

David espantant-se al veure levita la taula.

La forma de desenvolupar el videojoc es bastant cíclica. Primer s'implementa una funcionalitat i, a través d'alguns test i diferents proves, es va adaptant a les funcions que hi ha fins arribar a un punt que esta correctament integrat i es passa a la següent funcionalitat.

6. Proves

6.1. Introducció

Les proves no només s'han realitzat al finalitzar el desenvolupament sinó que també s'han hagut de fer durant el seu desenvolupament per assegurar-se que segueix pel camí correcte i corregir qualsevol petit error que a la llarga es pugui convertir en un problema important.

Per poder dur a terme això es creen dos tipus de proves. Les de test, per provar el motor i altres funcionalitats tècniques. I les de joc, per provar el conjunt de funcionalitats i si s'ha aconseguit un producte divertit.

6.2. Entorn de proves

Per realitzar totes les proves aquí esmentades s'han emprat els mateixos equips de desenvolupament. També s'ha provat el projecte amb l'equip personal de cada un dels membres del grup el que no ha suposat cap problema ni de compatibilitat ni de rendiment.

6.3. Proves tipus test

Aquestes proves es realitzaven per provar noves funcionalitats del motor sent exclusives d'aquest. Això significa que des de l'apartat 5.3 fins al 5.9 han tingut el seu test o més d'un per assegurar el bon funcionament del motor.

Aquests test ens permetien accedir directament a diferents funcionalitats del motor prescindint de molts altres aspectes, inclús les representacions gràfiques ja que basant-nos en la informació del *logger* obteníem la informació necessària. També ens permetia no embrutar el codi del videojoc amb proves i mantenir-lo el més net i ordenat possible.

6.4. Proves de joc

En aquest punt cobren importància les proves sobre la mecànica de joc i la integració de les diferents funcionalitats del motor. Els tests fets amb anterioritat ens asseguren que el motor funciona correctament el que ens permet centrar-nos en comprovar les noves implementacions.

Aquests proves del videojoc es poden separar en dos grans grups:

- Integració d'art i física: Degut a que van per separat s'ha d'assegurar que els models 3D es dibuixin en el mateix punt on hi ha l'objecte físic. Molts cops sortien desplaçats, error normalment degut a la posició del pivot del model 3D ja que l'objecte físic sempre el té en el mig.
- Mecànica de joc: Ens assegurem que les accions i esdeveniments que succeeixen durant el joc funcionen segons l'especificat a la mecànica de joc. També corregim aquesta segons el resultats observats durant les proves per assegurar-nos el ritme de joc. La introducció també forma part d'aquest tipus de prova el que també inclou el so.

Aquestes proves són les que ocupen més temps del projecte perquè s'ha d'assegurar el correcte funcionament del videojoc en tots els seus aspecte, tant tècnics com a nivell d'art i jugabilitat.

6.5. Proves de les especificacions

Per poder provar que els objectius marcats en les especificacions que anteriorment s'han comentat en els punts 4.2 i 4.3 s'han usat les diferents proves anteriorment esmentades, tant els tests com les proves de joc.

6.5.1. Funcionals

Aquestes especificacions són l'objectiu primari pel qual el grup va decidir fer aquest projecte. La seva realització sempre ha estat en ment de tots el membres del grup.

- Capacitat per entretenir a l'usuari.

Per poder comprovar aquest punt no només n'hi ha prou amb les proves de joc. S'ha deixar que algú extern a l'equip de desenvolupament provi el videojoc i preguntar-li la seva opinió. Els comentaris que ens han dit des de la més pura sinceritat ens fan creure que es va pel bon camí.

- Demostració de la capacitat dels integrants del grup de crear un videojoc.

Semblant al punt anterior però des d'un punt més tècnic. Al deixar-lo provar a diferents persones externes al grup també han fet comentaris a favor d'aquest punt. També tenim la informació tècnica que podem extreure del logger i del motor es pot assegurar que tot funciona correctament i té un aspecte d'acabat.

- Poder explicar una idea de joc de forma ràpida i concisa.

S'ha comprovat que qualsevol jugador ha copsat ràpidament la idea bàsica de joc poden interactuar ràpidament amb tots els elements de l'escenari. També s'ha comprovat que la durada de la demostració ha estat l'adequada per ser un producte ràpid de veure.

6.5.2. No funcionals

Tot i que es podria dir que aquestes especificacions no tenen tanta importància com les funcionals no hi hauria res més lluny de la realitat. El valor afegit que se li pot atorgar amb aquestes especificacions són el que el faran diferencia de la resta de productes similars fent-lo destacar per sobre de la competència. Encara que és cert que si no es portessin a terme no canviarien la funcionalitat del videojoc resultant.

- Qualitat gràfica adaptada als nivells exigits actualment.

Per poder portar a terme aquest punt s'ha optat per intentar evitar el realisme i adoptar una estètica més de dibuix animat permeten un nivell inferior de detall tant en el modelatge 3D com a la texturització d'aquest. Això disminueix la carrega en el motor gràfic permetent un desenvolupament menys optimitzat pel videojoc resultant deixant-lo més portable.

En general, el resultat gràfic resultant ha estat millor de l'esperat deixant-nos marge per millorar-lo de cara al futur.

- Qualitat sonora que empri els estàndards de so actuals en 3D.

Gracies a la implementació de la llibreria externa BASS, s'ha pogut recrear tots els sons necessaris tal i com el volíem. No obstant, queden detalls a polir com igualar el volum dels efectes sonors i les veus que son difícils de compensar. D'altra banda també tenim que les veus no han estat gravades ni per actors ni en un estudi de so notant-se en la qualitat d'aquestes. De totes formes es pot dir que a nivell tècnic s'ha complert aquest punt.

- Poder ser exportat a diferents plataformes.

Tot i que no s'ha realitzat aquesta especificació s'ha dissenyat tot el motor de joc i el videojoc tenint en compte tota l'encapsulació necessari així com les capes a implementar. Actualment seria una tasca acceptable fer un canvi de plataforma amb el codi actual.

- Tenir la capacitat de localitzar-se a diferents idiomes.

El fitxer XML d'idiomes que s'empra en l'execució del videojoc es pot canviar amb molta facilitat perquè n'utilitzi un altre sempre i quan aquest estigui correctament dissenyat. A nivell de veus s'haurien de tornar a gravar i canviar també el fitxer XML que s'encarrega d'aquestes. Podem assegurar que el canvi d'idioma es pot fer amb facilitat.

- Fer-lo el més accessible possible tenint en compte possibles minusvalideses dels usuaris.

Actualment el videojoc està completament subtitulat i amb onomatopeies pels sons. Els subtítols disposen de contrast respecte la seva caixa de fons i amb lletres suficientment grans per ser llegides amb facilitat. Tot això el fa accessible a la gent amb problemes d'oïda.

També es pot jugar amb una sola mà, per ser exactes, amb el ratolí. Això permet jugar-hi a la gent que li falti una mà.

7. Conclusions

7.1. Introducció

Aquí es detallen les conclusions que es poden extreure de la feina realitzada durant el transcurs del projecte. Primer comentarem les desviacions que s'observen al comparar la idea inicial amb la final. També es descriu la planificació futura per aquest projecte i, finalment, quines conclusions finals s'obtenen.

7.2. Desviacions

Realment hi ha hagut moltes desviacions respecte l'idea original. No obstant, això era d'esperar i estava previst i no ha afectat als temps assignats. És habitual, per no dir que és la norma, que el resultat final no sigui igual que el que es tenia previst des del principi però el resultat final sempre manté l'essència de l'original. En el nostre cas, fer una casa encantada controlada pel jugador que espanti al seus habitants.

Els temps assignats a cada tasca són orientatius tot i que han estat bastant exactes. Tota la funcionalitat del motor ha estat acabada a la data prevista. No obstant això, de vegades s'ha necessitat implementar alguna funcionalitat nova però el temps dedicat per fer aquesta millora esta previst en el desenvolupament del videojoc.

Entre les principals desviacions hi ha la referent al disseny on cada cop que veiem que el grup avançava en dos direccions tocava revisar el disseny del joc. Això va suposar un augment de temps en el referent a tot el disseny.

7.3. Millores pel futur

Un cop algú ens va dir "els projectes no s'acaben, s'abandonen" i és totalment cert quan es parla de videojocs. Sempre hi ha alguna cosa més a afegir, alguna cosa a polir, això no vol dir que no es tingui un producte final, simplement vol dir que tot es pot millorar. Realment, el que s'acaba és el temps i els diners però les idees tendeixen a reproduir-se i no acabar-se mai.

El grup seguirà treballant amb el projecte durant l'estiu tal i com ho ha fet fins ara per ampliar i millorar el producte final. No hem d'oblidar que el projecte és una carta de presentació per a qui l'ha fet i sempre es vol quedar bé. D'altra banda,

també és pot veure com un fill pel qual desitges que millori i veure'l fer créixer i fer-se gran.

Les tasques que ens em proposat per aquest estiu són:

- Millorar la *GUI* fent més fàcil i entenedor l'ús del videojoc.
- Implementar un soterrani amb llums i ombres dinàmiques.
- Millorar i incorpora nous sons.
- Permetre a l'usuari canviar la visualització dels subtítols per adaptar-se millor a les seves necessitats.

Tot i així sempre es podria continuar el projecte per incloure nivells addicionals o puntuacions entre altres temes. Per no parlar de l'exportació a altres plataformes diferents al Windows de PC.

7.4. Conclusions finals

Des d'un bon principi s'havia plantejat un videojoc d'una casa encantada controlada pel jugador on l'objectiu fos fer fora els seus ocupants a base d'ensurts donats per poltergeist, fantasmes i similars. Aquest punt sempre s'ha tingut en compte i, gracies això, s'ha pogut dur a terme. No obstant, la casa i les víctimes han anat variant durant el desenvolupament del projecte.

Una altra part important ha estat el treball amb equip. No només era una qüestió de distribuir temps i tasques entre els membres, també ens havíem de posar d'acord sobre la visió que teníem del producte final. Al tractar-se d'un projecte creatiu portat a terme per un grup amb diferents perfils professionals, aquest punt és summament important si es vol que tot arribi a bon port. Cadascú aporta la seva visió del conjunt i es van prenent les decisions que donen forma al producte final.

A nivell personal ha estat i és un projecte que em desperta gran interès del qual puc dir que he aprés molt. No només a nivell de coneixements sinó també sobre treball en equip. I no només en un equip homogeni, aquest grup estava format per gent de diversos perfils on cadascú té la seva visió del projecte. Només falta repetir-me un cop més per dir que sempre m'han agradat el videojocs i poder-ne crear un d'aquesta manera ha estat una gran experiència.

8. Bibliografia

La bibliografia emprada es remet a les ajudes que porten les diferents llibreries externes així com a pàgines web amb diferent informació:

- DirectX 9.0: fitxers d'ajuda, exemples i tutorials del SDK.
- PhysX: fitxers d'ajuda, exemples i tutorials del SDK.
- Cal3D: fitxers d'ajuda, exemples i tutorials incorporats a la llibreria.
- BASS: fitxer d'ajuda incorporat a la llibreria.
- LUA: fitxers d'ajuda i exemples incorporats a la llibreria.

(Aquesta pàgina s'ha deixat intencionadament en blanc.)

9. Apèndix

9.1. Glossari

3DSMax: Abreviació per *3D Studio Max*. Aplicació per crear models 3D.

Animació esqueletal: Tècnica que usa ossos per crear animacions en models 3D.

API: sigles per *Application Programming Interface*. Una interfície de programa per comunicar el software amb el hardware.

Asset: Recurs en forma de fitxer que es necessita pels videojoc. Inclou els fitxers que contenen els models 3D, les textures i els sons, entre altres.

Bass: Llibreria externa que s'encarrega del so.

C: Lleguatge de programació de propòsit general estructurat que permet crear aplicacions de forma òptima i controlar tots els recursos.

C++: Llenguatge de programació de propòsit general orientat a objecte que permet crear aplicacions de forma òptima i controlar tots els recursos.

Cal3D: Llibreria externa encarregada de l'animació esqueletal.

CPU: Sigles per *Central Processing Unit*. Component hardware encarregat de la majoria de tasques.

DirectX: Conjunt d'APIs de Microsoft destinades a facilitar totes les tasques múltimedia, especialment en videojocs.

GPU: Sigles per *Graphics Processing Unit*. Component de hardware encarregat del càlcul de visualització, especialment en 3D.

HLSL: sigles per *High Level Shader Language*. Llenguatge per *shader* per funcionar amb DirectX.

Lua: Llenguatge de programació estructurat tipus *script* amb semàntica ampliable.

LuaBind: Llibreria que converteix el Lua en un llenguatge orientat a objecte.

MaxScript: Llenguatge de programació tipus *script* que s'usa en el 3D Studio Max.

Mecànica de joc: Conjunt de normes que especifiquen com es un videojoc.

Model 3D: Conjunt de polígons i textures que representen un objecte tridimensional.

Os: Punts de referència d'un model 3D que permeten deformar-lo de forma simple aconseguint un efecte d'animació.

PhysX: Llibreria externa que s'encarrega del procés de la física.

Plug-in: Petit programa que s'utilitza en un altre que modifica el seu comportament.

Píxel: Punt mínim d'una pantalla o d'una imatge o textura.

Polígon: Un dels triangles que forma el model 3D

PPU: Sigles per *Physic Processing Unit*. Component hardware encarregat del càlcul de la física. Actualment moltes GPUs el porten integrat degut a la seva similitud.

Script: Conjunt d'instruccions que s'executen en calent durant el procés del programa. També permet canviar-les i no es necessària la seva compilació

Shader, pixel: Conjunt d'instruccions que actuen sobre el píxel del model 3D que es dibuixarà.

Shader, vertex: Conjunt d'instruccions que actuen sobre el vèrtex del model 3D que es dibuixarà.

Shader: Conjunt d'instruccions dirigides al hardware gràfic que ens permet operar amb el resultat a visualitzar just abans de la sortida per pantalla.

Textura: Imatge que s'aplica sobre el model 3D per dibuixar-lo.

Vèrtex: Punt que representa la unió de dos o més arestes que formen els diferents triangles d'un model 3D.

9.2. Document de disseny inicial

A continuació s'adjunta una copia del primer document de disseny tal i com es va consensuar i entregar. Destacar que actualment s'han canviat moltes coses i que només s'inclou de manera orientativa.

Casa encantada

1. **Game concept (descripción rápida y sintetizada al máximo de los elementos distintivos y definitorios del juego)**

   Juego de estrategia ambientado en una vivienda encantada cuyo propósito es expulsar a sus habitantes. El jugador lleva la casa.

2. **Descripción ampliada (añade información más detallada: público objetivo, estética, posicionamiento en el mercado, etc.)**

   Se trata de una parodia de toda el género de terror incluyendo el *survival horror*. Tenemos que manipular los objetos que hay en la casa para asustar a sus habitantes, esto va desde abrir y cerrar las luces hasta sacar fantasmas de la tele, pasando por la cubertería flotante y los ruidos variados de golpes y fantasmas.

   Se trata de un producto pensado para entretener a un público adulto aunque joven de todos los sexos.

   Se trataría de un juego tipo *causal* ambientado en la actualidad. Es importante destacar que quien tiene que tener miedo es la víctima, no el jugador.

3. **Sistema de juego:**

   3.1. Punto de vista

   El juego se ve en una perspectiva isométrica pudiendo ver toda la habitación donde este la víctima. Similar a la vista de el videojuego Sims.

   3.2. Acciones

   -Controlar la tv

   -Ejecutar Sonidos y voces fantasmagóricas

   -Abrir/cerrar puertas

   -Agua marrón por la ducha, water y grifos

   -Sangre por la ducha

   -Hacer volar cuchillos y tenedores

   -Mover muebles y electrodomésticos: p.e. mover una silla cuando alguien vaya a sentarse

-Mobiliario que se transforma en monstruo

-Manchas en la pared con forma de virgen

-Cuadro que observa por los ojos

-Puertas secretas a otras dimensiones

-Aparición de fantasmas.

3.3. Control

El control se llevaría a cabo con el ratón. Hay diferentes puntos calientes que al pasar por encima se puede desplegar con un click izquierdo. Por ejemplo: un tenedor al darle podemos hacerlo volar o perseguir al habitante de la casa. Con el botón derecho en cualquier punto de la casa se pueden hacer dos tipos de acciones: ruidos de golpes donde este el curso (pared, puerta...) o voces escalofriantes.

3.4. Gameplay

3.4.1.Jerarquía principal

Objetivo final del jugador en el juego: Expulsar a los habitantes.

Cómo expulsar los habitantes de un piso: Combinando las acciones disponibles se asustarán más o menos. Mediante un "asustómetro", parecido a un termómetro, podremos calcular la cantidad de susto que hemos conseguido en nuestras víctimas. Deberemos llegar al 100% y mantenernos ahí durante al menos 1 minuto, si alcanzamos el 120% el personaje morirá y será fin de juego. No debemos llegar a esa cifra, la casa sólo quiere que se marchen no matarlos. Ya que la llegada de la policía retrasaría los sucesos. También podemos morir si no hacemos nada para que ellos se asusten, ya que la barra de energía de la casa ira disminuyendo si ellos ganan terreno contra nosotros, por ejemplo no se asusten o hagan algo que no nos guste.

3.4.2.Desarrollo general

El desarrollo es lineal. El jugador podrá explorar la habitación donde se encuentre la víctima encontrando los puntos calientes y experimentar con ellos para comprobar que efecto tiene sobre la víctima.

3.4.3.Conversaciones

Sus habitantes hablan entre ellos y si el jugador los escucha sabrá cuáles son sus miedos.

3.4.4.Niveles

Hay un solo nivel formado por varias habitaciones de una casa. En cada nivel las víctimas tienen diferentes formas para recuperarse del miedo o para que no les afecte. Aunque en cada vivienda haya casi los mismos puntos calientes y objetos la forma en que tiene que usarlos variará en cada caso.

4. Narración

La vivienda es una casa con dos inquilinos. La vivienda quiere que se vayan todos para que la compre un especulador inmobiliario, construya un bloque de once pisos con cuatro viviendas por piso y así ser más respetada en la difícil sociedad de las casas encantadas.

Cada víctima también tiene su historia que sirve para poder asustarlos mejor saber que no les asustará. La pareja de protagonistas está en crisis y han decidido separarse. Discuten para ver quien se queda la casa porque los dos la quieren.

En este momento empieza el juego, con los protagonistas en casa haciendo sus tareas. La información sobre la voluntad de la casa de echar los inquilinos y la situación de la pareja ya se ha dado al jugador (en formato de texto o de animación).

El jugador debe asustar inicialmente a uno de los dos protagonistas alternativamente. En un principio ellos piensan que es un montaje del otro para que tenga miedo y deje la casa al otro. Los sustos desencadenan conversaciones entre los personajes de los que sabremos cuales son sus puntos débiles. Esto dará pistas al jugador para encontrar el siguiente puzzle y preparar otro susto. Los personajes se dan cuenta que no es un montaje del otro y cada vez se asustan mas. Las sorpresas van desde pequeñas cosas al principio, cada vez de mas envergadura hasta las apariciones fantasmales, etc.

Finalmente se irán de la casa (opcional: se pueden reconciliar).

5. Diseño de personajes

Pareja esta formada por un chico y una chica. Opcionalmente una mascota.

Chica: 28 años, rubia 1,72m 70kg. Trabaja como estilista para bandas de rock de la ciudad. No gana casi dinero por lo que la mantienen sus padres y su novio. Le gusta aparentar e ir "cool", tiene el pelo corto con flequillo, teñido de rubio platino, lleva unas grandes gafas de vieja, unas mallas negras elásticas brillantes, una camiseta de Mickey Mouse y unas bambas tobilleras Nike de color verde fosforescente y lila. Su cantante favorita es Bjork y la Casa Azul, también le encanta el cine independiente, al estilo Lars Von Trier.

Chico: 32 años, moreno 1,80 78kg modelo de pasarela. Es el típico guaperas, cuando era joven volvía locas a sus compañeras de clase. Con una mirada hace que las chicas se enamoren. Ojos azules, piel morena, rasgos andróginos. Le gusta lo mismo que a su novia: jugar al singstar, salir de fiesta con la cream de la cream de la moda barcelonesa y mirar el Facebook en su mack book pro...

Son modernos, trendy, repelentes y superficiales.

6. Arte

La estética del juego no es realista y tiene toques de cómic. También será referente la estética de las pelis de terror de serie B. La decoración de la casa tiene que representar la personalidad y gustos de sus habitantes.

7. Tecnología

Los requerimientos técnicos se centran el poder mostrar una habitación con todo su mobiliario y poder manipular este. Será necesario un motor de física para poder mover los objetos por la habitación así como hacer que la IA de la víctima se dé cuenta y reaccione a lo que sucede.

8. Referencias

Hay muchos juegos donde se está dentro de una casa encantada pero pocos que sea al revés, de las referencias que ahora proponemos sólo The Haunting (Megadrive) se acerca más a la idea que proponemos.

Videojuegos

The Sims (2000.PC,PS2,Xbox,GC): es un juego de estrategia simulador de vida publicado por EA y tiene numerosas secuelas. Recrea la vida de diaria de unas "personas" , te permite construir tu propia casa y ciudad.

http://www.youtube.com/watch?v=ib6gM1NtOuQ

http://www.youtube.com/watch?v=6UPHjy9Kv2A The Sims Haunting

-Luigi's Mansion (2001, Gamecube) : acción/aventura publicado en 2001 Nintendo EAD

Manejas a Luigi en busca de Mario por una enorme mansión atestada de fantasmas. Tu única ayuda son los consejos del profesor E. Gadd y de una genial arma: la Poltergust 3000 para succionar los fantasmas (además de una linterna)

http://www.youtube.com/watch?v=eU-wgVz9kUg

-The Haunting (1993, Megadrive)

Controlamos a un rebelde juvenil muerto (Polterguy) en una casa ocupada por una familía yuppie, los Sardinis. La tarea del jugador es asustar a los miembros de la familía para conseguir que marchen de la casa mediante la posesión de los objetos y muebles (parodia de Poltergeist) . Para conseguir el game over la barra de energía del fantasma tiene que llegar a zero. La vista es isométrica.

http://www.youtube.com/watch?v=Z5EVBk81UMU

-Haunted House (1982,Atari 2600)

El jugador es un par de ojos que se desplazan por una mansión encantada para recuperar las piezas de una urna funeraria. Es una juego de acción aventuro y uno de los primeros "horror survival" por así decirlo. Hay que escapar de arañas, murciélagos y fantasmas.

http://www.youtube.com/watch?v=nQGSuwZMWIQ

-The Ultimate Haunted House (1994, PC)

El jugador se encuentra en una casa encantada. Debe escapar antes de que transcurran 13 horas, por lo que tiene que explorar 13 habitaciones y encontrar las 13 llaves sino no podrá salir nunca. La mecánica consiste en interactuar con un grupo de fantasmas (con personalidad) y coleccionar objetos, docenas de actividades y puzzles que harán mover la historia. El humor es una de las características principales del juego, actividades como: laboratorio de monstruos, habitación de visionado de pelis de terror, componer música con un órgano …son muestra de ello

http://www.youtube.com/watch?v=buqKng1F7IA

Libros

-Charles Dickens Haunted House

http://www.bookrags.com/Charles_Dickens

Cine

The Amityville (Stuart Rosenberg 1979)

Beetlejuice (Tim Burton, 1988)

House on Haunted_Hill (William Castle, 1959)

Poltergeist (Tobe Hooper,1982)

House (Steve Miner, 1986)

9. Alcance de la demo

9.1. Escenario

Solo un piso. Parejita.

9.2. Personajes

Se modelará dos personajes: chico y chica. No obstante también se tendrán que modelar algún fantasma.

9.3. Animaciones

Víctima: idle, caminar, correr, asustarse.

Fantasmas: idle, levitar (moverse), asustar.

9.4. Acciones/control

Hacer ruidos, apagar y encender luces, mover objetos, aparición de fantasmas.

9.5. Tecnología

Motor 3D, cámara 3D, normalmapping, animación esqueletal, control, IA de la víctima, GUI.

9.6. Sistema de juego

Consistirá en asustar a la víctima hasta que se vaya por la puerta.