Índex

1. Motivacions....3
2. Objectius i Requeriments....5
3. Eines Utilitzades....7
3.1. Programes....7
3.1.1. Visual Studio 2010....8
3.1.2. Autodesk 3ds MAX....8
3.1.3. MySQL....9
3.1.3.1. MySQL Server....9
3.1.3.2. MySQL Workbench....9
3.2. Llibreries....10
3.2.1. ENet....11
3.2.2. OpenGL....12
3.2.3. SOCI....13
4. Estructura de l'aplicació....14
4.1. Visió General....14
4.2. Part client....14
4.2.1. Bucle Principal....15
4.2.2. Gestor d'Estats....16
4.2.3. Carregant dades....17
4.2.3.1. Models....18
4.2.3.2.Materials....19
4.2.3.3.Textures....20
4.2.4. Entrada....21
4.2.5. Xarxa....23
4.2.6. Actualitzant l'Escena....23
4.2.6.1. Input....24
4.2.6.2. Físiques....24

4.2.7. Dibuixant escena.....25
4.2.7.1. Renderitzant objectes.....26
4.2.7.2. Utilitzant Materials.....28
4.3. Part servidor.....29
4.3.1. Base de Dades.....29
4.3.2. Manejament de tots els clients.....31
4.3.3. Broadcasts.....31
4.4. Protocol.....33
4.4.1. Paquets Servidor-Client.....34
4.4.2. Paquets Client-Servidor.....36
5. Millores.....38
6. Bibliografia.....40

1. Motivacions

Com a moltes persones de la meva generació, des de ben petit m'han agradat molt els videojocs. Fa uns 7 anys vaig descobrir els jocs en línia, els quals em van resultar molt interessants. I encara més quan vaig descobrir el meu primer joc de rol en línia (MMORPG), Lineage 2. Aquests últims anys, mentre aprenia a programar, he anat descobrint que jo mateix podria fer alguna cosa.

MMORPG són les sigles de Massive Multiplayer Online Role Playing Game: Joc de Rol Multijugador Online Massiu. Els jugadors connecten a un servidor i poden interactuar entre ells en el món virtual que el joc ofereix. Hi pot haver milers d'usuaris connectats simultàniament al mateix món, generant una població i molt d'ambient si el servidor està ben programat, i latència i/o bloquejos si se supera el límit de jugadors simultanis que el servidor suporta.

El més atractiu d'aquest tipus de jocs és que dónen la capacitat de jugar amb i/o contra altra gent real. Resulta molt més emocionant veure intel·ligència real dins el joc, sense que tot sigui intel·ligència artificial. A més d'això hi ha el fet de que pràcticament tots aquests jocs són de l'estil Sandbox o sorral, cosa que significa que el jugador és totalment lliure de fer el que vulgui en un món obert, sense haver de seguir una trama lineal.

Durant el principi del segon curs vaig començar a investigar una manera de poder jugar a Lineage 2 des de la residència en la que estava aquí a Vic, ja que la seva connexió a Internet tenia un proxy que bloquejava tots els ports excepte HTTP i algunes poques excepcions més, així que no podia connectar al seu servidor per jugar. I aquí vaig topar amb L2JServer, un projecte open source que emula un servidor de Lineage 2. Només instal·lant una base de dades MySQL, el java JDK i extraient uns arxius a una carpeta, ja podies executar el teu propi servidor d'aquest joc. Per entrar-hi només calia enganyar el programa client redirigint la DNS oficial cap a la teva IP utilitzant el fitxer hosts de windows i ja hi podies entrar.

A partir d'aquí, fent proves, vaig acabar obrint un servidor al públic el qual va tenir molt d'èxit. Des d'aleshores he estat sempre dissenyant i programant idees noves pel servidor, per atraure més jugadors, i gràcies al codi de L2JServer he après moltíssim sobre el disseny i programació d'aplicacions client-servidor, sobretot com la part servidor gestiona totes les dades i la part de xarxa estructura i envia els paquets.

Aquesta experiència és molt satisfactòria, però molesta el fet d'haver partit d'una base tan avançada (el projecte L2J) i un joc ja fet (Lineage 2), a més del fet de no tenir cap mena d'accés al codi del client ni dret a modificar-lo ni distribuïr-lo. Així que em vaig proposar fer un MMORPG comercial des de zero.

Només em faltava aprendre a fer un client en 3D, o al menys un prototip, i a partir d'allí ja podria començar a dissenyar i programar el joc definitiu.

Fent aquest treball vull aprendre tots els coneixements bàsics que necessito per començar a fer un programa d'aquest tipus sense fer grans errors de disseny. Tinc la oportunitat d'equivocar-me i aprendre què hauria pogut fer de manera més clara i/o eficient, per aplicar-ho en el/s futur/s projecte/s.

2. Objectius i Requeriments

Aquesta aplicació consta de dues parts: el client i el servidor.

El client és una interfície gràfica que depèn totalment del servidor. Quan és exacutat ens demana un nom d'usuari i contrassenya per connectar al servidor. Si el servidor no està engegat o el client no té connexió o accés a aquest, no es podrà jugar.

Un cop connectat, tot el comportament dels elements que el client veu depenen totalment del servidor. El client és una simple terminal gràfica que fa peticions de permís (com per exemple per moure's) i el servidor controla tot el joc. Ell administra totes les dades, “ordena” a cada client connectat què és el que ha de mostrar i coneix les regles del joc.

Amb aquesta arquitectura, si un usuari decideix modificar el client per fer trampes i comença a enviar paquets que se saltarien les normes (no aplicable en aquest projecte ja que al ser només un prototip no hi ha cap disseny de joc ni normes), el servidor simplement els descartaria o encara més, si aconsegueix determinar sense cap dubte que un client modificat mai podria tenir tal comportament, el podria expulsar del joc, tancant la seva connexió i, si cal, impedint futures connexions des de la conta que ha fet trampes.

Per aconseguir aquesta arquitectura s'han de crear dos programes a part. L'aplicació servidor se la quedarà l'administrador del joc i l'executarà en un servidor dedicat, amb els ports necessaris oberts, i l'aplicació client es distribuïrà als potencials jugadors perquè connectin amb el servidor utilitzant aquest per comunicarse i poder jugar entre ells.

Resumint:

- El client és la interfície gràfica de cada usuari i la forma de comunicació d'aquest amb el món virtual o servidor.
- El servidor és on tot el joc transcurreix i el canal de comunicació entre tots els programes clients.

El procediment que es durà a terme quan l'usuari vulgui executar certa acció, com per exemple caminar cap a un punt, serà:

- L'usuari dóna clic a l'opció/botó o escriu el comando (en aquest cas dóna clic a un punt del terra per caminar).
- El client envia una petició (request) amb la informació necessària, la qual en el cas de caminar són les coordenades a les que vol anar.
- El servidor comprova les coordenades i en cas de que siguin vàlides, fa un broadcast (ho envia a tots els jugadors pròxims) del moviment i canvi de posició d'aquest jugador.
- El jugador que es volia moure veu com el seu personatge es mou cap al punt on volia anar, és més, tots els demés jugadors pròxims ho veuen.

Aquest procediment, si hi ha latència de resposta (lag) en la connexió o el servidor està sobrecarregat, es veurà molt clar en el programa client: qualsevol acció trigarà una mica a tenir una resposta o reacció.

3. Eines Utilitzades

Per dur a terme un projecte de programació són necessàries com a mínim dues eines en l'ordinador: un editor de text per escriure el codi i un compilador del llenguatge en el què estiguis escribint aquest codi.

En el cas d'aquest projecte, a més d'utilitzar un programa que fa aquestes dues funcions (l'entorn Visual Studio) i demés programes externs, s'utilitzaràn llibreries externes que ofereixen funcionalitats concretes per ser utilitzades en l'aplicació.

3.1. Programes

Els programes (o eines) que s'utilitzen en un projecte informàtic, solen ser aplicacions dissenyades per assistir en la creació de cert contingut, siguin altres programes, dibuixos, documents, models tridimensionals o bases de dades.

3.1.1. Visual Studio 2010

Aquest és l'entorn de desenvolupament que s'ha triat per aquest projecte, el qual serà escrit en C++.

Amb Visual Studio es pot crear un programa des de zero només escribint codi alhora que es gaudeix d'una bona interfície gràfica.

El més gran avantatge que Visual Studio ofereix és el depurador, que ens permet aturar el programa en qualsevol punt i veure el valor de totes les variables, i en cas de que siguin punters i/o objectes, el valor del que contenen. També es pot veure tota la pila de crides i fer avançar el programa pas a pas, i tot això junt ajuda a que la depuració del programa sigui molt més ràpida.

L'última versió, a més, subratlla en vermell els errors de compilació abans de que intentis compilar, agilitzant encara més el desenvolupament de l'aplicació.

Hi ha hagut algun dubte entre altres candidats com Eclipse, NetBeans o Borland C++ Builder, però al final les opcions mencionades han acabat sent massa atractives com per canviar d'opinió.

3.1.2. Autodesk 3ds MAX

Aquest programa és una eina molt potent i relativament fàcil d'utilitzar que serveix per crear models i animacions 3D.

La llicència de l'última versió d'aquest programa (2012) és molt cara (4.000 €), ja que en teoria està totalment dirigit a artistes gràfics professionals.

Hi ha un altre programa per modelar que és un projecte open source i per tant 100% gratuït (Blender) però és molt menys intuïtiu i l'ús que es farà d'aquesta eina és només per crear models de proves.

Per tant em vaig decantar per la versió de proves de 3ds MAX, amb la que el pots fer servir amb totes les seves opcions durant un mes de forma gratuïta.

Un cop acabat de fer un model, podrem utilitzar el seu exportador a format OBJ, un format bastant simple que el nostre prototip de client serà capaç de llegir i mostrar per pantalla.

3.1.3. MySQL

Aquests programes no són eines necessàries només pel desenvolupament, sinó que també seràn imprescindibles durant la vida del projecte. L'administrador del servidor haurà de mantenir un servei MySQL encès, i també haurà de poder executar consultes sobre aquest a l'hora d'administrar la base de dades.

S'ha triat MySQL perquè és una aplicació gratuïta i cada vegada més estable i avançada, com qualsevol altre servidor de bases de dades relacionals amb suport per consultes SQL (Access, Oracle, Sybase...).

3.1.3.1. MySQL Server

Aquest programa s'executarà com a servei en la màquina que conté la part servidor. És l'encarregat de contenir la base de dades i acceptar consultes (insercions, actualitzacions, baixes o simples consultes de dades) del servidor.

També pot ser executat en una màquina a part, en cas de que es vulgui dedicar tota una màquina exclusivament al servei de la base de dades, però llavors s'haurà de configurar el servidor que connecti amb la seva DNS o IP i no amb localhost.

3.1.3.2. MySQL Workbench

És l'entorn que MySQL ofereix per consultar en la base de dades i/o fer-hi modificacions. A part de ser molt versàtil i avançat, ofereix una interfície gràfica que el fa molt intuïtiu i fàcil d'utilitzar.

Serà executat pel programador del servidor a l'hora de fer proves, per tal d'inserir dades manualment o consultar l'estat de les actuals.

També serà executat per l'administrador del servidor quan ja estigui en funcionament, en cas de necessitar fer manteniment manual de la base de dades o consultar la informació de certs jugadors.

3.2. Llibreries

En pràcticament tots els llenguatges de programació un programa pot importar les funcions d'una llibreria externa ja programada anteriorment, estalviant així molta feina al programador, o simplement assistint-lo en la comunicació amb el hardware de l'ordinador o altres serveis.

En C++ hi ha dos tipus de llibreria: les estàtiques i les dinàmiques.

Les llibreries estàtiques tenen la extensió .lib i cada vegada que es crea l'executable del programa (linking) s'ha d'afegir tot el contingut necessari d'aquestes en l'exe, ralentitzant així el procés de compilació i fent que la mida de l'executable s'incrementi molt en cas de que s'incloguin moltes llibreries.

En canvi, les llibreries dinàmiques (.dll) no depenen gens del projecte en sí i ja venen compilades. Són llegides per l'executable principal a mitja execució (at runtime). El seu problema és que si s'oblida incloure'n alguna al directori on hi ha l'executable, aquest ja no funcionarà correctament. A més, són d'ús més avançat i si es comet algun error configurant la seva inclusió o importació, el programa compila correctament i no es nota l'error fins que s'executa el programa i aquest es tanca avisant del problema.

Per aquest projecte he escollit les llibreries estàtiques, ja que no s'inclouran gaires llibreries externes i l'increment del temps de compilació no és gens notable. A més, han resultat ser més fàcils de configurar i a l'hora de distribuïr el client o instal·lar el servidor, ens podrem oblidar d'aquests arxius extra que les llibreries dinàmiques haurien suposat.

Per instal·lar una llibreria estàtica al projecte, s'han d'incloure dos elements importants en el projecte: els fitxers de capçalera d'aquesta, els quals solen anar en una carpeta anomenada include, i el fitxer .lib. En la configuració del projecte s'han d'afegir els directoris on hi ha aquests fitxers: En la part del compilador s'ha de determinar la carpeta include de la llibreria com a directori addicional d'inclusió i en la configuració del vinculador s'ha d'afegir el directori del fitxer .lib com a llibreria adicional.

D'aquesta manera ja podrem utilitzar en el nostre programa les funcions que les llibreries ens ofereixen.

Seguidament seran descrites les llibreries utilitzades en el projecte.

3.2.1. ENet

Aquesta llibreria serveix d'interfície entre el nostre programa i WinSocks, fent la part de comunicacions molt més senzilla.

Està escrita en C, és multiplataforma i utilitza el protocol UDP.

El protocol UDP és més ràpid que el TCP, ja que no duu a terme tantes comprovacions com aquest segon.

S'ha triat aquesta llibreria perquè està orientada a simplificar les connexions entre programes i sobretot per el fet que utilitza el protocol UDP i no el TCP.

Fem aquesta elecció perquè en un joc en línia és moltíssim més important eliminar la possible latència que assegurar-nos de que tots els paquets arriben correctament.

En la majoria dels casos, un error de comunicacions només comportarà la pèrdua d'un paquet i, per tant el jugador haurà de tornar a intentar-ho de nou, no veurà el que ha passat en la seva interfície o en el pitjor dels casos veurà una cosa incorrecta. Com que tots els successos són duts a terme en un sol programa (el servidor) i el client és sols una interfície, (pràcticament un input/output), val la pena arriscar-se.

Un cas en el què no seria recomanat utilitzar aquest protocol seria en un programa de transferència de fitxers, ja que un sol error en un fitxer gran significaria un gran problema. Però en el nostre cas els errors són molt menys importants que la potencial latència addicional que causaria la utilització del protocol TCP.

3.2.2. OpenGL

OpenGL és la nostra interfície amb la targeta gràfica o GPU de l'ordinador on s'executa el client. Ve instal·lada per defecte amb windows i les llibreries estàndar, així que no ha fet falta instal·lar-la en el projecte.

Hi ha altres llibreries que amplien aquesta, com per exemple Glut, però cap d'elles ha sigut utilitzada per aquest projecte.

Està escrita en C i estructurada de forma molt tradicional. Conseqüentment, no té orientació a objectes i a més el seu ús ha de ser totalment seqüencial.

La simplicitat també caracteritza aquesta llibreria. Un cop has cridat les opcions per configurar la finestra com a llenç OpenGL (personalment aquesta és la part més difícil), dibuixar un polígon resulta molt senzill.

El que acaba sent més complicat en el seu ús és la il·luminació i texturitzat dels objectes, i també resulta incòmode haver de seguir un determinat ordre a l´hora de dibuixar els objectes en un frame.

3.2.3. SOCI

Finalment, la llibreria que ens comunica el servidor del joc amb el servidor MySQL de forma extremadament senzilla.

Està escrita en el més pur estil C++, amb objectes, namespaces i operadors sobrecarregats.

Per utilitzar aquesta llibreria de forma satisfactòria només cal utilitzar 4 funcions d'aquesta:

- session::open, la qual només necessita una cadena amb el host, base de dades i credencials per la base de dades i et connectarà amb aquesta.
- session::operator<<, que rebrà una cadena de caràcters on hi haurà la consulta SQL a fer.
- into, que s'usa en les consultes per posar el que retornin en variables.
- use, per utilitzar una variable en una consulta en comptes d'haver-la d'afegir a la cadena consulta anteriorment.

A més a més, és gratuïta i no hi havien gaires alternatives per connectar amb MySQL. Per això, SOCI ha sigut la llibreria escollida per aquesta tasca.

4. Estructura de l'aplicació

4.1. Visió General

Com s'ha exposat anteriorment en els objectius, aquesta aplicació consta de dos programes: el client i el servidor.

El servidor només serà executat una vegada en un servidor remot, al qual tots els clients tinguin accés, i el client serà executat múltiples vegades des d'ordinadors diferents que tindran connexió a internet, per poder-se comunicar amb el servidor.

Els clients mai es comunicaran entre ells directament. El servidor és el punt de trobada de tots els clients i on tota l'acció succeeix. Tota acció que hi hagi per part del client serà enviada al servidor, i si aquesta té algun efecte que ha de ser visible, ho serà per a tots els demés clients connectats.

A continuació seran descrits els dos programes que formen part del projecte, per separat.

4.2. Part client

La programació d'aquesta part del projecte serà basada en l'estructura d'un videojoc qualsevol, tenint-hi la major part de les seves propietats en comú.

Per tant, podrem dir que ni que el joc en sí no sigui localitzable en aquest codi, estem programant un videojoc.

4.2.1. Bucle Principal

Tot videojoc consta d'un bucle principal, cada volta del qual serà un frame mostrat per pantalla.

Abans d'entrar en aquest, però, s'han d'inicialitzar tots els objectes que formaran part del joc: la finestra de windows, el sistema OpenGL i el gestor de xarxa.

La condició de sortida d'aquest bucle s'haurà de complir quan el joc s'hagi de tancar, és a dir, quan l'usuari hagi fet clic a la creueta de dalt a la dreta o qualsevol altra opció que faci que el joc acabi la seva execució.

El contingut del bucle principal consta de tres funcionalitats: primer comprovar l'input, seguidament actualitzar l'estat dels objectes que formen part del joc segons el temps que hagi passat i finalment generar l'output.

La comprovació de l'input, en el nostre cas, constarà de dues parts: el que ens digui l'usuari de l'ordinador (teclat i ratolí) i el que ens digui el servidor (xarxa).

L'actualització dependrà de l'estat actual del joc, però principalment consisteix en actualitzar la posició dels objectes que posteriorment s'han de dibuixar, depenent de l'input que s'ha detectat anteriorment i del seu estat actual (per exemple, si tenen velocitat s'han de moure una mica cap a on va la velocitat).

Finalment, l'output o refresc de pantalla consisteix en dibuixar de nou l'escena amb tots els objectes presents, els quals hauran canviat de posició o qualsevol altre estat que es pugui veure quan són dibuixats.

Aquestes tres funcionalitats s'han de dissenyar de manera que s'executin molt ràpidament, ja que cada volta al bucle ha de ser un refresc de la pantalla i per anar bé hauríem de tenir com a mínim 30 frames per segon, és a dir, que la execució del contingut del bucle hauria de trigar com a màxim 1000 / 30 = 33 mil·lisegons.

Un cop es compleix la condició de sortida del bucle, el programa haurà d'alliberar tots els objectes, tancar la connexió i finalitzar els gestors.

4.2.2. Gestor d'Estats

Quan el joc inicii, haurà de demanar unes credencials a l'usuari per connectar amb el servidor. Un cop siguin validades, haurà de triar un personatge d'entre els que tingui o crear-ne un, el fet de poder crear un personatge en un menú a part i finalment el món en sí. Com que només tenim una finestra i un llenç OpenGL, haurem d'anar canviant d'estat del joc i segons aquest tindrem uns objectes en pantalla o uns altres.

Una manera molt eficaç de resoldre aquest problema és amb una màquina d'estats. Cada estat serà un objecte amb les seves pròpies funcions carregar, actualitzar, dibuixar i alliberar, les quals seran cridades en el bucle principal si aquest estat és l'actualment actiu.

L'algorisme de la gestió d'estats té l'aspecte següent:

```
mentre (GestorEstats::EstatActual() != SORTIR)
{
	GestorEstats::CarregarEstatActual();
	mentre (GestorEstats::EstatActual() == GestorEstats::EstatSeguent())
	{
		ActualitzarInput();
		GestorEstats::ActualitzarEstatActual();
		GestorEstats::DibuixarEstatActual();
	}
	GestorEstats::AlliberarEstatActual();
	GestorEstats::EstatAnterior(GestorEstats::EstatActual());
	GestorEstats::EstatActual(GestorEstats::EstatSeguent());
}
```

Quan vulguem passar d'un estat al següent, només haurem de cridar la funció GestorEstats::EstatSeguent(ESTAT) i al final de la volta de bucle s'alliberarà l'estat actual, a més de carregar-se l'estat següent al començar la següent volta de bucle.

D'aquesta manera, podrem tenir objectes estat amb les seves pròpies funcions les quals podrem programar segons el comportament que volem que cada estat tingui.

4.2.3. Carregant dades

Cada vegada que es canvia d'un estat a l'altre i també al principi de l'execució, quan tenim un estat inicial, s'ha dínicialitzar l'estat actual.

En la funció Carregar dels estats hi crearem i/o carregarem tots els objectes interns d'aquest.

Els objectes que formen part d'un estat són majoritàriament elements que es poden dibuixar o tindran una influència en l'aspecte del renderitzat en pantalla, és a dir, models, materials, textures, fonts, botons o llums.

Els models, els materials i les textures contenen estructures de dades grans, ja que són elements complexes, i, per tant, no es poden crear amb simples paràmetres de configuració. S'han de carregar des del disc dur, llegint cert format.

Els models 3D que carregarem utilitzen el format .obj, el qual ens especificarà tot el que necessitem per dibuixar-los en pantalla.

Els materials són bastant més senzills i normalment s'aplicaran sobre els objectes, per donar un color i demés aspectes gràfics als polígons d'un model.

Finalment, les textures són simples imatges (llegirem el format Targa) que es dividiran i dibuixaran sobre els polígons, segons les coordenades de textura que el model tingui especificades.

4.2.3.1. Models

Els models consten de quatre conjunts de dades: els vèrtexs, els vectors normals, les coordenades de texturitzat i les cares.

En el format .obj de Wavefront aquestes dades són llistades amb un format bastant senzill, cada element en una línia separada:

Si s'utilitzaran materials en el model, el primer a especificar-se serà el nom de la llibreria de materials que s'utilitzarà. La forma en què s'especifica és “mtllib fitxer.mtl”. Posteriorment s'explicarà com es carreguen les llibreries de materials.

Els vèrtexs tenen coordenades amb valors decimals absoluts i són les posicions de cada vèrtex que forma part del model. La forma en la què són especificats en els fitxers és “v X Y Z”, on X Y Z són les coordenades expressades en forma decimal amb 5 xifres significatives.

Els vectors normals són vectors normalitzats que s'utilitzaran per especificar la normal que té cada vèrtex per cada cara per així crear un efecte de suavitzat gràcies a la il·luminació. Són expressats amb el format “vn X Y Z”.

Les coordenades de texturitzat poden prendre valors del 0 al 1 i són punts que se situen en una textura sent el punt 0,0 la cantonada esquerra-amunt i el punt 1,1 la cantonada dreta-avall. Per dibuixar un model, dividirem una textura en diversos trossets gràcies a aquest sistema i aplicarem cada trosset a una cara diferent, creant així un texturitzat del model. Aquestes coordenades es llegeixen amb el format “vt X Y Z”, on Z sempre és 0 i es pot ignorar.

Finalment, les cares estan formades per grups de vèrtex, i cada vèrtex formant part d'una cara estarà associat a una normal i unes coordenades de texturitzat.

Per identificar els elements quan els especifiquem en les cares, s'haurà assignat a cadascun un número mentre es llegien. El primer vèrtex serà el número 1, el 235è el número 235, etc. I el mateix amb els vectors normals i les coordenades de texturitzat.

El format de les cares és el més complex: consta d'un nombre de conjunts d'elements especificats amb els seus identificadors: “f v1/vn1/vt1 v2/vn2/vt2 v3/vn3/vt3 v4/vn4/vt4”, on vN, vnN i vtN són identificadors d'elements. Normalment les cares són de 4 vèrtexs, però poden ser-ho de 3 o més de forma indefinida.

A més d'això, abans d'especificar-se un grup de cares es pot indicar un material a utilitzar per aquest amb “usemtl nomMaterial”, on nomMaterial és el nom identificador del material a utilitzar, llegit anteriorment de la llibreria de materials especificada al principi.

Per desar i llegir els objectes i materials s'ha escollit el format Wavefront per la seva simplicitat i facilitat de lectura comparada amb la d'altres formats, a més d'estar desats en format text i així donar-nos la possibilitat d'escriure'n un sense la necessitat d'un editor 3D amb exportador.

4.2.3.2.Materials

Els materials en el nostre prototip consten de diverses propietats bàsiques:

- Color ambient: és el color del material en sí, expressat en format RGB (Red Green Blue o Vermell Verd Blau).
- Color difós: sol ser exactament el mateix que el color d'ambient.
- Color especular: és el color al qual tendirà reflectir la llum en els punts on sigui més especular. Normalment és blanc o una tonalitat gris.
- Transparència: el tant per u de transperència que té el material, sent 0.0 opacitat pura i 1.0 invisibilitat.
- Brillantor: com més alt sigui aquest valor, més concentrats estaran els punts de reflexió de llum (zones especulars).
- Mapejat de textura: és el nom de la textura que serà mapejada amb aquest material, substituïnt el seu color d'ambient.

El format dels colors en el format Wavefront material és com el de les coordenades en els objectes: tres números decimals separats per espais.

Una llibreria de materials pot contenir més d'un material, així que el nom de cadascun d'ells serà declarat just abans de definir-ne les propietats de la forma “newmtl nomMaterial”.

Aleshores, les seves propietats seran especificades amb els següents codis seguits d'un espai i el seu valor corresponent:

- Color ambient: “Ka”.
- Color difós: “Kd”.
- Color especular: “Ks” (amb “s” de “specular”).
- Transparència: “Tr”.
- Brillantor: “Ns” (amb “s” de “shininess”).
- Mapejat de textura: “map_Ka” i “map_Kd” pels dos colors ambient i difós. També solen ser exactament el mateix, en cas de que s'estigui aplicant alguna textura.

Un cop s'apliqui un material sobre una malla o grup de cares, aquestes mostraran un aspecte molt més realista i atractiu.

4.2.3.3.Textures

Les textures són part dels materials, com aquests són part dels objectes o malles. Els materials ens especifiquen el nom del seu fitxer, i a partir d'aquí les hem de poder carregar.

Les textures utilitzades en aquest projecte són les de format Targa, un tipus de fitxer molt utilitzat per desar textures.

El fitxer targa, com tot fitxer d'imatges, està en format de bits. Consta d'una capçalera i un cos.

En la capçalera del fitxer hi podrem trobar informació com el tipus de color, les dimensions, la profunditat de bits o el tipus d'imatge.

En el cos trobarem la informació relativa a cada píxel de la imatge, ocupant 3 caràcters de longitut cadascun: un pel valor de blau, un pel valor de verd i l'últim pel valor de vermell.

Després de llegir la imatge, haurem de girar l'ordre dels colors (de BGR a RGB) i seguidament girar la imatge segons les dades de la capçalera indiquin (verticalment, horitzontalment o d'ambdues maneres).

4.2.4. Entrada

A l'hora de detectar l'entrada de l'usuari, primer s'ha de comprovar si n'hi ha amb la funció PeekMessage.

De no ser així, el sistema quedaria en espera fins que l'usuari efectués algun moviment amb el ratolí o premés alguna tecla, cosa que bloquejaria el bucle principal de manera que no s'actualitzaria gens freqüentment.

En cas d'haver-se detectat algun event de windows, s'ha de detectar quin tipus d'event és.

El nostre prototip detecta els següents:

- WM_CLOSE i WM_DESTROY: aquests events indiquen que l'usuari està tancant l'aplicació amb la creueta, així que farem que el següent estat sigui el de SORTIR cridant a la funció GestorEstats::SeguentEstat(SORTIR).
- WM_ACTIVATE: aquest event ens indica que el nostre programa ha deixat de ser la finestra principal o ha sigut minimitzat, o tot el contrari, que ha passat a ser la finestra principal. Per saber quines de les dues possibilitats ha succeït comprovarem el segon paràmetre de l'event. Si és WA_INACTIVE voldrà dir que ha perdut el focus i farem que la taxa de refresc passi a ser de només 5 frames per segon.

- WM_SYSCOMMAND: si el segon paràmetre d'aquest event és SC_SCREENSAVE o SC_MONITORPOWER voldrà dir que l'ordinador està intentant passar al salvapantalles o apagar el monitor perquè hi ha hagut cert temps d'activitat. Nosaltres retornarem true per evitar que s'activi el salvapantalles i que s'apagui el monitor.

- WM_KEYDOWN: això vol dir que s'ha premut una tecla i que el segon paràmetre és equivalent a l'identificador de la tecla premuda. Tenim dos arrays global de booleans que memoritzen l'estat de les tecles: un per saber si estan premudes i l'altre per saber si s'acaben de prémer en aquest instant (només és true durant un frame quan es prem la tecla). Comprovarem que el primer array en la posició d'aquesta tecla sigui cert i en cas de que no, posarem el segon array en aquesta posició a true. Seguidament posarem el primer a true.

- WM_KEYUP: significa que s'ha deixat anar una tecla, i reaccionarem posant l'array de tecles premudes en la posició de la tecla premuda a false.

- WM_LBUTTONDOWN: això vol dir que s'ha fet clic amb el botó esquerre del ratolí. Tenim un sistema similar al de les tecles però de només dos booleans, per saber si s'acaba de fer clic o s'està clicant el botó esquerre del ratolí.

- WM_LBUTTONUP: posarem el booleà de que estem clicant a false.

- WM_RBUTTONDOWN i WM_RBUTTONUP: exactament el mateix que amb el botó esquerre del ratolí, però amb unes altres variables i detectant el botó dret.

- WM_MOUSEMOVE: gràcies a aquest event podrem mantenir-nos actualitzats de la posició actual del ratolí. Si és detectat, mitjançant el tercer paràmetre de l'event podrem saber a quines coordenades ha canviat la seva posició i actualitzar les nostres variables de posició del ratolí.

Així que després de detectar l'input de l'usuari, l'estat de certes variables globals s'haurà actualitzat segons el que s'ha llegit i podrem procedir a actualitzar l'escena tenint en compte tots els events d'input llegits.

4.2.5. Xarxa

Gràcies a la llibreria ENet, el fet de comprovar si hi ha paquets, quants n'hi ha i en cas de que sí col·locar les dades rebudes en una estructura d'event és tan senzill com cridar la funció enet_host_service.

La estructura ENetEvent consta de 5 elements: el tipus d'event (recepció de paquet o desconnexió), el Peer que ha causat l'event (en aquest cas sabem que sempre serà el servidor, ja que només tenim una sola connexió amb ell), l'id del canal, un enter amb dades que no utilitzarem i el paquet que s'ha rebut (només si l'event és de recepció d'un paquet).

En cas de que rebem l'event de desconnexió, avisarem de que ens hem desconnectat del servidor i tornarem a la pantalla inicial amb la funció GestorEstats::EstatActual(TITOL).

Si l'event és de recepció d'un paquet, aleshores haurem de llegir-lo i canviar dades segons el tipus paquet que sigui i les dades que contingui, cosa que serà explicada amb més detall en l'apartat 4.4 Protocol.

4.2.6. Actualitzant l'Escena

La funció que actualitza els components de l'escena rep un paràmetre: el temps que ha passat des de l'última actualització (frame time o delta time). D'aquest temps dependran tots els canvis que tinguin a veure amb les físiques o els possibles events temporitzats.

En aquesta funció canviarem el valor de la posició i/o altres dades dels objectes segons l'input que haguem detectat, el temps que hagi passat (delta time) i els paquets rebuts.

4.2.6.1. Input

En els estats que continguin àrees de text, s'haurà de traduïr l'input detectat a un caràcter en l'àrea de text activa. Això suposa un problema, ja que la configuració del teclat difereix segons l'idioma que tingui l'usuari en el seu ordinador, és a dir, que els caràcters que estan associats a cada tecla poden diferir. En aquest prototip hem donat suport bàsic i general al teclat, sense detectar l'idioma d'aquest.

Les tecles també es poden assignar a accions segons l'estat, només s'ha de comprovar l'array de booleans de tecles premudes en la posició desitjada.

També tenim l'input del ratolí. Gràcies a la detecció d'input efectuada anteriorment podem saber si un botó del ratolí està clicat i si acaba de ser clicat, a més de la seva posició. Amb aquestes dades podem detectar si s'ha fet clic a un botó o element 2D de l'escena.

Per detectar si estem fent clic a un element 3D de l'escena, haurem de projectar les coordenades del ratolí en aquesta primer calculant la profunditat ortogonal en la que estan les coordenades 2D amb la funció glReadPixels, i després calculant les coordenades 3D del ratolí en l'escena a partir d'aquesta profunditat utilitzant la funció gluUnProject.

Les coordenades detectades seran les del primer objecte amb el que el ratolí col·lisioni, i a partir d'aquestes podrem calcular si el ratolí està “col·lisionant” amb l'objecte, així detectant si estem fent clic a un objecte 3D de l'escena.

4.2.6.2. Físiques

Tots els objectes 3D de l'escena tenen propietats físiques, totes sent vectors tridimensionals: posició, velocitat linear, direcció a la que mira i velocitat angular.

A la posició de cada objecte se li sumarà la seva velocitat actual multiplicada pel temps de frame, fent així un efecte de moviment cap a la direcció on apunta la seva velocitat i a una velocitat equivalent al mòdul d'aquesta per segon.

El mateix s'aplica a la rotació de l'objecte, però en aquest cas s'aplica a les coordenades independentment/seqüencialment, ja que la rotació en realitat no consisteix en un vector, sinó en tres eixos i tres angles.

Quan es dibuixi l'escena es notarà el canvi en la posició dels objectes que tenien velocitat diferent a 0, i just per això hem d'estar actualitzant i redibuixant l'escena constantment.

4.2.7. Dibuixant escena

Aquí és on entra la seqüència de crides a funcions d'OpenGL. Primer de tot, renderitzarem els objectes tridimensionals de l'escena:

- Ens assegurem d'estar amb la matriu de projecció, és a dir, en el mode perspectiva i activem el mode DEPTH_TEST perquè el motor col·loqui els objectes més propers a la càmera a sobre, i no els últims que han sigut dibuixats.
- Activarem les llums, utilitzant glEnable(idLlum) i seguidament una seqüència de crides a les funcions glLightfv(idLlum, propietat, valor) depenent del tipus de llum que estiguem renderitzant.
- Renderitzarem tots els objectes tridimensionals presents a l'escena, iterant sobre una llista. En el següent apartat s'explicarà amb més detall com renderitzem un objecte amb OpenGL.
- Desactivem les llums amb la funció glDisable(idLlum).

Seguidament, renderitzem els objectes que estan “enganxats al monitor”, els elements de la interfície bidimensional (botons, etiquetes...):

- Posem la matriu en mode ortogonal, desactivem el mode DEPTH_TEST i activem les transparències (mode BLEND).
- Dibuixem els elements 2D amb simples polígons bidimensionals (crides a la funció glVertex2f).
- Desactivem les transparències.

L'ordre d'execució seguit és aquest perquè així ens assegurem de que tots els elements de la interfície van a sobre, i a més desactivant el mode DEPTH_TEST l'ordre de dibuix d'aquests elements dependrà de l'ordre en què hagin sigut afegits a la llista.

4.2.7.1. Renderitzant objectes

Un objecte tridimensional està format per una malla (del seu fitxer .obj) i els seus corresponents materials, a més de les seves propietats físiques i l'escala.

Les posicions dels vèrtexs de la malla han sigut normalitzades al carregar-la, així que l'objecte per defecte té un diàmetre de 1 i està a la posició 0,0.

Per això, abans de dibuixar-lo, li aplicarem les tres matrius de transformació bàsiques:

- Translació: per moure el lloc on serà dibuixat l'objecte. La funció a cridar és glTranslatef(X, Y, Z).
- Rotació: per rotar l'objecte. La funció d'openGL per rotar un objecte rep (glRotatef) 2 paràmetres, el primer és l'angle i els tres següents formant l'eix de rotació. Com que nosaltres tenim la rotació desglossada en els eixos X, Y i Z, la cridarem 3 vegades: una per la rotació sobre l'eix X, un per l'Y i una altra pel Z
- Escala: el nostre objecte està normalitzat i potser el volem fer més petit o més gran. La funció glScalef(X, Y, Z) escalarà el nostre objecte com li especifiquem, però hi ha un problema: també escalarà els vectors normals dels vèrtexs en les cares, cosa que espatllarà les llums, ja que les normals han d'estar normalitzades perquè la reflecció de la llum i els materials funcionin. Hi ha un mode OpenGL que evita aquest problema, ni que sigui a un cost de renderitzat afegit: NORMALIZE.

L'ordre en què son afegides aquestes matrius a l'actual és important, ja que no és el mateix rotar l'objecte i després escalar-lo sobre les coordenades normals que no pas fer-ho al revés. El resultat és molt diferent.

Un cop llesta la matriu de transformació, renderitzarem la malla en sí.

Iterarem sobre cada cara de la malla, i dins de cada iteració farem el següent:

- Comprovar el material d'aquesta cara i si ha canviat, activar-lo (explicat en l'apartat següent).
- Comprovar el nombre de vèrtexs que té la cada. Si en té 3, comencem un triangle (glBegin(GL_TRIANGLES)), si en té 4 comencem un quadrilàter (glBegin(GL_QUADS)) i si en té més de 4 començarem un polígon (glBegin(GL_POLYGON)).
- Per tots els vèrtexs d'aquesta cara, primer especificarem el vector normal del vèrtex amb glNormal3f(X, Y, Z). Seguidament, si té texturitzat, especificarem la seva coordenada de texturitzat amb glTexCoord2f(X, Y) i finalment especificarem la posició del vèrtex amb glVertex3f(X, Y, Z).
- Quan haguem especificat tots els vèrtexs d'aquesta cara, simplement haurem de cridar la funció glEnd() per dir-li a la targeta gràfica que ja hem especificat la cara i la pot dibuixar.

I d'aquesta manera ja haurem renderitzat un objecte, amb els seus materials inclosos.

4.2.7.2. Utilitzant Materials

Per utilitzar un material, només s'han de fer quatre crides a funcions:

- Color ambient: glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, color) on color és un array de floats contenint els valors RGB del color ambiental del material.
- Color difós: exactament igual que amb el color ambient, però posant GL_DIFFUSE com a segon paràmetre i utilitzant el color difós del material en el tercer paràmetre.
- Color especular: també s'especifica de la mateixa manera, utilitzant el valor GL_SPECULAR.
- Brillantor: com que només és un número, l'especificació d'aquest paràmetre és glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, brillantor).

Després de preparar el material, tots els polígons que siguin dibuixats estaran pintats amb aquest fins que no se'n prepari un altre o s'acabi de dibuixar el frame.

4.3. Part servidor

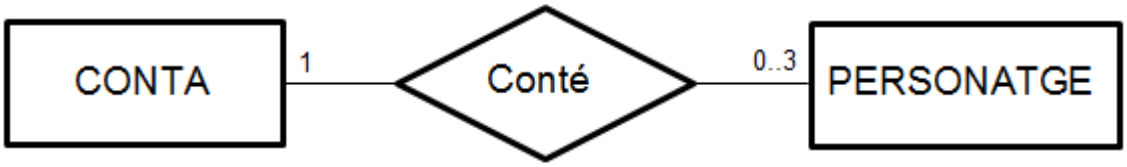
El servidor haurà de gestionar múltiples connexions amb clients i dur a terme tot el que passa en el món virtual, mantenint a tothom qui estigui connectat actualitzat del que passa i alhora rebent els paquets dels clients i actuant en conseqüència. També gestionarà una connexió amb un servidor MySQL per consultar i desar les dades necessàries, siguin credencials d'usuaris o dades de personatges.

4.3.1. Base de Dades

Abans de poder-nos comunicar amb un servidor MySQL, aquest ha d'estar instal·lat i executant-se, a més de contenir la base de dades i taules necessàries.

Instal·lar un servidor MySQL és molt senzill, només hem d'anar a la seva pàgina, escollir un mirall de descàrrega i un cop descarregat seguir els passos de l'instal·lador.

Seguidament, amb el MySQL workbench, podrem crear la base de dades i afegir-hi les taules necessàries. La base de dades del nostre prototip consta de només dues taules relacionals: accounts (contes) i characters (personatges). Aquesta és la seva estructura:



Accounts:

- name (cadena): és el nom de conta amb el que els usuaris accediran.
- password (cadena): la contrassenya corresponent a la conta, encriptada amb l'algorisme MD5. Per comparar-la amb la que l'usuari entri, només cal encriptar aquesta segona també a MD5 i comparar els resultats.

- last_ip (cadena): aquí es guarda la última IP amb la que l'usuari ha accedit, per així nosaltres poder tenir un control aproximat de les persones que entren amb múltiples contes o el país des del que es connecten.

Characters:

- char_id (enter): és el número identificador del personatge, el qual serà utilitzat posteriorment també com a clau en el mapa on són carregats els objectes personatge presents al món, tant al client com al servidor.
- char_name (cadena): és el nom del personatge en qüestió, el qual serà mostrat per pantalla en el client, a sobre de cada objecte 3D que el representi.
- **account_name** (cadena): és una clau forana cap a la taula accounts, ja que cada conta pot contenir 3 personatges.

Tal i com s'ha explicat en la descripció de la llibreria SOCI, un cop estigui tot instal·lat i hi hagi una connexió establerta correctament, fer consultes a la base de dades des del servidor serà una tasca tan fàcil com inserir la cadena consulta a l'objecte connexió i rebre o inserir les dades d'aquesta com a paràmetres.

Exemple d'inserció d'una conta:

```
_sql << "INSERT INTO accounts (name, password, last_ip) VALUES (:name, :password,
:ip)", soci::use(username), soci::use(cryptedPass), soci::use(ipAddress);
```

Exemple d'actualització de la IP d'una conta:

```
_sql << "UPDATE accounts SET last_ip = :ip WHERE name LIKE :name",
soci::use(ipAddress), soci::use(username);
```

Exemple de consulta de les dades d'un personatge:

```
_sql << "SELECT char_id, char_name FROM characters WHERE account_name LIKE
:name", soci::into(ids), soci::into(names, ind), soci::use(peer->accountName);
```

4.3.2. Manejament de tots els clients

La part de xarxa del servidor també utilitza ENet, i s'inicialitza amb la mateixa configuració, amb la diferència que acceptarem múltiples connexions en el nostre objecte ENetHost. Tindrem un bucle principal en el què anirem comprovant els paquets que rebem de cada client i actuant segons aquests.

Quan rebem una connexió d'un client, li assignem un objecte ENetPeer.

Si les credencials que ens ha enviat són vàlides, el mantindrem connectat amb el servidor, i sinó li enviarem un missatge d'error especificant la causa amb un codi d'error i el desconnectarem.

Un cop el client hagi triat el seu personatge i ens envii el paquet que indica que ja ha carregat l'estat principal del joc (el món), carregarem aquest jugador de la base de dades i l'afegirem a una estructura de dades (map) amb el seu identificador com a clau.

Seguidament, li enviarem la seva informació i la de tots els altres personatges que l'envolten, perquè els dibuixi a tots.

Si un client es desconnecta, el seu objecte jugador serà tret del map i es farà un broadcast (enviament massiu d'un paquet) de que aquest jugador se n'ha anat, perquè els objectes client també l'esborrin i no el dibuixin més.

4.3.3. Broadcasts

En el servidor, hi ha successos que només s'han de notificar a un sol client i d'altres que s'han de notificar a tots els que hi hagi connectat.

Cada objecte jugador té associal el seu corresponent Peer o client al qual s'han d'enviar els paquets. Per tant, el fet d'enviar un paquet a un client, tenint l'objecte personatge corresponent a mà, és molt fàcil.

Quan es tracta d'enviar un paquet a tots els usuaris connectats, direm que estem fent un broadcast.

Per fer el broadcast d'un paquet, aquest pot ser creat només una vegada i aleshores enviat a tots els jugadors. Per això els tenim tots guardats en un std::map d'objectes jugador amb clau el seu id, i el que haurem de fer és simplement iterar sobre aquest mapa.

Per implementacions futures en què tinguem un món molt més gran, s'hauran de comprovar les posicions dels jugadors depenent del punt des del que es fa el broadcast (normalment des de la posició d'un d'ells), i enviar el paquet als que estiguin dins de cert radi d'aquest.

I encara més endavant, per donar suport a milers de connexions simultànies, s'hauran de mantenir els personatges en diferents contenidors (maps) segons la regió en la que estiguin, per així evitar el fet d'haver de comprovar les posicions de tots aquests per cada broadcast que es vulgui efectuar.

4.4. Protocol

En tot tipus de comunicació entre un emissor i un receptor, hi ha d'haver un llenguatge que els dos coneguin, en el qual l'emissor parli i el receptor sàpiga escoltar.

En el cas de les comunicacions entre programes informàtics, això s'anomena protocol. Però dels d'alt nivell, no de la categoria del TCP/IP o el Token Bus: un protocol dissenyat pel programador de l'aplicació per poder passar dades entre una màquina i l'altra.

A l'hora d'enviar les dades, primer haurem d'omplir un buffer de bytes que seguidament serà enviat al destinatari. Aquest buffer s'omplirà amb les funcions writeX(dades), on X depèn del tipus de dades que s'hi “escriguin”.

Un cop el receptor obtingui aquest buffer, el podrà desglossar en les mateixes parts utilitzant les funcions readX(), en les quals X també depèn del tipus de dades llegit i retornat per la funció.

En el nostre protocol, tindrem 5 tipus de dades:

- Byte o Char (C): Ocupa 1 bytes i pot ser usat per variables que prenguin valors de -128 a 127 o de 0 a 255.
- Short (H): Ocupa 2 bytes i pot prendre valors més grans (de -32768 a 32767 o de 0 a 65535).
- Integer (D): Ocupa 4 bytes i hi podrem inserir nombres enters dels de sempre.
- Long (L): Un nombre enter amb 8 bytes dedicats al seu alt rang de valors.
- Float (F): També ocupa 8 bytes (com un double, no com un float), però està dedicat als nombres de coma flotant, és a dir, que amb aquestes funcions podrem enviar números decimals.

- String (S): Les cadenes de caràcters. No tenen una mida exacta, sempre ocuparan un byte per cada caràcter més un altre, el qual serà 0 i estarà al final de tot per marcar fi de cadena. La funció readS() anirà llegint caràcters fins que trobi el 0, el qual també treurà del buffer.

Així que amb l'objecte Paquet que conté el buffer i aquests mètodes, enviar dades d'un programa a l'altre és molt més fàcil. Només cal tenir els dos programes amb el protocol sincronitzat. De no ser així hi haurà un gran error al passar-se dades.

Tots els paquets començaran amb una C, la qual és el número identificador de cada paquet.

Seguidament, s'explicaran els paquets que han sigut dissenyats per aquest prototip. Cada part d'aquests serà descrita amb el format X(nom) on X serà la lletra corresponent al tipus de dades d'aquesta part i nom el seu nom o una petita descripció del que s'envia.

4.4.1. Paquets Servidor-Client

Aquest són els paquets que el servidor enviarà i el client llegirà:

- SP_AUTH: C(0), D(missatge).

  És la resposta a un intent d'autenticació, on missatge porta 0 si la connexió ha tingut èxit o 1 si les credencials són invàlides.

- SP_CHAR_SELECTION_LIST: C(1), D(per) (D(id), S(nom)).

  Aquí s'indiquen els personatges que hi ha a la llista de personatges a triar, just abans d'entrar al món.

  Aquest paquet és de tamany indefinit, ja que depèn del seu valor “per”. Aquesta part ens indica quantes voltes hem de fer a un bucle de lectura DS, on cada D és l'identificador d'un personatge i cada S el seu nom.

- SP_CHAR_CREATED: C(2), D(missatge).

  Aquest paquet s'envia quan el client ha intentat crear un personatge. El missatge pot prendre 3 valors: 0 si el personatge s'ha creat correctament, 1 si l'usuari està intentant crear un tercer personatge (cada conta en pot tenir màxim 3), i 2 si un personatge amb el nom demanat ja existia.

- SP_USER_INFO: C(3), D(id), S(nom), F(posX), F(posY), F(posZ), H(angle).

  En aquest bloc hi tenim la informació del jugador en sí: el seu identificador, el seu nom, la seva posició i l'angle al que mira. Aquest paquet ja forma part de quan el client està dins el món.

- SP_CHAR_INFO: C(4), D(id), S(nom), F(posX), F(posY), F(posZ), H(angle).

  És la informació respecte els altres personatges. Actualment té exactament la mateixa estructura que USER_INFO, però en posteriors versions del prototip es podria voler enviar més informació sobre un mateix que sobre els demés i si els dos fossin un sol paquet s'hauria de refer el protocol afegint un dels dos paquets per diferenciar-los.

- SP_MOVE: C(5), D(id), F(posX), F(posY), F(posZ).

  S'envia a tots els clients propers quan un personatge s'ha mogut. Amb només d'identificador del personatge i les coordenades n'hi ha prou.

- SP_DELETE_OBJECT: C(6), D(id).

  També s'envia a tots els clients. Indica quan un personatge s'ha desconnectat, i el client reaccionarà deixant de dibuixar-lo.

- SP_SAY: C(7), S(nom), S(text).

  Quan algú escriu una cosa s'envia aquest paquet a tothom, perquè es vegi per pantalla qui ho ha dit i què ha dit. També es pot enviar per part del servidor, és a dir, posant un nom creat pel dissenyador o administrador (per exemple, “Déu dels Vents”) i enviant missatges de part d'aquest com a part del joc en sí.

- SP_TARGET_SELECTED: C(8), D(id).

  Aquest paquet no és de broadcast i indica al jugador que acaba de seleccionar a un altre personatge.

4.4.2. Paquets Client-Servidor

Aquest són els paquets que el client enviarà i el servidor llegirà:

- CP_AUTH: C(0), S(usuari), S(contrassenya).

  És el paquet que s'enviarà just al connectar amb el servidor, per identificar-nos i seguidament esperar per una resposta (SP_AUTH).

- CP_CHAR_SELECTION_LIST_REQUEST: C(1).

  És tan sols un paquet per notificar al servidor que el client ha accedit a la pantalla de la llista de personatges, esperant a rebre el paquet corresponent (SP_CHAR_SELECTION_LIST).

- CP_CHAR_CREATE_REQUEST: C(2), S(nom).

  Aquest paquet indica al servidor que un client vol crear un personatge amb aquest nom. El servidor el crearà si pot, i enviarà un “missatge” de resposta amb el resultat (SP_CHAR_CREATED).

- CP_ENTER_WORLD: C(3).

  És un altre paquet de notificació, i en aquest cas es notifica al servidor que el client ja ha arribat a la pantalla del món i està esperant a rebre tota la informació corresponent (SP_USER_INFO i SP_CHAR_INFO si s'escau).

- CP_MOVE_REQUEST: C(4), F(posX), F(posY), F(posZ).

  Quan un usuari dóna clic al terra per moure's en aquella posició, el client primer ha de demanar “permís” per moure's i si tot va bé el servidor en fa un broadcast a tots els usuaris propers (SP_MOVE).

- CP_SAY: C(5), S(text)

  Quan un usuari escriu un missatge i l'envia prement enter. El servidor processarà el text i si no és massa llarg en farà un broadcast. També és opcional afegir un filtre d'insults al servidor i substituïr les paraules del filtre per una altra o simplement decidir descartar el missatge (SP_SAY).

- CP_INTERACTION_REQUEST: C(6), D(id)

  Aquest paquet és enviat quan l'usuari fa clic a un objecte de l'escena, demanant al servidor una interacció amb aquest. El servidor comprovarà si el té seleccionat i en cas de que no, farà que el seleccioni (SP_TARGET_SELECTED).

5. Millores

Fent aquest projecte he posat en pràctica tot el que havia après, a més d'aprendre moltíssimes coses sobre com fer un joc en 3D.

Però el que més valoro d'haver-lo fet és que he pogut cometre errors de disseny i detectar-los i aprendre d'ells un cop ja tenia l'aplicació molt avançada.

Si això hagués passat amb un projecte comercial, hauria hagut de refer gran part del codi, suposant això una gran pèrdua de temps i diners a invertir en el projecte.

L'error més gran que he detectat en la meva aplicació és haver començat amb tantíssimes variables estàtiques i públiques o, encara pitjor, variables globals. Això fa que el programa perdi molta cohesió i encapsulament i conseqüentment les proves per mòduls acabin sent molt limitades. També he comès un error molt greu deixant els comentaris pel final, perquè aleshores ja pot ser que no recordis exactament què fa aquell tros de codi i acabi tot sense comentar.

A més, el fet d'anar ràpid a que tot funcioni en comptes de esforçar-se en aconseguir un programa ben estable i un codi ben net també és un error freqüent que acaba fent que el temps de producció de l'aplicació s'allargui i no s'escurçi, a més de tenir un resultat bastant pitjor del que podria haver tingut.

La part servidor ha acabat tenint molt poca orientació a objectes, i pràcticament tot el que fa el programa està en un fitxer .cpp. Això s'ha de millorar, modulant més l'aplicació i fent que tot siguin objectes, un dedicat a cada cosa. També crec que Java seria un bon llenguatge per programar el servidor, i que utilitzant aquest protocol determinat es podria comunicar perfectament amb el client ni que aquest estigui en C++.

També, com s'ha dit en l'apartat de broadcasts, s'haurà de gestionar la forma en què es fan aquests, dividint el món en diverses parts i gestionant unes llistes de coneixement (knownlist) per a cada jugador, en les què es desaran punters o referències als personatges que cada jugador pot veure en la seva pantalla.

I finalment el fet d'haver triat OpenGL i no DirectX. OpenGL m'ha ensenyat les bases de la programació en gràfics 3D, així que no em penedeixo d'haver-lo escollit per aquest projecte. Però crec que DirectX, amb la seva orientació a objectes i les seves noves versions ofereix un rendiment molt elevat i suport a efectes molt més sofisticats.

6. Bibliografia

Per realitzar aquest treball no s'ha consultat cap document físic, està tot basat en el que he après estudiant a la UVic, l'any que he cursat a DigiPen i practicant amb el servidor L2JServer i estudiant la seva estructura. Quan hi ha hagut manca d'informació sobre certa funció o llibreria, s'han consultat els dubtes en la pàgina web dels desenvolupadors de les llibreries relacionades o exemples-tutorial d'ús.

En cap de les fonts citades posteriorment es trobarà un article del què hagi pogut extreure informació present en aquesta memòria, sinó referències a llibreries i projectes molt extensos dels quals he après al llarg del temps.

A més a més, pràcticament totes les pàgines web tenen múltiples autors o representen empreses o organitzacions.

Per aquests motius esmentats anteriorment, no se seguiran els criteris estàndars per elaborar la bibliografia.

http://www.l2jserver.com

Aquesta és la pàgina del projecte l2j, un projecte d'emulació d'un servidor per al joc Lineage 2 programat en Java.

http://msdn.microsoft.com/es-es/library/ms123401.aspx

La biblioteca per a programadors de Microsoft, on es poden trobar les especificacions de totes les funcions relacionades amb les seves llibreries.

http://www.opengl.org/documentation

La documentació del projecte OpenGL, de la que es poden consultar les funcions d'aquesta llibreria gràfica.

http://enet.bespin.org

Les descripcions de totes les funcions ENet, juntament amb uns petits exemples d'ús de la seva llibreria de xarxa.

http://soci.sourceforge.net

El lloc web d'on es pot descarregar la llibreria SOCI i aprendre com instal·lar-la i utilitzar-la.

http://www.marek-knows.com

Marek A. Krzeminsk va començar a fer el seu joc en OpenGL i penjar vídeos amb els passos que seguia. Gràcies a alguns d'aquests vídeos vaig poder fundar les meves bases en programació de jocs i gràfics OpenGL fa un any.

http://en.wikipedia.org/wiki/Wavefront_.obj_file

Especificació del format Wavefront OBJ i MTL.

http://www.wotsit.org

Aquesta pàgina conté l'especificació de molts formats, d'entre les quals vaig consultar la del TGA.