UNIVERSITAT OBERTA DE CATALUNYA

ENGINYERIA INFORMÀTICA DE SISTEMES

DESENVOLUPAMENT D'UN VIDEOJOC EN 3D BASAT EN OPENGL

Alumne: David Lacasa

Consultora: Eva Monclús Lahoya

2 de Juliol del 2004

ÍNDEX

ÍNTRODUCCIÓ

L'objectiu d'aquest treball ha estat desenvolupar un videojoc en un entorn de 3 dimensions utilitzant les llibreries gràfiques d'OpenGL.

Avui en dia la programació de jocs per PC es divideix bàsicament en dos tipus: els jocs que utilitzen les llibreries gràfiques DirectX proporcionades per Microsoft i el que fan servir l'estàndard OpenGL. OpenGL va ser creada el 1992 per un conjunt d'empreses amb l'objectiu de crear una llibreria gràfica lliure. Degut a que no es necessita cap llicència per utilitzar-la, OpenGL té l'avantatge d'estar implementada en multitud de llenguatges de programació i està disponible per varis sistemes operatius.

Aqueta aplicació, a més, ha estat desenvolupada en Java, la qual cosa dóna més indepència encara, ja que aquesta és precisament la filosofia d'aquest llenguatge de programació. Per fer-la totalment independent, la interacció gràfica del programa amb l'usuari s'ha realitzat amb les llibreries AWT de Java, sense que hi hagi d'intervindre cap programa extern.

Si bé és cert que el desenvolupament i creació d'un programa d'aquest tipus requereix de l'esforç de més d'un programador i de dissenyadors gràfics, amb aquest treball s'ha pretès fer una petita introducció al món de la programació de videojocs per algú que, com jo, no tenia cap experiència en aquest camp.

En la finalització d'aquest treball puc dir que malgrat el molts problemes i dubtes que he tingut, al final estic content del resultat que he obtingut i dels coneixements que he adquirit.

OBJECTIUS

L'objectiu principal del treball ha estat el desenvolupament d'un videojoc. En concret, les característiques que es volien implementar inicialment eren les següents:

- Vista del joc en 3 dimensions on l'usuari controla els moviments de la càmera per desplaçar-se a través de l'entorn (el personatge que mou l'usuari no es mostra).
- L'etorn del programa constarà d'escenaris tant interiors (habitacions tancades) com exteriors.
- L'objectiu del joc serà trobar la sortida en una determinada quantitat de temps.
- Control de la càmera per teclat i, opcionalment, per ratolí.
- Interacció de l'usuari amb objectes de l'entorn del programa (en croncret s'havia pensat amb diverses claus que obririen certes portes del mapa).
- Programació d'efectes (llums, sombres, boira...)

De tots els punts comentats, s'han complert tots menys el penúltim, i l'últim només parcialment. Durant l'últim més i mig han sorgit molts problemes per implementar les llums en el programa, ja que aquestes no es visualitzaven correctament. Tant la consultora de l'assigntaura com jo hem estat intentant trobar la font d'aquests errors, però no ha estat possible resoldre el problema a temps. Això, ha fet que no hi hagués temps per implementar el penúltim punt.
A canvi de les llums, però, s'ha tingut temps d'implementar altres efectes que en un principi no estaven en el plantejament inicial:

- Implementació de diferents sons segons l'acció que fa l'usuari (caminar, obrir una porta...)
- Creació de l'efecte atmosfèric de pluja
- Creació d'una pantalla final segons s'hagi guanyat o perdut la partida

METODOLOGIA

Per fer el desenvolupament del programa, es van intentar seguir els següents passos:

- Anàlisi de les funcionalitats
- Anàlisi de les classes i estructures de dades per resoldre cada funcionalitat
- Implementació de les classes
- Provatura del funcionament

Degut a que era la primera vegada que s'implementava una aplicació com aquesta i a la desconeixença que es tenia de moltes de les funcions i mètodes de l'API d'OpenGL, no es va poder seguir amb exactitud els passos escrits anteriorment. En la majoria de casos, es feien primer provatures de les funcions d'OpenGL i després, segons els paràmetres i els mètodes que es feien servir s'analitzava i s'implementava la classe corresponent.

1.- ANÀLISI DE LES FUNCIONALITATS

En aquesta fase es va fer un esboç del que hauria de fer el prograna, el mapa del que es compondria el joc i les textures que s'utilitzarien en el programa. En concret, es van pensar les següents funcionalitats:

- Vista de l'escenari en 3 dimensions i control de la càmera mitjançant els cursors (vistes laterals i verticals).
- Mapa format per habitacions i connectades entre elles per portes.
- Optimització en el procés de dibuixar el mapa. Degut a que el mapa del joc constava de moltes habitacions, es va decidir que en tot moment el programa només dibuixaria l'habitació on estava l'usuari. Això, comportava que tota habitació havia de ser tancada. No podien haver portes obertes que conectéssin dues habitacions, ja que a través de la porta oberta no es veuria res. Aquesta funcionalitat faria que l'escenari fos una mica repetitu (hi hauria moltes portes), però en canvi es guanyaria en rapidesa d'execució.
- El temps restant per trobar la sortida es mostraria en la part baixa de la pantalla i s'aniria actualitzant cada segon.

2.- ANÀLISI DE LES CLASSES

En la fase d'anàlisi de classes es van definir les classes que s'utilitzarien per poder implementar les funcionalitats. Des d'un principi ja es va pensar en que l'estructura principal estaria formada per les següents classes:

- Classe principal (**Main**) que engegaria el procés del joc. Crearia tota l'estructura necessària inicial (escena i finestres) on es mostraria el joc.
- Classe que pintaria per pantalla els gràfics del joc i s'encarregaria de detectar les tecles que es preméssin per tal d'interactuar amb l'usuari (**Escena**).
- Classe on estaria emmagartzemada tota la informació de les habitacions del joc (**Mapa**).
- Classe que emmagatzemaria la informació relacionada amb una habitació (**Sector**).
- Classe que representaria una llum en el sector (**Llum**).
- Classe que representés un objecte (**Objecte**), una clau, que serviria per obrir certes portes del mapa.
- Classe que tindria la informació dels vèrtex de les parets de les habitacions que es dibuixarien per pantalla (**Triangle**). La raó d'escollir el triangle com a polígon per dibuixar les parets de les habitacions, va ser simplement per la rapidesa. La majoria de targetes gràfiques tenen funcions molt optimitzades per dibuixar aquest tipus de polígon, la qual cosa fa que l'aplicació tingui un millor comportament. Moltes vegades és més ràpid dibuixar dos triangles que no pas un quadrat.
- Classe que representaria un vèrtex amb les seves coordenades (**Vertex**).
- Classe **Rellotge** que tindria un contador qua aniria baixant cada segon.

3.- IMPLEMENTACIÓ DE LES CLASSES

Una vegada es va començar a implementar les classes i de fer les primeres provatures, es va haver de canviar una mica el disseny inicial degut a que es van detectar els següents problemes:

- Inicialment cada Triangle tenia la seva equació del pla on estava dibuixat. Aquesta equació, es feia servir per detectar les direccions on estava cada paret del sector. Així, per cada moviment que fèia l'usuari amb les tecles del cursor es podia detectar la distància de l'usuari amb el pla on estava el triangle, i per tant decidir si hi podia haver una col.lisió. Degut a que totes les equacions dels triangles d'una paret eren iguales, els càlculs de detecció de col.lisions eren repetitus. Per tant es va afegir la classe **Paret.** La Paret, tindria una sola equació del pla (calculada a partir de la normal de qualsevol vèrtex de la paret)

que seria igual a la de tots els triangles que la composaven. Així doncs, ara el Sector no estaria composat de triangles, sinó de parets.

- Els càlculs per definir una equació del pla a partir dels vèrtex d'una paret eren una mica complicats, per la qual cosa es va decidir crear una classe (**Equacio**) que representés l'equació. Com que aquesta classe també fèia servir coordenades, es va crear l'estructura hereditària de les classes **Punt3D**, Vertex i Equacio.
- Per tal de connectar les habitacions, es va decidir crear una nova classe (**Interseccio**). Aquesta classe, tindria informació de les coordenades on es podria fer un canvi de sector i del sector destí que s'hauria de dibuixar quan s'arribés a les coordenades. Gràficament, correspondria a les coordenades on estarien dibuixades les portes. Així doncs, quan l'usuari s'acostés a una paret, a part de mirar si es podruiria col.lisió o no, també es comprovaria si aquesta col.lisó correpondria a les coordenades d'una intersecció. En el cas de que així fos, la classe Escena ho detectaria i dibuixaria el sector destí de la intersecció.
- Degut a que les interseccions es calculaven a partir de l'equació del pla d'una paret, l'aplicació va quedar limitada a una intersecció per paret. Si una paret tingués més d'una intersecció, l'aplicació no seria capaç de decidir quina de les interseccions havia estat creuada. Per solucionar aquest problema, es va decidir que un sector podria tindre més de 6 parets (4 laterals més el sostre i el terra). Malgrat que gràficament es podria veure una paret amb dues portes, per exemple, l'estructura interna del programa definiria dues parets, una al costat de l'altra, amb una intersecció cadascuna que es correspondria a cada porta.
- Quan es va implementar el sector número 9 (sector amb vistes a l'exterior), es va crear un mètode per pintar les textures que simulava un cel en l'infinit. Aquest mètode, va resultar que no funcionava bé quan s'utilitzaven triangles per dibuixar una paret. S'havien d'utilitzar quadrilàters perquè l'entorn es veiés correctament. Degut a això, es va eliminar la classe Triangle i es va crear la classe **Poligon**. Bàsicament, la classe va tenir la mateixa estructura, però en comptes de definir 3 vèrtex per dibuixar el polígon, ara el número de vèrtex podia ser qualsevol. D'aquesta manera, les parets dels sectors dibuixaven poligons de 3 vèrtex (triangles) i les parets que simulaven el cel en dibuixaven quatre.
- Quan es va decidir implementar la classe Rellotge, es va veure que el procés resultava més difícil que el que es crèia inicialment. Finalment es va optar per implementar aquest mètode mitjançant el patró Observer. La classe Rellotge, implementava una classe interna (**Contador**). Rellotge, s'encarregava de crear

un Contador cada segon i executar-la com un fil independent del fil on s'estava executant l'aplicació. Contador, mitjançant l'interfície **RellotgeListener**, simplement excutava un mètode de la classe Escena perquè aquesta actualitzés el temps que sortia per pantalla.

- En l'implementació de la classe Llum és on van sorgir la majoria de problemes. Quan es van fer les primeres proves, es va detectar que el comportament de les llums no era el correcte i que la visualització dels sectors era defectuosa. Durant un mes i mig tant es va estar mirant de detectar els possibles errors que provocaven la una il.luminació tant efectuosa, però malgrat els esforços que es van fer no es van poder detectar a temps. Així doncs, la classe Llum va ser eliminada del projecte. El retard que van provocar aquests errors, va fer que no es tingués temps d'implementar la funcionalitat de l'interacció amb objectes (claus) i per tant la classe Objecte també va ser eliminada.
- Mentre s'intentaven detectar i corregir els errors d'il.luminació, es va tenir temps d'implementar 3 classes que inicialment no estaven pensades per incloure-les en l'aplicació:
    - La classe **So** executa arxius wav segons les accions que fagi l'usuari. Aquesta classe formava part de l'estructura de Escena. Inicialment, es carregaven tots els arxius de so en memòria, i la classe Escena, depenent de les accions que feia l'usuari executava un arxiu o un altre.
    - Les classes **Pluja** i **Gota**, van servir per implementar l'efecte de pluja del sector 9. Gota simplement és un punt en l'espai amb un paràmetre velocitat. Segons aquest paràmetre, la gota caurà més ràpidament o no (és a a dir, el punt es mourà més ràpidament o més lentament sobre l'eix Y). La classe Pluja, s'encarrega de crear totes les gotes dintre d'una superfície donada. Degut a que el dibuix de la pluja fèia servir bastantes línies de codi, es va optar per canviar les responsabilitats de dibuixar. Els mètodes per dibuixar, es van passar de la classe Escena a la classe Sector. D'aquesta manera, seria el propi sector el que s'encarregaria de dibuixar les seves parets i, quan fos el cas, la pluja.

4.- PROVATURA DEL FUNCIONAMENT

En aquesta fase el que es va fer va ser provar el funcionament i comportament del programa a mesura que s'anaven implementant funcionalitats.

Les 4 fases de la metodologia, per tant, no només han estat executades correlativament, sinó que a més ho han fet de un mode cíclic. Per cada funcionalitat (o a vegades classe), es feia un anàlisi, una implementació i una provatura.

En la implementació de les classes,a més, quan el disseny o requeria es buscaven els recursos necessaris per poderá completar (arxius wav, imatges de textures, exemples de codi d'altres aplicacions o tutorials, etc).

PLANIFICACIÓ

La planificació pensada inicialment va ser la següent:

	Descripció de la tasca	**Interval de temps**
1	Planificació de les tasques a realitzar i del pla de treball	23/2 - 7/3
2	Entrega del pla de treball.	8/3
3	Instal.lació i configuració de l'equip de treball (java, opengl,...). Provatures amb l'API d'OpenGL i disseny de les classes del programa.	9/3 - 21/3
4	Implementació de l'esquelet del programa (l'usuari s'haurà de poder moure per un petit entorn en 3D amb algunes textures incorporades.	22/3 - 4/4
5	Entrega de l'esquelet del programa (PAC 1).	5/4
6	Implementació de tot el mapa per on es mourà l'usuari amb il.luminació incorporada.	6/4 - 20/4
7	Implementació del temps en la partida i inici d'implementació de la interacció de l'usuari amb alguns objectes.	21/4 - 9/5
8	Entrega de la PAC 2.	10/5
9	Finalització de la interacció amb els objectes i incorporació d'efectes (sombres, boira, etc).	11/5 - 25/5
10	Redacció de la memòria i del document de presentació virtual.	26/5 - 17/6
	Entrega de la memòria i del document de presentació virtual.	18/6

Degut als errors que es van detectar alhora d'implementar les llums, la planificació no va poder ser seguida segons el pla exposat. En concret, la planificació només es va poder seguir fins al punt 8. Degut al retard, la duració del punt 10 va ser allargat fins al dia 2 de juliol.

DESCRIPCIÓ DE L'APLICACIÓ

1.- REQUISITS

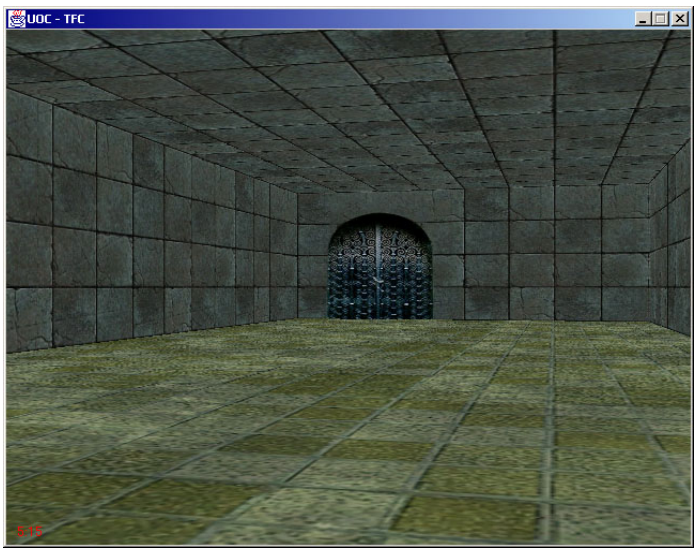
Els requisits per executar correctament l'aplicació son els següents:

- Màquina virtual Java versió 1.3 o superior
- Llibreries correctament instal.lades de GL4Java versió 2.8.2

És important tenir l'última versió de les llibreries GL4Java, ja que en versions anteriors s'han detectat problemes a l'executar l'aplicació. En concret, els problemes venien donats en la gestió de les textures. Per crear-les s'han utilitzat funcions pròpies de la versió 2.8.2. En versions anteriors aquestes crides poden provocar errors aleatoris durant el funcionament del joc.
Per fer desenvolupament i provatures del joc s'ha utilitzat el sistema operatiu Windows 2000 amb 256 MB de memòria RAM, encara que el joc hauria de funcionar correctament en Linux.

2.- MANUAL D'ÚS

L'objectiu del joc és trobar la sortida del laberint on s'està dintre d'un interval de temps determinat (5 minuts i mig). El laberint es composa d'habitacions i passadissos que es comuniquen entre ells mitjançant portes. Per obrir una porta i passar a la següent habitació, l'usuari simplement s'ha de posar en la direcció de la porta i avançar. El control de l'usuari es fa mitjançant el teclat. Les tecles, a més de permetre el moviment, poden activar o desactivar alguns efectes. En concret aquestes tecles son les següents:



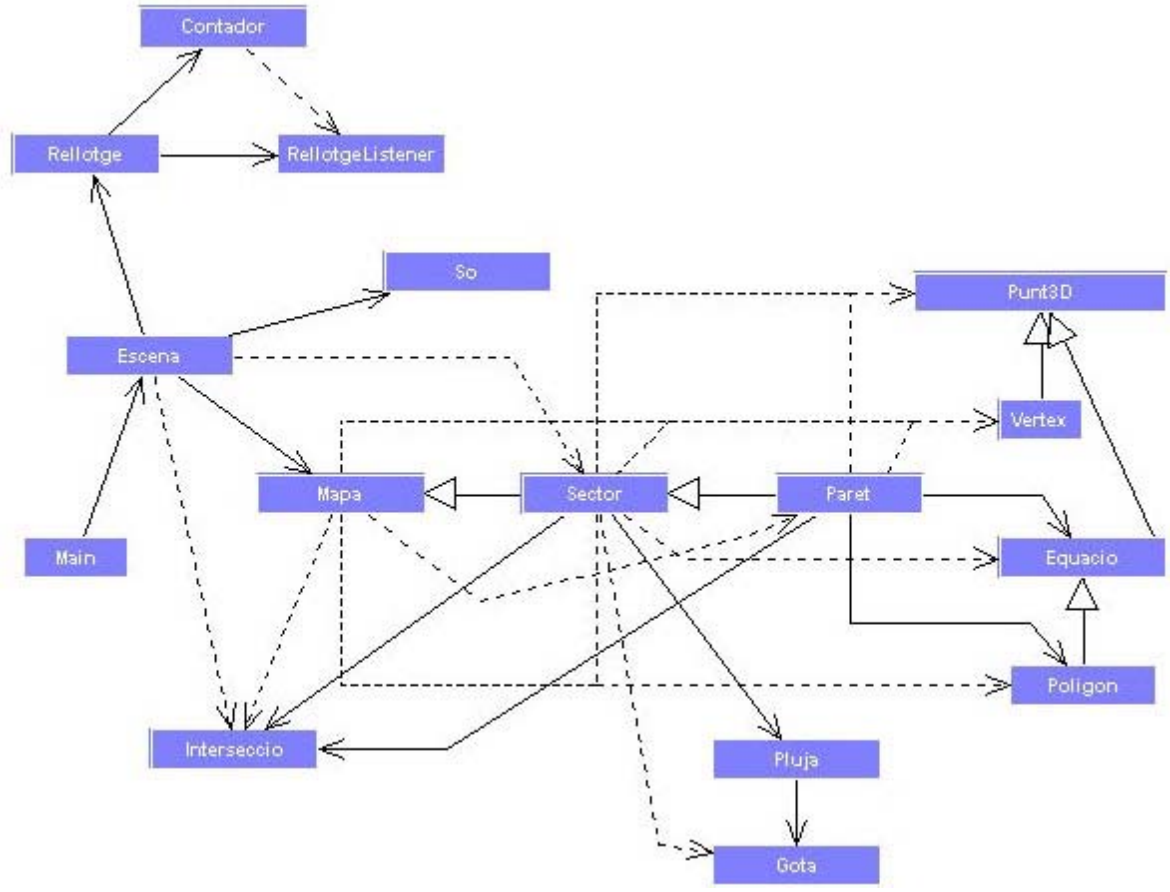
- cursor esquerre i dret: permeten rotar la vista horitzontalment la càmera
- cursor amunt: avança en la direcció on es mira
- cursor avall: retrocedeix en la direcció contrària on es mira

- tecles "Av. pàg" i "Re. pàg.": roten la càmera verticalment (cap amunt o cap avall segons la tecla)
- tecla 'P': permet activar i desactivar la pluja en el sector amb vistes a l'exterior (sector número 9)
- tecla 'B': activa, desactiva o canvia la boira. S'han implementat 4 modes de boira (de menys espesa a més espesa). El primer de tots, i és el que està activat per defecte és el mode sense boira. Cada cop que es premi la tecla 'B', es canviara la boira al següent mode.
- tecla ESC: permet tancar l'aplicació i sortir del programa. És equivalent a tancar la finestra on s'executa el joc.

El joc s'executa amb la comanda "java Main" des d'una consola de comandes i situats a la carpeta on es troben els arxius class.

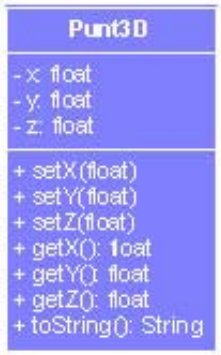
3.- ESTRUCTURA DE DADES

El diagrama de classes utilitzat en l'aplicació ha estat el següent:



A continuació es farà una descripció de cada classe amb les seves funcionalitats i es comentaran els mètodes o fragments de codi que es considerin més rellevants.

3.1.- Classe Punt3D



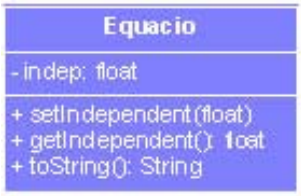
La classe Punt3D es fa servir per representar punts en l'espai. Disposa només dels atributs x, y i z que son les coodenades del punt.

Aquesta classe la fan servir molts mètodes de l'aplicació, sobretot per evaluar la validesa del moviment de l'usuari i per calcular les normals dels poligons.

3.2.- Classe Vertex

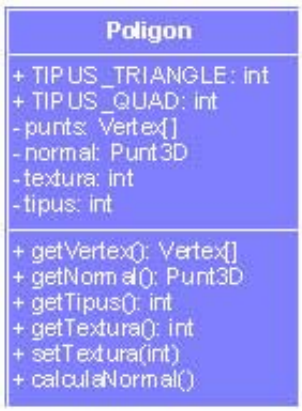
La classe Vertex hereda de Punt3D, a la qual se li afegeixen dos atributs: xText i yText. És una de les classes més utilitzades en l'aplicació, ja que es fa servir per representar els vèrtex dels poligons. Els dos atributs que té indiquen la posició x i y de la textura dintre del polígon.

3.3.- Classe Equacio



La classe Equacio hereda també de Punt3D i es fa servir per representar l'equació en el pla que ocupa cada paret. L'equació del pla seguiex és del tipus ax+by+cz+d=0, on les variables x, y i z són els atributs heredats de Punt3D i d és l'atribut indep de la classe. L'equació en el pla es fa servir durant el joc per calcular la distància de l'usuari amb les parets de les habitacions, i decidir per tant si hi pot haver-hi col.lisió.

3.4.- Classe Poligon



La classe Polígon representa el triangle o quadrilàter del que estan formades les parets. L'atribut tipus, valdrà TIPUS_TRIANGLE o TIPUS_QUAD segons la forma que hagi de tindre el polígon. A part dels vèrtex (atribut punts) dels que està format el polígon, la classe també té informació sobre la seva normal i de la textura que ha d'utilitzar el programa quan el polígon es dibuixi per pantalla.

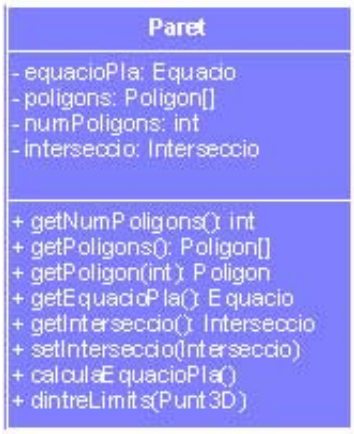
El mètode més important de la classe és el calculaNormal. Aquest mètode calcula la normal a partir dels 3 primers punts del polígon. En la pràctica la normal calculada hauria servit per donar més realisme a

l'aplicació una vegada estigués implementada la il.luminació, però degut a que finalment aquest mòdul no s'ha programat, l'efecte de les normals en el programa no té esdevé tant important.

3.5.- Classe Interseccio

La classe intersecció representa la unió entre dues habitacions (gràficament representades per portes en el programa). Aquesta classe forma part de la classe Paret (és un atribut). Quan el programa detecta que hi ha hagut una col.lisió amb la paret, es mira si la posició on està l'usuari està dintre dels límits de la classe Intersecció de la paret on s'ha produït la col.lisió. El límits d'Intersecció, són els atributs xMin, xMax, yMin, yMax, zMin i zMax. Moltes vegades només interessa calcular l'interval d'una certa coordenada (x o z), per la qual cosa en la creació de la Intersecció (que es fa en la classe Mapa) es posen valors mínims molt baixos i valors màxims molt alts en les coordanes que no interessa calcular l'interval per tal de que la condició sempre es cumpleixi. Per exemple, si suposem una paret situada al llarg de l'eix z y amb x=2 que té una intersecció, quan l'usuari s'acosta a la paret només ens interessarà si la posició de la component z de l'usuari està dintre dels límits de les components z de la intersecció. Les components x i y no farà falta que es tinguin en compte, ja que la x serà sempre constant (valdrà 2) i la y sempre serà la mateixa ja que l'usuari no pot volar i per tant no canvia mai. Per això, al crear la intersecció podriem posar com a valors xMin=-100 y xMax=100. La posició x de l'usuari sempre estarà dintre dels límits, però el que interessa realment veure és si la component z està també en el rang de zMin i zMax. Aquesta classe, també guarda el sector destí a on va a parar l'usuari si atravessa la intersecció, i la posició d'aquell sector on s'ha de situar a l'usuari inicialment quan es dibuixi per primera vegada el sector.

3.6.- Classe Paret



La classe Paret es fa servir per representar una paret, sostre o terra d'una habitació. Conté informació de tots els poligons que formen la paret, així com de l'equació del pla on està situada i de la intersecció, si en té, amb un altre sector.

El mètode calculaEquacioPla, calcula l'equació del pla de la paret a partir de la normal de la paret i d'un punt que està dintre el pla (un vèrtex d'un polígon de la paret).

El mètode dintreLimits ens diu si el punts que li passem està dintre dels rangs de la intersecció de la paret. Aquest mètode es fa servir per detectar si l'usuari està davant d'una porta quan s'ha produït una col.lisió amb la paret.

3.7.- Classe Sector

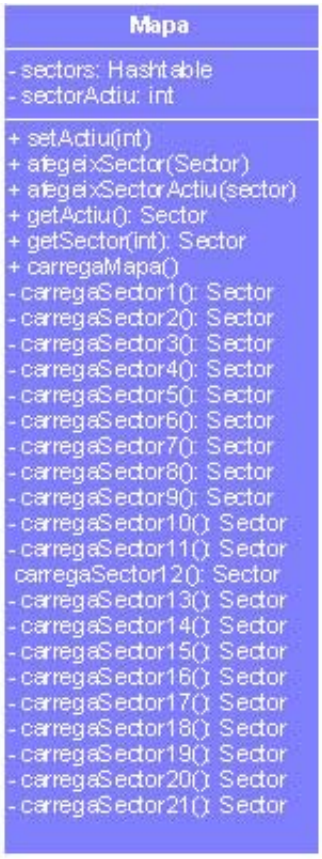
La classe Sector representa una habitació del mapa. Té emmagatzemada les parets que formen el sector, el seu identificador, la intersecció de la paret que creua l'usuari (si n'hi ha alguna) i l'objecte Pluja. En la classe hi ha alguns mètode que cal comentar.

El mètode dibuixaSector fa un recorregut per totes les parets, poligons de la paret i vèrtex de cada poligon i els pinta per pantalla. Segons el tipus de poligon es dibuixaran triangles o quadrilàters. La textura de cada poligon també s'aplicarà quan es dibuixin.

El mètode dibuixaPluja és semblant a l'anterior. Si l'atribut pluja no és nul (això només passa en el sector exterior número 9) es fa un recorregut per totes les gotes de la pluja i es dibuixen. En aquest cas, el polígon que es dibuixa és una línia de llargada 0.05. Una vegada s'ha dibuixat, es crida el mètode avança de la classe Gota perquè la pròxima vegada que es dibuixi ho faci en una posició de l'eix Y més baixa. D'aquesta manera, es simula la caiguda de les gotes.

El mètode evaluaMoviment, retorna MOV_INVALID, MOV_VALID o MOV_INTERSECCIO segons sigui el cas. Per cada paret, es calcula la distància que hi ha del punt que es rep per paràmetres al pla on està la paret. Si aquesta distància és menor o igual a 0.2 vol dir que hi ha una col.lisió. En el cas que el punt estigui dintre dels límits de la intersecció de la paret, es retorna MOV_INTERSECCIO. Si no, s'evaluen les demés parets i si el punt no està dintre del rang de les interseccions es retorna MOV_INVALID. Si no es detecta cap col.lisió es retorna MOV_VALID.

3.8.- Classe Mapa



En la classe Mapa és on es guarda tota l'estructura de l'escenari per on es mourà l'usuari. Bàsicament el mapa només és una llista de sectors guardats en una Hashtable que permet la seva localització a través de l'identificador del sector. L'atribut sectorActiu, indica en tot moment l'identificador del sector on està l'usuari i, per tant, el sector que s'ha de dibuixar per pantalla.

El mètode carregaMapa, fa una crida a tots els mètodes carregaSector de la classe. Cada mètode carregaSector, s'encarrega de crear l'estructura del sector i afegir-la al mapa. En cada sector, es creen les parets que el formen, i per cada paret es creen els poligons i vèrtex que la formen. En cada paret, també es creen les interseccions que es faran servir per canviar de sector.

3.9.- Classe Pluja

Donada una superfíci inicial que se li passa per paràmetre al constructor, la classe Pluja s'encarrega de distribuïr uniformement un número de gotes que dependrà de l'atribut intensitat (com més alt més gotes hi haurà. Perquè la distribució de les gotes no sigui tan uniforme, s'introdueix per cada gota una petita variant aleatòria per cada component (x, y i z). Això, fa que totes les gotes no caiguin al mateix temps ni que hi hagi la mateixa distància entre elles. La component aleatòria es prou petita per tal d'evitar que unes gotes es superposin amb les veïnes. Inicialment, les gotes es comencen a dibuixar a partir de del valor Y igual a l'atribut altura de la classe.

Per tal de simular la caiguda de les gotes, després de dibuixar-les s'actualitza la posició Y de cada gota fent una crida al mètode avança.

3.10.- Classe Gota

Aquesta classe representa una gota que està emmagatzemada dintre la classe Pluja. Els atributs x, y i z permeten saber la posició de la gota en cada instant. els atributs xIni, yIni i zIni tenen la posició inicial de la gota quan aquesta es va crear.
L'atribut velocitat, indica amb quina mesura la gota caurà. Com més gran sigui aquest atribut, més ràpid es reduirà la component Y de la gota quan caigui i, per tant, més sensació de velocitat de caiguda donarà quan es dibuixi per pantalla.
El mètode avança, simplement redueix la component Y segons el valor de velocitat.
El mètode inici, canvia les coordenades actuals per les d'inici. El mètode es crida quan es detecta que la gota ha arribat al terra (component Y igual o menor que zero). D'aquesta manera, es dóna la sensació d'un nombre infinit de gotes, ja que quan aquestes arriben al final, tornen a començar la seva caiguda des de la posició inicial.

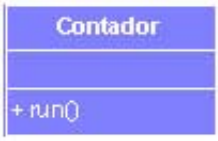
3.11.- Classe So

Aquesta classe s'encarrega d'executar els sons que es senten quan s'executa el joc. El constructor de la classe, carrega totes els arxius wav en memòria mitjançant el mètode carregaClip.
La resta de mètodes (excepte silenci), executen un so quan s'executen. El mètode silenci, atura l'execució del tots els sons que hi hagi en el moment de la crida.
El mètode camina, executa dos sons seguits. Per tal de que l'execució d'un so no comenci quan l'altre encara s'està executant, s'utilitza l'atribut booleà runningStep2. La resta d'atributs, són del tipus Clip, i serveixen per poder executar un arxiu wav una vegada aquest ha estat carregat.

3.12.- Classe Rellotge

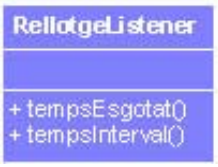
La classe Rellotge es l'encarregada de calcular el temps que falta per acabar la partida.
L'atribut temps, és un objecte de tipus Timer que cada segon executarà, en un nou fil, una instància de contador. L'atribut numIteracions indica el nombre de segons totals i per tant el nombre de vegades que s'haurà de crear un fil nou. L'atribut iteracioActual, indica el segon actual en que s'està.
El mètode parar atura l'execució de nous fila i per tant atura el contador de temps. El mètode addRellotgeListener afefeix en una llista un objecte que implementa la interfície RellotgeListener.

3.13.- Classe Contador



Aquesta classe és interna de la classe Rellotge, i quan s'executa ho fa en un fil nou. Cada execució d'aquesta classe implica un increment de l'atribut iteracioActual de la classe Rellotge i de l'execució del mètode tempsInterval de tots els objectes RellotgeListener que tingui la classe Rellotge en el Vector llistaListener. Si quan s'executa aquesta classe l'atribut iteracioActual no és més petit que numIteracions, s'atura el rellotge i es crida al mètode tempsEsgotat de tots els objectes RellotgeListener que tingui la classe Rellotge.

3.14.- Classe RellotgeListener

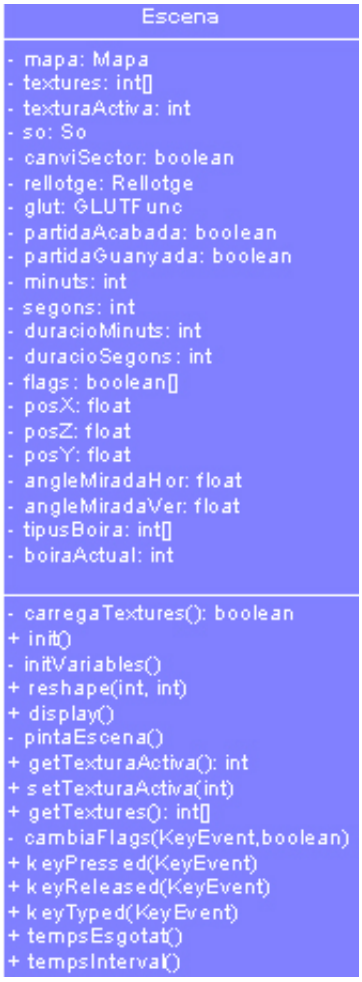


RellotgeListener és una interfície que es fa servir per notificar a les classes que l'implementin de cada segon que passa en la classe Rellotge (mètode tempsInterval) i de l'esgotament del temps marcat com a límit (mètode tempsEsgotat).

3.15.- Classe Main

Aquesta és la classe que s'executa quan s'inicia el programa. la seva funció és la de crear una finestra AWT, afegir-hi l'objecte escena i iniciar l'animació.
La finestra AWT capturarà els events de tancament de finestra.

3.16.- Classe Escena



La classe Escena és l'encarregada de pintar els gràfics per pantalla i de captar els events de teclat que l'usuari envia. Entre els seus atributs s'inclouen el rellotge per mesurar el temps de partida, el mapa on està carregada l'estructura del mapa, les textures que es faran servir en el programa, atributs booleans que indicaran els events de teclat que estan activats, variables que indiquen la posició de l'usari, el tipus de boira que s'ha de dibuixar i una variable de tipus So per executar els sons del joc.

El primer mètode que s'executa quan es crea l'objecte Escena és el mètode init. En ell, es carreguen en memòria totes les textures que s'utilitzaran i s'inicialitzen els primers paràmetres de l'aplicació (textures activades, tipus de model, característiques de la boira, etc). Al final d'aquest mètode, es fa una crida a initVariables, que s'encarrega d'inicialitzar les variables de la partida (so, rellotge, posició de l'usuari, sector que es pintarà, etc).

El mètode més important de la classe és el pintaEscena. En ell és on es fa tota la gestió de pintat per pantalla segons el valor dels atributs que tingui. El primer que fa, és comprovar que la partida no hagi finalitzat. Si és així, pinta per pantalla una pantalla segons s'hagi guanyat o s'hagi perdut la partida. Per pintar l'escena, el primer que es fa és rotar la matriu identitat per aconseguir l'efecte de mirada horitzontal i vertical. Després, es fa una translació al punt on està situat l'usuari. Amb aquesta disposició de la matriu, es pinta el sector on està l'usuari i la pluja (si aquesta està activada). Després del pintat, s'actualitzen les variables segons els events de teclat que estiguin activats (boira, curors, etc) i es comprova que la nova posició on està l'usuari sigui una posició vàlida. Al final del mètode, es pinta en la part inferior de la pantalla el temps que resta per acabar la partida.

El mètode cambiaFlags canvia l'estat d'una posició de l'array flags segons el valor booleà que se li passi al mètode. En l'array flags, és on es guarden els estats de la pulsació de les tecles. La posició zero, per exemple, pertany al valor de si està premuda o no la tecla RePag. Quan el programa detecta que s'ha premut una tecla, es crida a aquest mètode que comprova que la tecla sigui una de les que gestiona el programa per actualitzar-li el valor.
Els mètodes tempsEsgotat i tempsInterval són heredats de l'interfície RellotgeListener. Quan la classe Contador s'executa, crida a tempsInterval d'Escena per tal de que actualitzi el cronòmetre que es mostra per pantalla. Si el temps s'ha esgotat, Contador crida a tempsEsgotat d'Escena. Si s'executa aquest mètode voldrà dir que la partida s'ha perdut.

4.- ERRORS

Totes les proves que s'han efectuat han donat com a resultat que l'aplicació funciona correctament i que en teoria no s'haurien de produïr errors durant l'execució. Malgrat això, s'han detectat dues errades quan l'aplicació s'executa amb el sistema operatiu Windows (es desconeix si aquestes errades també es produeixen amb Linux):

- L'aplicació captura els events de teclat cada cop que l'usuari prem les tecles del cursor, AvPag, RePag, tecla P, tecla B i ESC. Quan la finestra de l'aplicació es minimitza i es torna a maximitzar, l'aplicació deixa de capturar els events de teclat. L'aplicació no dóna cap error d'execució, però tampoc permet interactuar amb el joc. L'única manera d'arregla el problema és tancant la finestra
- Tant si es guanya (es troba la sortida) com si es perd (s'esgota el temps), el joc mostra una pantalla final. Aquesta pantalla conté caracters escrits que a vegades no es dibuixen de forma parcial. Aleatòriament, poden haver algunes lletres que no estiguin dibuixades totalment.

COCLUSIONS

Els objectius que s'havien marcat al principi crec que s'han complert en la major part. Una vegada acabat el projecte puc dir que realitzar-lo ha estat més difícil del que m'esperava però que malgrat això m'he quedat amb ganes de continuar millorant el programa afegint-hi més funcionalitats i de corregir errors que no s'han tingut temps de corregir. Tot i que les textures donen un realisme força bò a les habitacions, la implementació de les llums hauria donat sens dubte més realisme a l'entorn del programa.
Alguns dels aspectes que trobo que s'haurien de millorar i afegir en la implementació actual són els següents:

- Definir millor les responsabilitats de cada objecte. Tal com es pot veure en el diagrama de classes, hi ha moltes classes que són utilitzades per moltes altres classes, la qual cosa fa que un canvi en una classe pugui afectar al funcionament de moltes altres. Definint millor les funcionalitats de cadascuna, es podria arribar a que les classes no depenguéssin tant unes de les altres i que els canvis que s'efectuéssin afectéssin només a una part concreta del programa.
- La classe Mapa té mètodes per crear cadascun dels sectors amb la informació de cada vètex, polígon, paret, intersecció i sector. El més òptim, seria definir un únic mètode per carregar tot el mapa, i que la informació d'aquest estigués emmagatzemada en un fitxer amb un format específic.
- Les textures de les portes algunes vegades queden un mica irreals degut a que no tenen profunditat. Seria interessant poder dibuixar marcs a les portes per tal de donar-lis un major realisme.
- Les habitacions no contenen cap objecte i per tant queden molt buides. Es podria implementar una classe Objecte que formaria part de cada sector, que podria dibuixar qualsevol objecte per pantalla (mobles, torxes, etc). Certament la implementació d'objectes en el sector faria canviar el mètode de comprovació de col.lisions.

BIBLIOGRAFIA

- Richard S. Wright Jr / Michael Sweet: Programación en OpenGL. Madrid, Anaya Multimedia
- Dave Shreiner: OpenGL reference manual. Addison, Wesley Publishing Company
- UOC: Mòduls didàctics d'Informàtica Gràfica I. Barcelona, Universitat Oberta de Catalunya
- Documentació d'internet
    - http://nehe.gamedev.net/
    - http://www.opengl.org/
    - http://java.sun.com/j2se/1.3/docs/api
    - http://www.jausoft.com
    - http://www.gametutorials.com
- Altres referències amb recursos gràfics o sons
    - http://www.greeneclipse.net/thewavfiles/sounds_e.html
    - http://www.members.tripod.com/~buggerluggs/ie/wav-dir184.htm
    - http://textures.forrest.cz
    - http://www.i-tex.de/gallery.htm
    - http://www.3dtotal.com
    - Joc per PC Half Life: Blue Shift