2013 - 2014

StreetJumper

STREET JUMPER

AITOR ARQUÉ ARNAIZ

UAB

PROJECTE Nº **5635 - 1**. IMPLEMENTACIÓ D'UN VIDEOJOC DE PLATAFORMES 2D AMB REALITAT AUGMENTADA I MODE MULTIJUGADOR.

Memòria del Projecte de final de carrera
Enginyeria en Informàtica
Realitzat per :
Aitor Arqué Arnaiz
i
dirigit per :
Ramon Grau Sala
Bellaterra , Juny 2014

ÍNDEX :

SUMARI :

Un dels principals problemes durant la carrera , és la gran varietat de temàtiques a tractar i la falata d'especialització en molts àmbits. Els Gràfics per Computador i els Videojocs són un exemple, hi ha un gran ventall de tècniques gràfiques aplicades a multitud de tecnologies (PC, dispositius mòbils, TV, etc...) , acompanyades de molts llenguatges de programació diferents com java o C++ entre d'altres.

La meva motivació durant aquest projecte és introduir-me en detall en el procés de desenvolupament d'un videojoc per dispositius mòbils amb sistemes operatius Android, aprenent d'aquesta manera tècniques de dibuix, llenguatges *d'scripting*, tècniques gràfiques avançades i realistes, i finalment, entendre el funcionament intern d'un videojoc a baix nivell. Durant el procès hi ha molts llenguatges de programació involucrats, desde java per la lògica del videojoc fins llenguatges de GPU's com GLSL (shaders) o Lua-Bind com a llenguatge d'*scripting* , és interesant aprendre amb detall aquests llenguatges donades les seves aplicacions al món laboral i la seva poca utilització durant la carrera d'Enginyeria Informàtica.

Per altre banda també ens trobem amb moltes classes de videojocs , tot i que els dispositius mòbils són tecnològicament "nous" i tenen un hardware limitat, la seva comercialització ha sigut tan ràpida i favorable que podem trobar tantes classificacions de videojocs com per plataformes PC o consola. Cada classe de videojoc té les seves peculiaritats i introdueix molta variabilitat a l'hora de desenvolupar el software o les fases de disseny, no és el mateix implementar un joc 2D que 3D, o encarat a primera persona com FPS (first – person - shooter) que vista en tercera persona (estratègia), etc... apareixen molts estils de joc el que dificulta encara més la generalització de les tècniques a utilitzar durant la implementació. Aquest projecte està encarat a realitzar un videojoc 2D i anomenat de "plataformes" , un estil de joc clàssic que ja va aparèixer cap als anys 80 a les primeres consoles. A mès s'introduiràn alguns aspectes més novedosos , com la realitat augmentada i la jugabilitat on-line (multiplayer) per intentar donar-li un estil més actual i atractiu per els usuaris finals.

1. INTRODUCCIÓ :

1.1 INTRODUCCIÓ MOTIVACIÓ I OBJECTIUS :

Els objectius del projecte són per una banda aprendre el procès de disseny i implementació d'un videojoc (incloent totes les parts involucrades), en el meu cas específic, de plataformes per sistemes operatius Android, i la motivació d'apendre a realizar aquestes tasques al detall. El procès de disseny implica documentar-se sobre l'estat de l'art dels videojocs , en general, per diferents plataformes (PC, Android, iOS, etc…) , els diferents objectius i estudiar tots els procesos/parts involucrades en el desenvolupament d'un videojoc (artístic, codificació, disseny, engine …).

1.2 ESTAT DE L'ART :

Estudiar l'estat de l'art del disseny d'un videojoc és una tasca complexa i extensa, per tant aquí comentare els punts més importants i essencials de l'estat de l'art i els procesos més crítics durant les diferents fases :

1.2.1 Realitat Aumentada : La realitat aumentada s'esta convertint avui dia en una tecnología comú en el món de les aplicacions mòvils, tot i que va començar èssent molt específica i cara farà uns deu anys. Els videojocs és l'àmbit on més s'ha integrat la realitat aumentada donat que són aplicacions idòneas per donar a conèixer la nova tecnología a les masses, i la indústria atrau i afecta a un nombre enorme de consumidors. Tot i el treball que s'ha dut a terme en aquest àmbit , encara és una tecnología que no s'ha consolidat , i gran part dels videojocs desenvolupats són imcomplerts o formen part d'àmbits educatius i d'investigació. Aquí es presenten alguns exemples de videojocs amb RA considerats d'entreteniment (com és el cas d'aquest projecte) [1] :

1.1 AR Defender : tower defender (iPhone)
1.2 Invizimals : animals haunter (play station)
1.3 Art of Defense : object tracking (smartphone)
1.4 AR Quotes : toss virtual rings (iPhone and 3D)

1.5 MyTown : find other players by GPS (iPhone)
1.6 AR Squash : tenis motion tracking (smartphone 3D)
1.7 Augmented Coliseum : robots battle (smartphone)
1.8 CurBall : ball obstacles (smartphone)
1.9 Butterfly Effect : Catching butterflies (smartphone)
1.10.ARBattleCommander : Strategy game (smartphone)
1.11.ARQuake : FPS game (smartphone)

Com es pot observar no hi ha gairabé cap joc de plataformes o obstacles amb realitat aumentada , i molts dels jocs existents están disponibles únicament per iPhone. Abans d'acabar aquesta part cal destacar les limitacions més greus de la realitat augmentada :

-Portabilitat : la baixa resol.lució , contrast i lluminància poden afectar molt a l'hora de visualitzar l'escena en ambients oberts (exterior).

-Seguiment i Cal.libració : Retards en el processament del moviment de la càmera poden afectar al seguiment d'algun objecte de l'escena a temps real (problema colateral amb el S.O que no sol estar preparat per aplicacions a temps real).

-Profunditat : La baixa resol.lució i calitat dels displays produeix que alguns objectes de l'escena apareixin a una distancia de profunditat diferent a la que hauria de ser.

-Acceptació Social : S'ha de tenir en compte que tots aquests conceptes canvien de manera "radical" el concepte de videojoc que hi había fins al moment , i encara s'han de resoldre alguns conflictes a nivel de software entre altres causes per a què la seva comercialització sigui àmplia i variada.

1.2.2. On – Line Gaming : El problema esencial (limitacions) de les arquitectures de xarxes per als videojocs depèn molt del tipus de joc, es a dir, en alguns entorns el problema d'estudi recau en la latencia, o en l'ample de banda , o en temps real , etc… Per altre banda les diferents arquitectures físiques de les xarxes (Ethernet, atm, token ring, FDDI, …) dificulta encara més la tasca d'implementar la jugabilitat online de l'aplicació. El suport d'aquests dispositius per accedir a la xarxa també és bastant limitat (Wireless, LAN, WIMAX, GPRS i Bluetooth) i els protocols que soprta consumeixen una cantitat de batería considerable , el que crea la necessitat de crear un software de transmissió de dades altament eficient i de baix consum. A continuación s'exposen les tres arquitectures conegudes i utilitzades fins al moment per qualsevol videojoc del metrcat :

- P2P : Utilitzada en alguns jocs , però no gaire generalitzada donats els següents inconvenients :

1. Cada dispositiu ha de correr el seu propi videojoc i la conexió entre dispositius s'implementa amb pas de missatges , el qual dificulta el control de latencia i per altre banda ocasiona delays importants sota infrastuctures Wireless amb baix ample de banda.

2. El dispositius mòvils tenen una capacitat de còmput limitada i aquesta arquitectura manté la CPU més ocupada.

3. Difícil d'implementar.

- Client / Server : Aquesta arquitectura és la preferida per totes les companyíes de videojocs , donat que és fácil d'implementar, fácil de mantener la integritat en el servidoe centralitzat i és més difícil de hackejar i chatejar. Adequada per jugabilitat masiva en xarxa (milers d'usuaris).

- Hybrid :Té en compte els avantatges de les dues arquitectures comentades anteriorment. Redueix el coll d'ampolla del pas de missatges , tot i que l'ample de banda requerit no és petit.

1.2.3. Motors Gràfics : El desenvolupament d'un videojoc complert és una tasca molt complexa i requereix d'un àmpli ventall de coneixements , per tant , durant el desenvolupament del projecte es donarà ús del motor gràfic 2D per Android

AndEngine donat que és lliure i adequat per realizar moltes de les tasques requerides en aquest projecte, com físiques, animacions, I.A. , Realitat Augmentada, etc… Tot i que es farà ús d'aquest motor es comenten a continuació alguns dels motors més innovadors del moment per aplicacions mòvils[10] :

- AndEngine :

  - Optimització per Android i compatibilitat amb la versió 1.6.
  - Mode SpitScreen.
  - Multijugador en xarxa.
  - Texturas vectorials (SVG).
  - Live Wallpaper
  - Posibilitat de MuiltTouch.
  - Motor de físiques amb Box2D
  - Posibilitat ' implementar realitat aumentada.

- LibGDX :

  - Shaders, vertex y soport ' OpenGL 1.0, 1.1 y 2.0 (només en Android 2.0).
  - Sistemas de párticulas.
  - Càrrega de so mp3, OGG i waw.
  - Càrrega de mapas TMX Map Editor.
  - Soport per ratolins, teclats i touch.
  - Soport del compàs, accelerometre i vibrador.
  - Soport d'events remots, per exemple interactuar amb un pc.
  - Físiques amb Box2D.

- Forget3D :

  - Càrrega de models MS3D, G3D (Glest) i MD2.
  - Textures de 8 y 24 bits.
  - Manager d'escenas.
  - Control de càmeras.
  - Sistema d'iluminació.
  - Soport de fonts.

…….. molts més.

2. VIABILITAT I PLANIFICACIÓ :

2.1 MERCAT AL QUE VA DIRIGIT :

Aquest videojoc és un joc d'entreteniment per plataformes mòvils , basat en jocs clàssics com el Mario , Rayman i d'altres amb noves modalitats com la possibilitat de jugar amb realitat augmentada i la jugabilitat on-line per intentar èsser més novedos i divertit. Es apte per tots els públics , el mercat que intenta abarcar es bastant ampli , per tant es pot dir que va dirigit a tothom , tot i que pot semblar més atractiu per a nens o adolescents.

2.2 MODEL DE DESENVOLUPAMENT DEL SOFTWARE :

Durant el procès de desenvolupament de qualsevol software , s'ha de mantenir una mateixa filosofía o paradigma a l'hora d'implementar el projecte gestionant les parts que el componen i posant objectius per mantenir el control del mateix.
A dia d'avui existeixen moltes filosofíes de desenvolupament del software ja estudiades i testejades incloses en els llibres d'Enginyeria del software :

1. Model en cascada : Marcada per les etapes del cicle de vida del software, on cada etapa ha d'esperar necessàriament a que finalitzi l'anterior.

2. Model en espiral : Les activitats no estàn fixades a priori , sinò que es fixen en funció de l'anàlisi de riscos que es realitza a cada iteració.

3. Model de prototips : S'inicien les iteracions amb els objectius globals i a cada iteració es detalla més.

El software vindrà fixat amb uns requisists inicials especificant exactament el que es vol aconseguir. Un problema comú és que aquests requeriments varíen al llarg del temps , motiu per el qual han de ser clars i detallats.
El model que es farà servir en aquest projecte serà el model en cascada, es pot veure el model a la figura 1 :

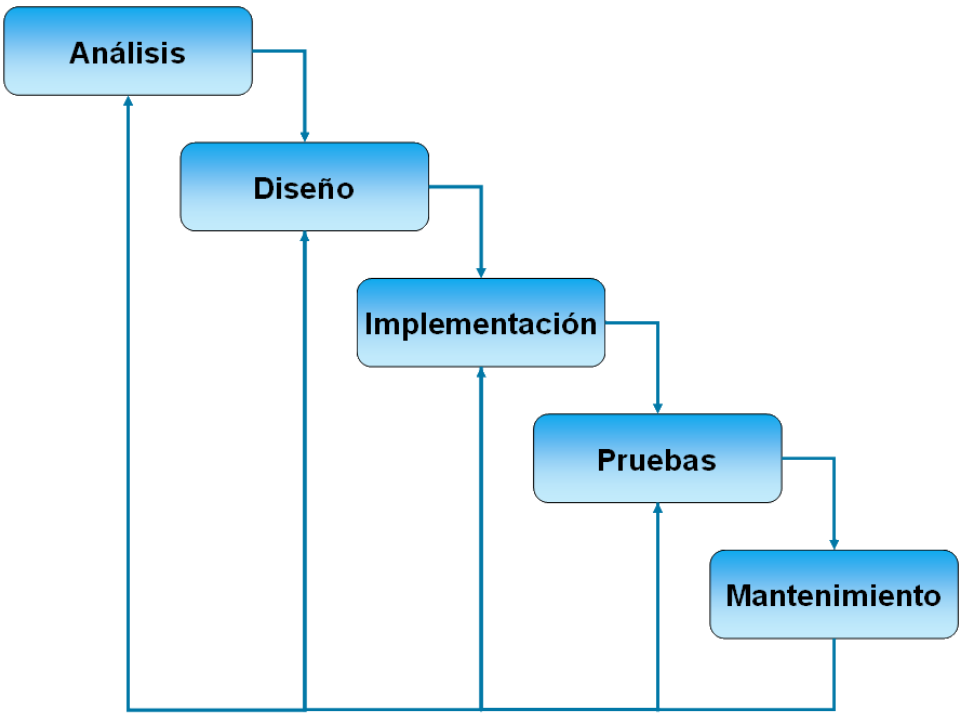


Figura 1 : Model Cascada

Requeriments: S'especifiquen tots els requeriments (funcionals i no funcionals) sobre el desenvolupament del software per a la correcta interpretació de què es vol i com es vol a l'aplicació.

Disseny: Es realitza el disseny de la nostra aplicació, estructures de dades i moviment de les dades en el nostre sistema.

Codificació i test unitari: Es traspassa el disseny de codi i es realitzen les proves bàsiques de funcionament dels diferents submòduls que componen l'aplicació.

Integració del sistema: Integració dels submòduls enllaçant les seves entrades i sortides per compondre el sistema global.

Operació i manteniment: "*Testing*" del sistema global per trobar el major nombre possible de fallades i així tornar a iterar el cicle des de qualsevol fase anterior millorant la qualitat final del software.

2.3 RECURSOS MATERIALS I COSTOS :

En aquest projecte distingirem els recursos materials en : recursos software i hardware.

1. Recursos Software :

Eclipse : Gratuït

Motor Gràfic AndEngine :

- Llicència LGPL : Gratuït

Eines de dibuix (artista) :

- Game Maker : entre 18$ i 500$
- Easy Button & Menu Maker : 30$
- AndEngine Level Editor : Gratuït*

2. Recursos Hardware :

Pc : intel i7 2600k 3.4 GHz , 8GBytes RAM i GTX 650 (Desenvolupament software)

- Utilitzant els material universitat : Taxes matrícula
- Adquirint de manera pròpia : 1000$

Dispositiu mòbil (smartphone) amb sistema android >= 2.2

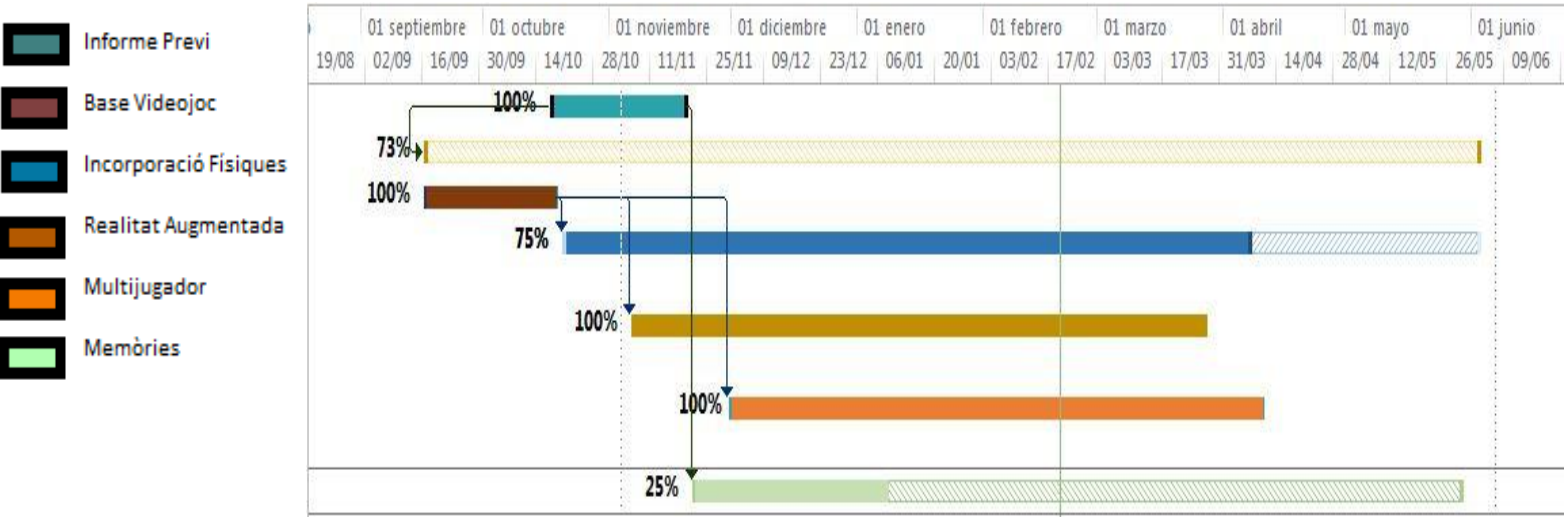
- Utilitzant material universitat : inviable
- Adquirint de manera pròpia : 100$

Els recursos que s'han hagut d'adquirir durant aquest projecte han sigut la llicència bàsica del game maker (només eina de dibuix) i un dispositiu mòbil amb un sistema Android igual o superior a la versió 2.2.

Costos = 18$ + 100$ = 118$

2.4 PLANIFICACIÓ TEMPORAL :

Per mostrar la planificació prevista de totes les tasques involucrades en aquest projecte, a continuació es mostra un diagrama de Gannt amb les tasques desglossades, el temps previst per a cada tasca i l'estat en el que es troba cada una d'elles.



Base Videojoc : Aquesta tasca es refereix a la implementació de les parts essencials de qualsevol joc per plataformes mòbils : renderització de escena splash , escena del menú principal, escena del menú d'opcions, pantalles d'espera (loading) per passar d'un menú a un altre , càrrega de fonts de texte etc...

Incorporació de físiques : Mòdul/s software encarregat de crear les físiques de l'escenari de joc (en general), per exemple : definir el concepte de gravetat a l'escena , definir cosos estàtics o dinàmics, registrar "conectors" de física entre cosos i escena etc...

Realitat Augmentada : Aquest mòdul serveix per poder carregar com a imatge de fons de l'escenari de joc (background) els frames capturats per la càmera del dispositiu mòbil.

Multijugador : Part corresponent a l'establiment de connexió entre dos dispositius mòbils a partir del protocol WiFi , i la correcta sincronització per mostrar els múltiples jugadors a "temps real" en els dos dispositius conectats.

2.5 VIABILITAT DEL PROJECTE :

Donat que aquest videojoc incorpora realitat augmentada el fa bastant diferent i "peculiar" respecte a altres videojocs del mercat per a tecnologíes mòvils, Com s'ha comentat a l'apartat 2 els videojocs que incorporen realitat augmentada són de l'àmbit educatiu o están inacabats i per altre banda no n'hi ha cap de plataformes el que fa aquesta aplicació diferent tenint en compte la cantitat masiva de videojocs ja existents. Un altre punt a favor és el fet de l'existència de motors gràfics lliures (open source) que faciliten i possibiliten el desenvolupament d'un videojoc a nivell individual.

3. DISSENY:

3.1 DISSENY D'ESCENES :

En aquest apartat s'analitzaràn tots els diferents escenaris que formen un videojoc , en general. Normalment un videojoc , independentment de la plataforma , es divideix en un subconjunt d'escenaris diferenciant per exemple l'escenari de joc de l'escenari del menú principal o el menú d'opcions. Els escenaris com el menú principal, opcions o càrrega es podrien considerar una *GUI* per a què l'usuari final pugui configurar algunes de les opcions del joc , o tingui la possibilitat de consultar els controls, etc... , en canvi, els escenaris de joc són aquells que permeten interactuar al jugador amb els gràfics o la *IA* creats entre d'altres. En aquest cas concret he decidit dividir el videojoc en 4 tipus d'escenes diferents : Splash scene (pantalla de càrrega o inici) , Main menú scene (menú principal) , Options scene (menú dòpcions) , Loading scene (canvis entre escenes) i Game scene (escenaris de joc).

Aquí s'explicaràn com són aquestes escenes i quina és la seva utilitat , més endavant al punt d'implementació , s'explica i s'analitza la seva implementació.

3.1.1 SPLASH SCENE :

Aquesta escena és la primera que es carrega al iniciar l'aplicació. El seu objectiu es mostrar algun logotip representatiu del videojoc ja sigui del motor gràfic utilitzat com és el cas d'aquest projecte o altres, normalment a l'inici es mostren diverses escenes "splash" seqüencialment amb l'objectiu de fer "marketing" mentre s'inicialitzen i es carreguen els menús.

L'escena creada per aquest videojoc té aquest aspecte i és la única "splash scene" existent :

Aquest logotip mostra el motor gràfic que s'ha utilitzat per implementar l'aplicació (*AndEngine*, més endavant s'analitza amb més detall).

3.1.2 MAIN MENU SCENE :

Aquesta escena es pot considerar l'escenari principal , o un dels més importants de qualsevol videojoc. En aquest normalment hi figuren les opcions més essencials i bàsiques de configuració del videojoc com poden ser la configuració dels gràfics, els controls de jugabilitat, subtítols, etc... acompanyat de submenús per ajustar altres camps més específics.

L'objectiu més important d'aquest menú és la facilitat per l'usuari final de configurar les opcions i començar o renaudar una "partida". És molt important que l'usuari (jugador) no tingui masses dificultats per entendre com procedir en aquest menú. Amb aquest objectiu s'ha dissenyat el menú principal d'aquest projecte , subdividint en tres submenús totes les opcions que es comenten a continuació (i una captura de pantalla per mostrar l'aspecte visual) :

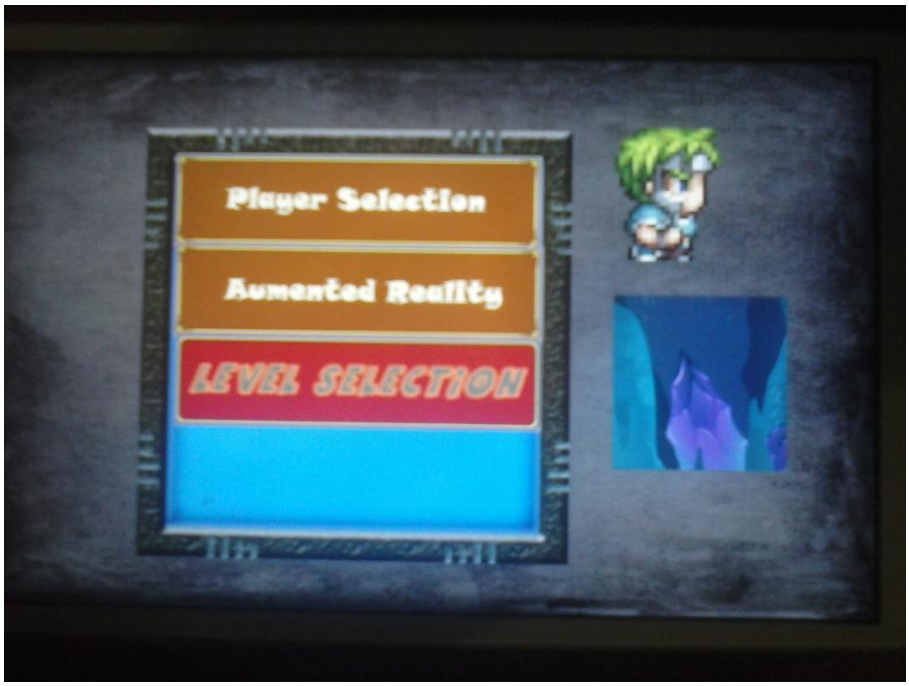
New Game : Aquesta opció carrega directament l'escenari de joc o "pantalla" escollit per defecte (o per l'usuari), començar nova partida.

Options : Aquesta opció desplega un nou escenari o menú on es poden configurar les opcions del videojoc. Al següent apartat es comenten els aspectes bàsics i la funcionalitat d'aquest menú.

Load Game : Aquesta opció carrega l'escenari de joc corresponent en l'últim estat "estable" en el que el va deixar l'usuari. La idea és carregar l'últim escenari de joc carregat anteriorment i no tornar a començar desde el principi cada vegada.

3.1.3 OPTIONS SCENE :

Menú on es pot escollir a partir de tres botons : el jugador o player , el nivell o pantalla i activar/desactivar la realitat augmentada. La idea d'aquest menú també és la mateixa , és a dir , les opcions han de ser intuïtives i fàcils de configurar per l'usuari final. A continuació es mostra l'aspecte d'aquest escenari :



Com es pot observar a la captura de pantalla corresponent del menú d'opcions, hi ha tres botons que possibiliten el canvi d'alguns aspectes del joc : la primera opció (selecció de player) canvia el jugador mostrant la seva textura (shape) al costat dret del botó. La segona opció (realitat augmentada) modifica una variable d'estat activant o desactivant la realitat augmentada encarregada de mostrar com a background el surface view de la càmera del dispositiu mòbil. La tercera i última opció d'aquest menú (selecció de nivell) possibilita a l'usuari escollir de manera directa el nivell /pantalla que es vol carregar mostrant a la part dreta del menú un petit requadre amb una imatge que intenta èsser representativa de l'escenari que es vol carregar.

3.1.4 LOADING SCENE :

L'objectiu d'aquesta escena és mostrar un missatge de càrrega clàssic durant el procès de transició d'un escenari a un altre, per exemple, mentre es descarrega el menú principal i es carrega la primera pantalla, entre pantalles , etc... l'aspecte visual és molt sencill : un fons negre amb un missatge centrat sempre a la càmera.

3.1.5 GAME SCENE/S :

Conjunt d'escenes que formen el videojoc en sí, és una seqüència d'escenaris (o vulgarment pantalles) on l'usuari pot interactuar amb el medi creat i ha de superar una sèrie d'objectius marcats per poder finalitzar una "pantalla" i passar a la següent.
En el cas d'aquest projecte s'han dissenyat quatre escenaris amb un conjunt de plataformes i obstacles que intenten apropar-se a un videojoc clàssic de plataformes.

Per a dur a terme el disseny d'una escena de joc (en aquest cas *2D* i de plataformes) es necessari distingir principalment 4 fases :

1. Disseny del "background" : Aquesta part és la més simple, s'ha de dissenyar (dibuixar) el fons o paisatge sobre el qual posteriorment s'afegiràn les plataformes o objectes que interactuaran amb el jugador. No és més que una textura sobre la qual es desplaça la càmera mostrant aquest paisatge de manera parcial i dinàmica.

2. Disseny del "foreground" : Tots aquells objectes sobreposats al background amb els quals pot interactuar el jugador. Per exemple : plataformes, obstacles, el propi jugador, monedes , etc... per a aquesta fase és recomenable utililitzar eines de disseny de nivells ja siguin en *xml* o *tmx*, a l'apartat 3.3 s'explica amb més detall l'ús d'aquestes eines.

3. Disseny dels models : Aquesta part és la més complexa, durant aquesta fase s'han de dibuixar a mà (amb eines de dibuix vectorial o bé el conegut com píxel art) tots els objectes que formes l'escenari. A l'apartat 3.2 s'explica amb detall tot aquest procès.

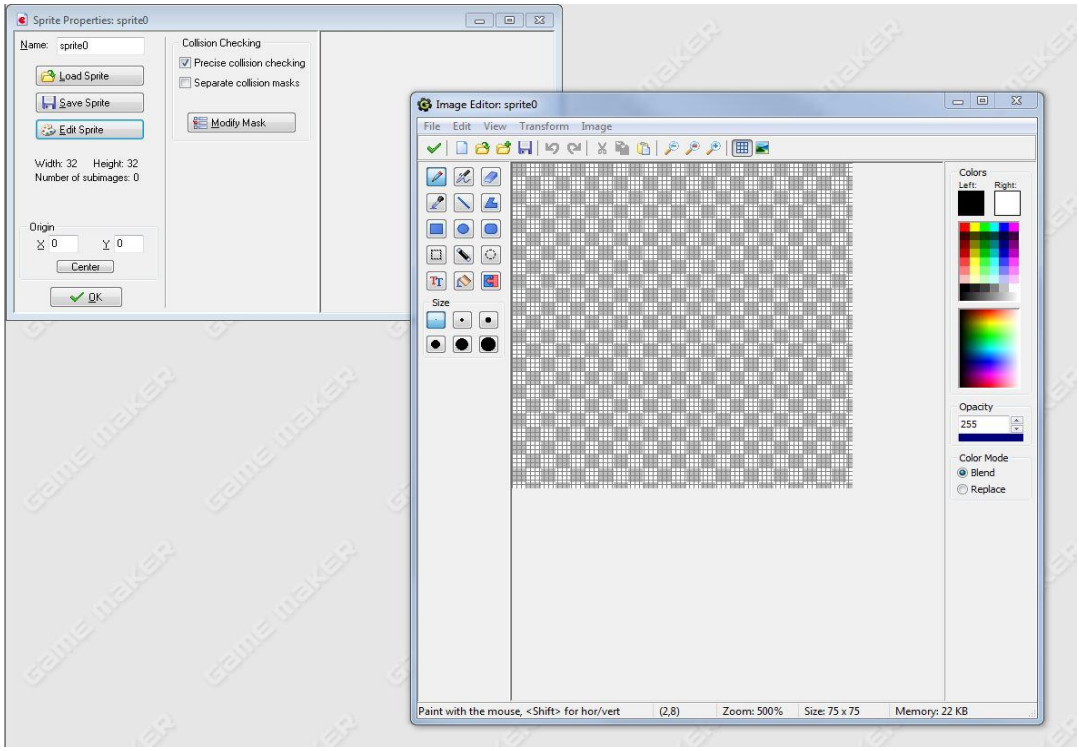
4. Disseny de la jugabilitat : Aquesta part pot semblar de no gaire rellevància però és molt important. S'ha de pensar i analitzar com l'usuari ha d'interactuar amb l'escenari a partir dels dispositius d'entrada/sortida ("els controls"). Si la jugabilitat és complexa l'usuari necessita d'un petit tutorial o procès d'aprenentage per a la correcta manipulació de l'aplicació i aquest fet pot suposar certa pèrdua d'interès per part de l'usuari, en canvi si la jugabilitat és massa sencillla l'usuari pot trobar l'aplicació poc interesant.

3.2 DISSENY DE MODELS GRÀFICS :

El disseny dels models és una tasca complexa que requereix d'un aprenentatge i uns coneixements que dependràn del nivell de qualitat que sigui necessari assolir. Hi han multitud d'eines de disseny gràfic , fet que dificulta encara més el procès de disseny donat que habitualment un àmbit de disseny està lligat a un conjunt d'eines específiques, per tant també és necessari aprendre a utilitzar aquestes eines.

En el cas d'aquest projecte el disseny de models es simplifica molt donat que és un joc *2D*, fet que facilita l'aprenentatge de les eines (són més sencilles, p.e gimp vs 3dsmax) i el dibuix dels "sprites". És important remarcar que durant el procès de desenvolupament d'un videojoc real , el disseny de models és una tasca dels dissenyadors gràfics i no dels programadors/codificadors , en aquest cas no ha sigut així a causa de falta de recursos al ser un projecte individual .En aquesta part s'exposen les eines utilitzades pel disseny gràfic dels models d'aquest projecte, i quin procès ("fluxe") de disseny s'ha seguit :

Game Maker : És una eina molt complerta amb varis nivells de llicència, i amb la seva totalitat es pot crear un videojoc complert (inclòs el disseny i molt més : físiques, animacions, etc...). Per a aquest joc s'ha usat la llicència bàsica que només permet utilitzar els mòduls de dibuix ("sprite" i "background") , és una eina de dibuix classificada com "píxel art" perquè la creació dels objectes/textures es dibuixen desde zero a partir de pintar píxels lògics en un mapa de píxels d'un tamany donat. Les avantatges d'aquesta aplicació són : sencill d'utilitzar , no requereix d'un aprenentatge difícil , i compatibilitat amb molts formats d'imatge tot i que a contrabanda el nivell de qualitat dels models que es pot aconseguir no és gaire alt. A continuació es mostra la *GUI* principal d'aquesta eina de disseny :



Com es pot observar a la figura és una eina molt bàsica (dibuix de línies, cercles, rectangles i llapis per pintar), amb la qual no es poden crear models massa complexes. Si es vol dibuixar una textura/objecte complicat és recomanable dibuixar les parts individualment i posteriorment col.locar-les desde el videojoc en les posicions corresponents per obtenir la textura final. Per aquest conjunt de raons s'ha utilitzat aquesta eina per dibuixar i crear els models del "foreground" dels escenaris i per als botons dels menús (principal i opcions) donat que són objectes petits i sencills que no requereixen de eines gaire complexes.

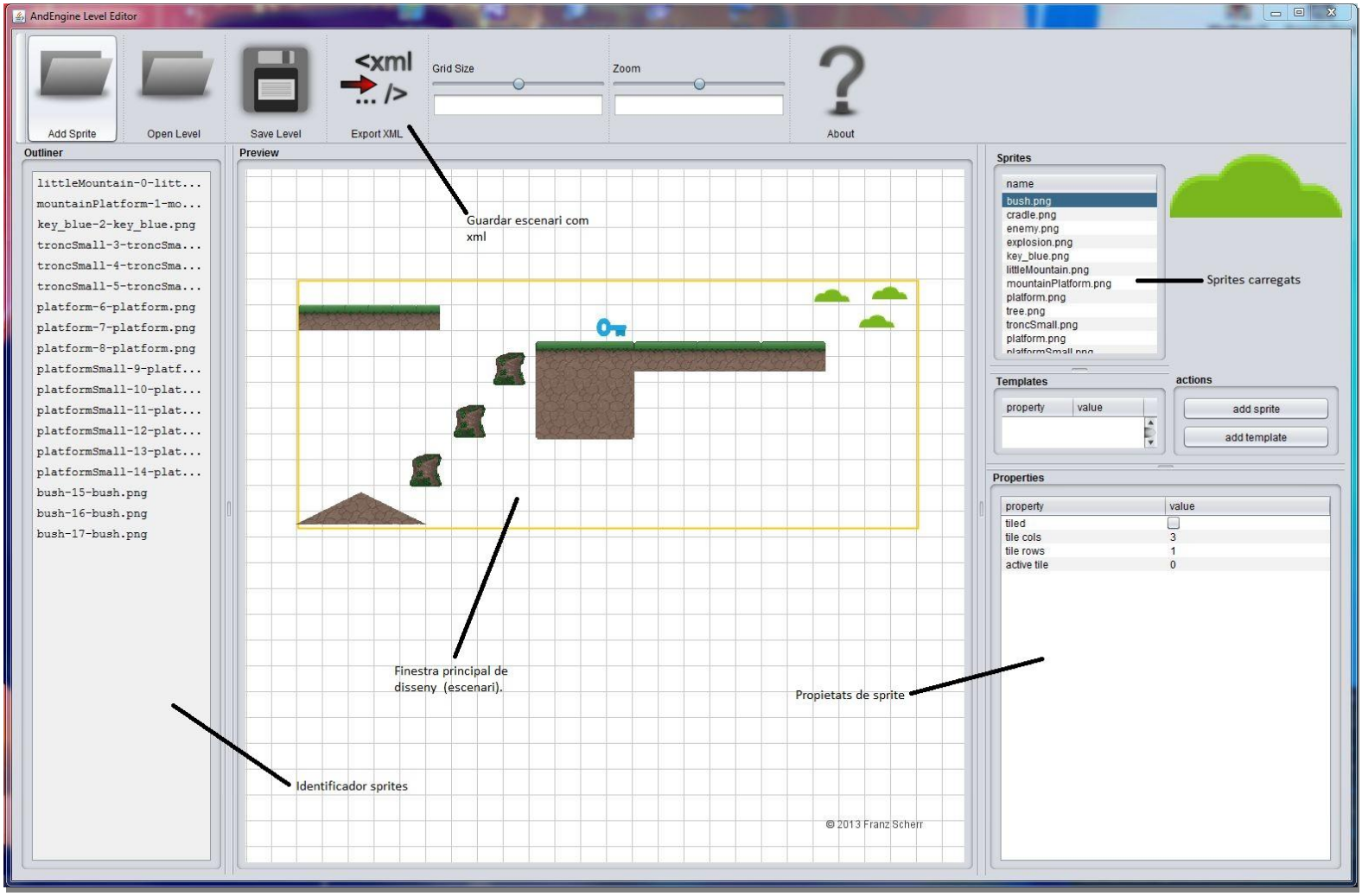
Tiled : Aquest software de disseny és conceptualment molt diferent al game maker , però amb les mateixes finalitats i utilitats. La diferència essencial amb l'eina anterior és que aquesta es capaç de carregar un conjunt d'objectes predefinits (mapa d'objectes) i el procès de dibuix es basa en formar un model a partir d'aquests blocs carregats prèviament , facilitant la tasca de disseny. Per aquest motiu en aquest projecte s'ha decidit utilitzar aquesta eina per al disseny del background dels escenaris, el muntatge per blocs accelera el disseny de les imatges de gran tamany i varietat d'objectes i colors com es el cas del background.

(Alguns dels models creats per als escenaris es mostren a l'annexe d'aquest document).

3.3 DISSENY DE NIVELLS (LEVEL EDITOR) :

El procès de dissenyar i crear els nivells (o "pantalles") és una tasca metòdica , laboriosa i requereix de certa imaginació, sobretot molta experiència per a aconseguir crear un escenari de joc divertit i atractiu per a l'usuari final.
En general existeixen eines d'ajuda automatitzada per a dissenyar els nivells d'un videojoc, en el cas concret dels videojocs *2D* clàssics els escenaris es generen en *xml* a partir de editors de nivell (Level Editors). La idea es poder col.locar els *sprites* de l'escenari de manera gràfica a través d'una petita finestra o graella per facilitar la creativitat i el procès de muntatge de l'escenari a mès d'accelerar la seva implementació generant de manera automàtica el *xml* final.

Per a desenvolupar aquest projecte s'ha adquirit de manera lliure un petit editor de nivells *xml* per a AndEngine molt bàsic i s'ha modificat per a intruduïr noves funcionalitats que faciliten més la gestió dels escenaris, com un projecte independent desenvolupat en NetBeans. Aqui es mostra una captura de pantalla on es pot veure quines funcionalitats pot oferir aquesta eina :



És un editor molt simple i visualment no gaire atractiu , però facilita enormement la tasca de muntatge de nivells (escenaris) en format *xml* fent el seu procès menys laboriós i més divertit.

3.4 LLENGUATGES DE PROGRAMACIÓ :

Durant el procès d'implementació del videojoc s'ha donat ús de múltiples llenguatges de programació depenent dels objectius , les necessitats i les tècniques gràfiques. Aquí es presenten els llenguatges utilitzats i les seves funcions :

1. Java : Llenguatge predominant , al ser una aplicació per sistemes operatius Android casi tota l'aplicació està escrita en Java. Tota la infrastuctura, lógica i base del videojoc està construida en aquest llenguatge.

2. C++ : Llenguatge utilitzat per realitzar alguns càlculs complexes que necessiten executar-se molt ràpid i de manera molt eficient. Principalment són petits blocs de codi molt localitzats que implementen alguna funcionalitat com per exemple , algunes col·lisions.

3. GLSL : Llenguatge de GPU's propi d'openGL utilitzat per als shaders introduïts en alguns escenaris com "*water shader"* i *"radial blur shading"*.

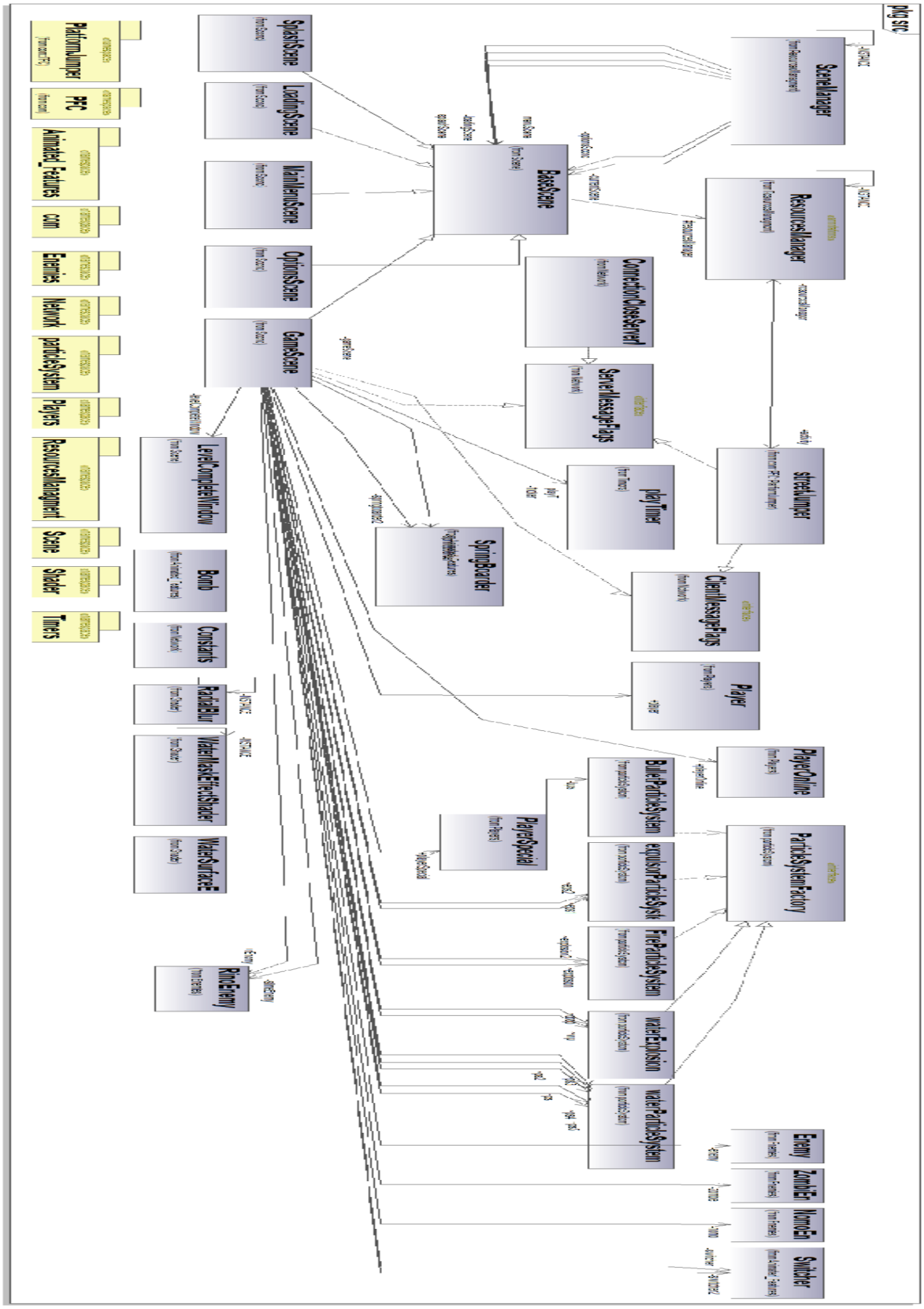
4. OpenGL natiu : Utilitzat per comunicar l'aplicació principal amb els shaders (definir les variables uniform , les varying, passar paràmetres desde fora , etc...). A més s'ha de tenir en compte que el motor gràfic utilitzat (AndEngine) està implementat en openGL 2.0.

5. XML : Llenguatge essencial , tots els nivells (escenaris) estàn escrits en xml generat a partir d'un editor de nivells.

3.5 DIAGRAMA UML (DISSENY DE L'APLICACIÓ) :

A continuació es mostra una imatge amb el diagrama de classes global de l'aplicació on bàsicament només es pot veure el nom de les classes i les seves interrelacions. S'ha de tenir en compte que es un diagrama complexe amb moltes classes i moltes relacions entre classes (inclòs de diferents paquets) , fet que dificulta poder col.locar tot el diagrama en una pàgina de manera que es pugui lleguir i entendre. És per aquest motiu que a l'annex hi figuren els diagrames de classes parcials (per paquet) mes complerts i llegibles on si que es pot veure cada classe quins mètodes conté i quines relacions existeixen entre les demès classes del mateix paquet.

La idea d'aquest diagrama de classes global es mostrar l'estructura general del software i les dependències existents entre els mòduls.



4. IMPLEMENTACIÓ:

4.1 ANDENGINE : EINA DE DESENVOLUPAMENT :

La implementació d'un videojoc és una tasca molt complexe gairabé impossible d'assolir de manera individual, motiu per el qual és necessari utilitzar un motor gràfic (o llibreries) com a eina d'ajuda per a tota la implementació del joc : gràfics (renderització de textures , animacions , sistemes de partícules , shaders , etc…) , Intel.ligència artificial , network (establiment de conexió , pas de missatges , etc…) i molt més.

Per a aquest cas concret s'ha decidit utilitzar AndEngine com a motor gràfic , és una eina gratuïta per sistemes operatius Android que només necessita la màquina virtual Dalvik i les llibreries d'openGL, presents a tots els sistemes, tot i la seva facilitat d'ús és una eina de programació de baix nivell i té algunes limitacions importants. A continuació es mostren les avantatges i desavantatges observades durant el desenvolupament de l'aplicació :

Avantatges :

- Aprenentatge ràpid i simple.

- És lliure (open source)

- Està ben documentat i hi ha molts exemples d'ús.

- Encara està en desenvolupament.

Desavantatges :

- Mala gestió de memòria.

- Si es carreguen moltes textures (o partícules, etc...) en un escenari el consum de recursos hardware i el consum energètic es disparen de manera exponencial , a mès es produeixen errors interns de mala gestió per part del motor.

- Programació de baix nivell , poc abstracte.

- poc portable (segons el dispositiu o sistema l'aplicació produeix fallades diferents) i inestable (té sortides inesperades).

https://github.com/nicolasgramlich/AndEngine/tree/GLES2-AnchorCenter

4.2 IMPLEMENTACIÓ DELS ESCENARIS :

El procès d'implementació dels escenaris consta de moltes etapes (com es comenta a l'apartat de disseny d'escenaris) , aquí s'exposaràn totes les tècniques gràfiques i algoritmes que s'han utilitzat en cada un dels cinc escenaris deixant de banda el disseny de models. La part mès laboriosa de la implementació són els escenaris de joc tot i que els menús d'usuari també són complexes de realitzar.

En aquesta part del document s'explica com s'han implementat desde les escenes de càrrega fins als escenaris de joc : algoritmes per crear efectes (p.e. shaders o partícules) , lógica de funcionament , transició entre escenaris , etc…

4.2.1 IMPLEMENTACIÓ SPLASH SCENE :

L'escena d'inici de l'aplicació, és la primera "pantalla" que es mostra al iniciar el joc. L'objectiu és mostrar algún sprite mentre es carrega el menú principal d'usuari, la seva implementació és molt sencilla , consta d'un fons negre amb un logotip del motor gràfic (AndEngine) centrat a la càmera. Aquesta escena no contè algoritmes ni tècniques de cap mena , es col.loca un background (més o menys complexe) i un sprite sobre aquest background.
Donat que és una escena introductòria i temporal ha de mostrar-se només durant un petit període de temps corresponent a la càrrega del menú principal, per tant es controla amb un temporitzador software que mostra l'splash scene durant 2 segons (temps suficient per càrregar les textures del menú principal).

4.2.2 IMPLEMENTACIÓ DEL MENÚ PRINCIPAL :

L'escena principal , posterior a *l'splash* scene, està implementada de la següent manera :

1. Es genera un background de 800x480 (tamany de la càmera), el resultat és una imatge de fons on posteriorment s'afegiràn *ítems*.

2. Afegir les textures dels botons sobre el background generat anteriorment. Les funcionalitats que ofereixen els botons són : *Mew Game* , *Load Game*, *Options* i *Online Options*.

Cada vegada que l'usuari pulsa un item (*button*) s'executa el codi corresponent a la funcionalitat que tè associada el ítem (p.e. *New Game* inicía una nova partida). Per altre banda per a què l'usuari pugui veure visualment qun ítem ha presionat es fa un escalat al botò corresponent, és a dir, mentre l'usuari mantè el botò presionat es mostra un escalat de 1.5 i quan ja no hi ha contacte es mostra en el seu tamany inicial.

4.2.3 IMPLEMENTACIÓ DEL MENÚ D'OPCIONS :

El menú d'opcions és un submenú del menú principal , quan l'usuari presiona el botò options del menú principal s'obre un nou menú on es poden configurar totes les opcions del joc, la implementació és exactament igual que el menú principal , un background que conté ítems. Les opcions configurables són les següents : selección de jugador , selecció d'escenari, activar/desactivar shaders, activar/desactivar partícules, activar/desactivar realitat augmentada.

Al presionar qualsevol dels botons d'opcions es fa un escalat (igual que a tots els menús), i en aquest cas es mostra un sprite (textura) a la dreta del botò presionat mostrant la opció seleccionada (p.e.si es presiona selecció del jugador es mostra la textura del jugador seleccionat seqüencialment). Per altre banda també es guarda l'estat del menú , és a dir, cada vegada que es surt del menú , es realitza una partida i es torna altre vegada al menú d'opcions es mostren totes les opcions seleccionades tal i com van quedar la última vegada.

4.2.4 IMPLEMENTACIÓ DELS ESCENARIS DE JOC :

La implementació dels escenaris de joc és la part més complexe i laboriosa de tot el videojoc. Per una banda està la part de disseny comentada a l'apartat 3.1 i per altre banda la implementació de la lógica , la I.A., la rendereització de gràfics, shaders, partícules, etc… de tots i cada un dels escenaris. A continuació s'exposen els mètodes o algoritmes més importants utilitzats a cada escenari (pantalla)

4.2.4.1 ESCENARI 1 :

Aquest escenari és el primer i està ambientat en un castell amb neu. Per pasar al nivell següent es necessari agafar unes claus per desbloquejar unes portes sense colisionar amb més de tres obstacles.
La implementació de la llògica és exactament igual per a tots els escenaris : per una banda hi ha definida una física global (gravetat) i una física individual (cada objecte pot tenir propietats físiques diferents com pes , inercia etc…), i per altre banda per calcular les colisions entre els objectes de l'escenari es fan servir envolvents (caixes que contenen objectes), per tant el que s'ha de fer és calcular les colisions entre caixes i no entre objectes facilitant molt la implementació de les colisions :

Si (*player colisiona amb plataforma_1*) **llavors**

    Operacions

Sino Si (*player colisiona amb plataforma_2*) **llavors**

    Operacions

etc…

Els algoritmes més complexes d'aquest escenari són : Shaders per simular aigua en moviment i sistemes de partícules per generar neu i explosions.Els shaders están

implementats en GLSL (llenguatge de GPU's d'openGL) i l'objectiu es aplicar una trigonometría sencilla a la textura de l'aigua per generar moviment , amb l'inconvenient de la capicitat de còmput afegida que això suposa per un dispositiu móvil (a l'anex 6.2 s'explica aquest algoritme al detall). Les partícules s'implementen amb un mòdul d'ajuda del motor gràfic que ja es capaç de generar sistemes de partícules i només s'han de configurar els paràmetres per obtener els resultats esperats. L'objectiu dels sistemes de partícules en aquest escenari són : crear l'efecte visual de pluja de neu i generar explosions cada vegada que el jugador colisiona amb una "mina".

4.2.4.2 ESCENARI 2 :

Aquest escenari de joc està ambientat al desert i és el més llarg de tots, l'objectiu és desbloquejar unes palanques i tornar a la porta d'inici. Com s'ha comentat en el cas anterior la implementació de la lógica de joc és la mateixa en tots els escenaris (físiques, colisions, etc…), els algoritmes diferencials d'aquest escenari són les colisions especials i els sistemes de partícules. En aquest escenari quan el jugador colisiona amb unes plataformes "especials" les plataformes cauen de maneres diferents sobre aigua estàtica provocant l'efecte de "chapuzón" amb sistemes de partícules.
Finalment hi ha un jugador intel.ligent que es mou en mode patrulla en una zona concreta de l'escenari esperant que el jugador principal s'acosti per disparar , aquest efectes visuals també són generats amb sistemes de partícules.

4.2.4.3 ESCENARI 3 :

El tercer escenari està ambientat en un bosc on l'objectiu és el mateix que al primer escenari, adquirir dues claus i obrir una porta sense colisionar amb més de tres obstacles. En aquest escenari no hi han algoritmes diferents als anteriors, es simula pluja de foc amb sistemes de partícules i pel reste és exactament igual al reste d'escenaris.

4.2.4.4 ESCENARI 4 :

Aquesta pantalla està ambientada en una cova i els objectius són dos : adquirir una clau per desbloquejar una palanca i matar a un enemic.
El que diferencia aquest escenari del reste és la funcionalitat afegida de poder llançar granades , aquesta funcionalitat s'implementa col.locant una textura amb forma de bomba a la part esquerre de la càmera i una sèrie d'objectes amb textura de granada a l'escenari de tal manera que cada cop que el jugador agafa un objecte "granada" , apretant la textura de la càmera es genera un efecte de llançament de l'objecte amb un angle i velocitat determinats. Com al reste d'escenaris quí també s'implementen sistemes de partícules per simular polen d'una sèrie de plantes que hi han col.locades durant l'escenari.

4.2.4.5 ESCENARI 5 :

L'últim escenari de joc, ambientat en un camp d'aviació on les plataformes són petits avions, l'objectiu és adquirir tres medalles (or, plata i bronze) per a veure un missatge de joc

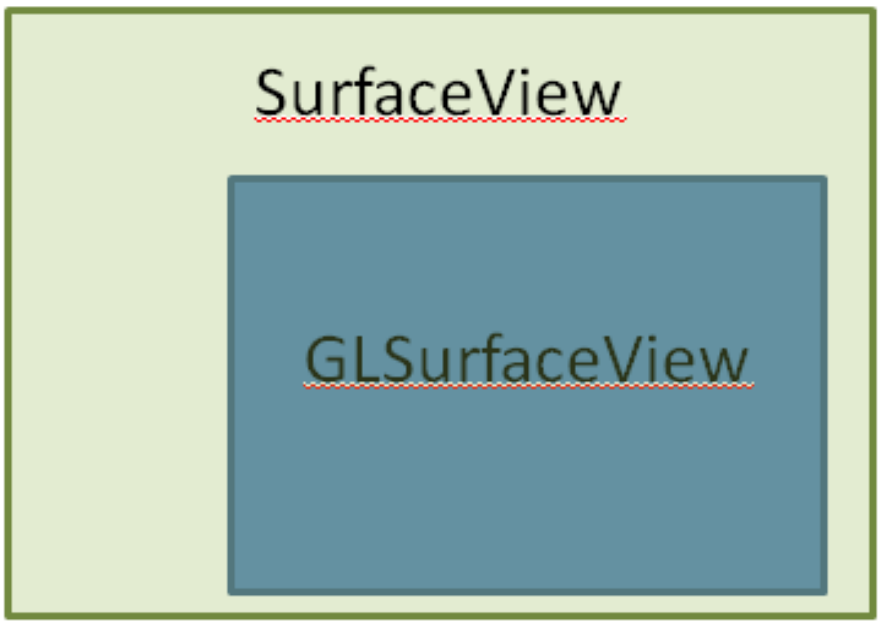
finalitzat. Aquest escenari no contè cap algoritme o técnica gràfica en concret, és el més simple de tots.

A tots els escenaris hi ha música de fons i es produeix un so cada vegada que el jugador salta o colisiona amb algún obstacle. També hi han monedes d'or , plata i bronze i en funció de la cantitat adquirida pel jugador es passa de nivel amb una , dos o tres medalles.

Finalment també s'ha implementat un screen de final d'escenari que contè un background transparent amb les estrelles corresponents i un HUD del jugador (vides, temps, puntuació, etc…).

4.3 IMPLEMENTACIÓ REALITAT AUGMENTADA :

El mòdul de realitat augmentada és una de les parts més potents i complexes de l'aplicació. L'objectiu d'aquest mòdul és incorporar la possiblitat de col.locar com a background desls escenaris les imatges capturades per la càmera del dispositiu móvil. Per a implementar aquesta part s'ha utilitzat una extensió del motor gràfic (AndEngine) anomentat *AugmentedRealityExtension* que ha facilitat enormement la tasca d'integrar la realitat augmentada en el videojoc.
Per a poder visualitzar les imatges capturades per la càmera del smartphone s'ha de crear un *Surface view* d'android per poder col.locar sobre aquesta estructura les textures associades a cada escenari. S'ha de tenir en compte el fet que no es pot interactuar amb la realitat augmentada (background), només es pot visualitzar donat que per fer posible la interacció amb el background s'hauria d'implementar també un mòdul de visió encarregat de segmentar l'escena en objectes i calcular profunditats i posicions entre aquests per convertir la imatge del background en un conjunt d'objectes amb els que podría interactuar el jugador igual que amb un altre objecte definit a l'escenari. A continuació es mostra una imatge a mode d'exemple que intenta mostrar la idea de la realitat augmentada implementada en aquesta aplicació i finalment s'exposa el procès d'implementació d'aquest mòdul amb els pasos seguits :



El primer problema a resoldre és el que s'anomena "Surface View", per poder mostrar les imatges (frames) capturades per la càmera del dispositiu móvil en el "background" dels escenaris s'ha de crear aquesta estructura, és important tenir en compte que la renderització dels frames capturats per la càmera en pantalla s'implementa en openGL natiu en GLSurfaceView en el cas d'aquesta aplicació (GLSurfaceView no està accelerat per hardware). Les imatges capturades desde la càmera s'han de col.locar/mostrar al GLSurfaceView amb les següents característiques :

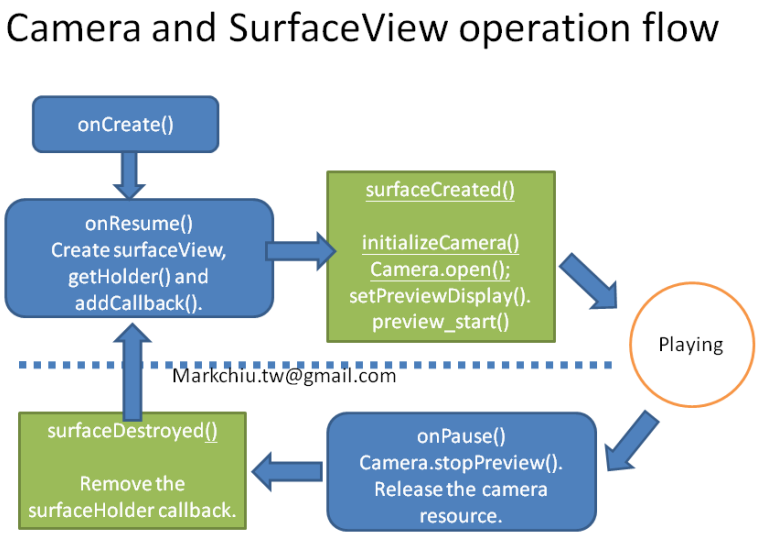
1. comunicar openGL ES amb el sistema de display

2. Permetre a openGL ES integrar-se amb el cicle de vida "normal" d'una activitat d'Android

3. Definir un format de píxel concret pel frame buffer

4. Crear i gestionar una rendereització paral.lela utilitzant threads

A continuació es mostra el fluxe de execució d'una aplicació Android amb un GLSurfaceView. La idea és sencilla , El primer pas és crear l'estructura (mètode onCreate()) , un cop creat s'inicialitza i s'obre la càmera del dispositiu i s'envíen els frames al frame buffer del sistema operatiu per mostrar en pantalla les imatges capturades.
Mentre s'executa l'aplicació es poden produir alguns events que modifiquin una mica el fluxe, per exemple : es pot parar o renaudar la captura d'imatges, es pot destruir etc… tot i que el més normal en totes les aplicacions és mantenir el GLSurfaceView durant tota l'execució de l'aplicació.



En el cas particular d'aquest videojoc hi han dues opcions o modalitats de joc : realitat augmentada o mode normal. Per a poder implementar les dues modalitats s'ha implementat de la següent manera :

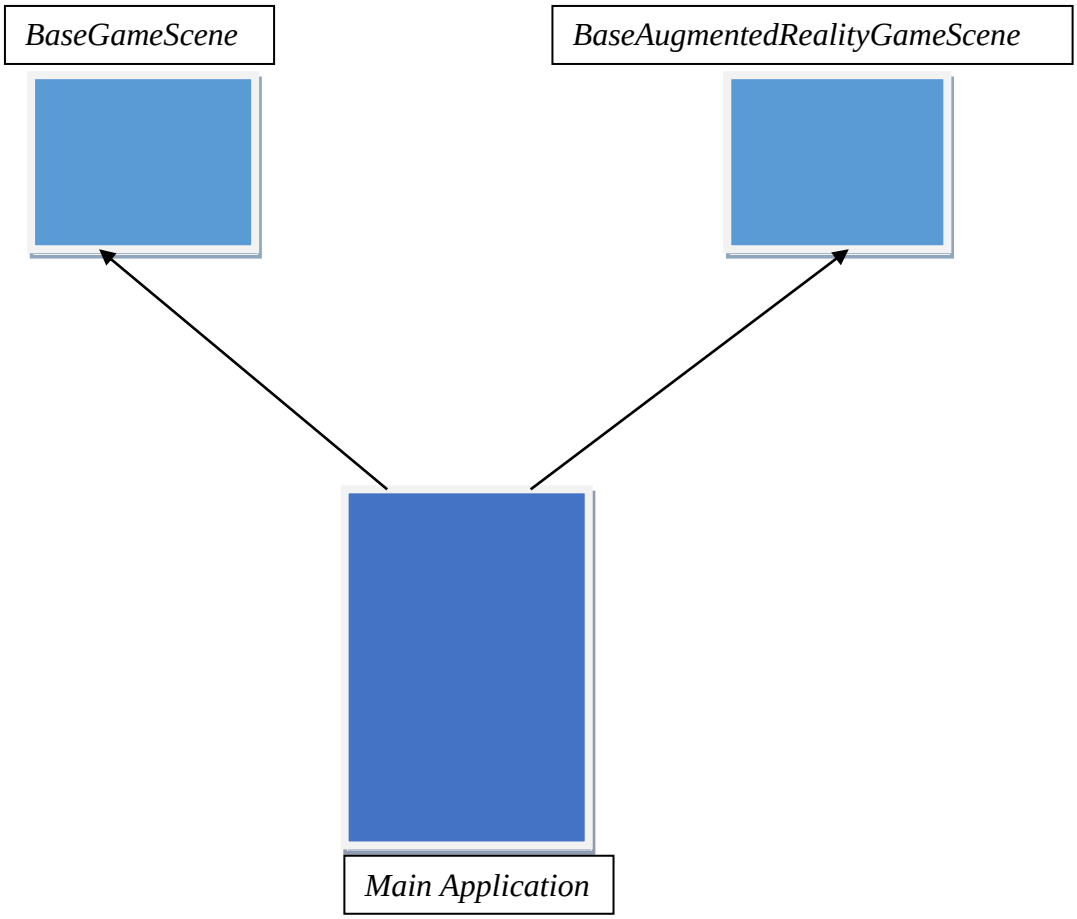
Si *(configuració per defecte* ***o*** *no RA)* ***llavors***

Crear un GLSurfaceView
Mostrar frames de la càmera sobre el GLSurfaceView
Col.locar el background 2D (sprite) sobre els frames de la càmera

Sinó si *(RA)* ***llavors***

Crear GLSurfaceView
Assignar 100% transparencia al background 2D (sprite) -→ (present però no visible)
Mostrar frames de la càmera sobre el GLSurfaceView

La idea és sempre mostrar els frames capturats desde la càmera i segons la opció/modalitat assignadaes es col.loca sobre aquests frames un sprite (o background) 2D amb o sense transparencia per èsser visible o no visible per l'usuari.
És important analitzar i extraure algunes conclusions d'aquesta manera d'implementar, és a dir, la manera més correcte d'implementar aquesta funcionalitat sería utilitzant el concepte de multiherència (no present en el lleunguatge Java) possiblitant així començar l'execució de l'aplicació heredant d'una activitat amb RA o sense RA segons l'opció/modalitat escollida:



BaseGameScene i *BaseAugmentedRealityGame* són dues clases del motor AndEngine ja definides, l'objectiu de la multiherència és separar les dues modalitats en dues clases diferents : segons la modalitat escollida per l'usuari s'hereda d'una activitat o d'un altre segons si volem tenir realitat augmentada o no. A causa que el compilador de Java no permet la multiherència s'ha implementat amb la metodología mostrada anteriorment.

4.4 IMPLEMENTACIÓ DE LA "NETWORK ":

El software encarregat de la xarxa del videojoc implementa les següents funcionalitats :

1. Implementa una arquitectura client – servidor.

2. Implementa l'establiment de conexió amb protocol WiFi (router).

3. Implementa un protocol de pas de missatges basat en flags

4. Implementa l'encapsulament de les dades necessàries i la transmissió per WiFi.

L'objectiu d'aquest mòdul és poder comunicar múltiples dispositius mòvils amb una arquitectura client – servidor centralitzada per veure en cada dispositiu els diferents jugadors conectats a "temps real". Aquesta part presenta alguns problemas difícils de resoldre a causa dels *delay's* durant la transmissió que es comenten a l'apartat 4.4.4.
Per altre banda , per a facilitar la tasca d'implementació s'ha utilitzat una extensió del motor gràfic :*AndEngineMultiplayerExtension* que ja incorpora algunes ajudes per establir conexió , pas de missatges i arquitectures client-servidor o p2p.

S'ha de tenir en compte que si no es compleixen alguns requeriments no es pot jugar online: es necessari disposar s'una bona cobertura WiFi (en cas contrari l'establiment de conexió pot no funcionar a la primera), només funciona en xarxes LAN és a dir tots els dispositius mòvils han d'estar conectats al mateix router i la batería ha d'estar per sobre d'un 25% carregada. Per últim, la conexió entre els dispositius no és trivial ni gaire intuïtiva per l'usuari final, l'establiment de conexió és complexe motiu per el qual s'ha afegit al menú principal una petita finestra informativa a mode d'ajuda per l'usuari.

4.4.1 ARQUITECTURA CLIENT - SERVIDOR :

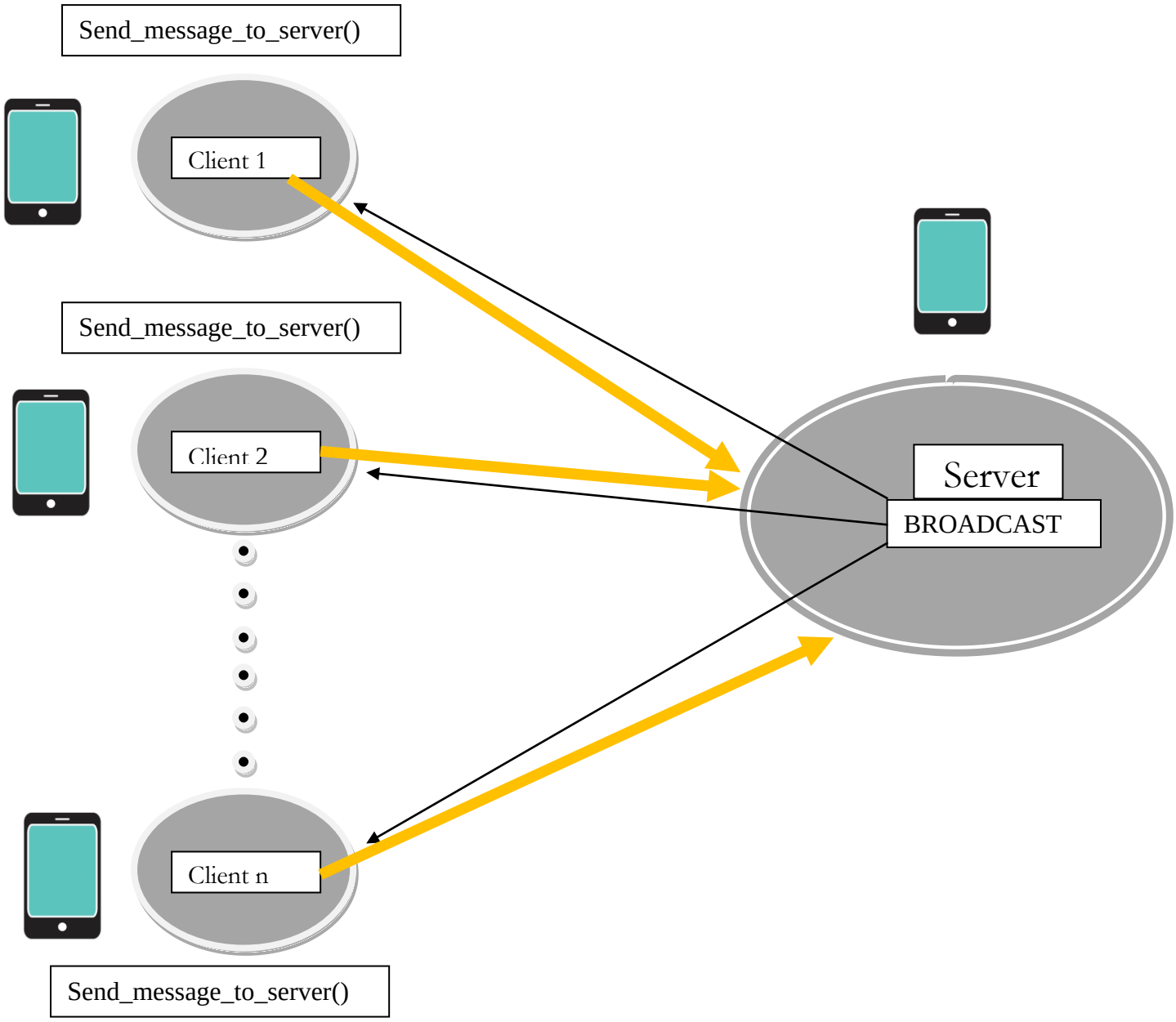
L'arquitectura de xarxa que s'ha decidit implementar ha sigut client-servidor donades les següents raons :

1. Facil escalabilitat per a *n* dispositius.

2. Implementació més simple.

3. Robustesa.

És una arquitectura centralitzada, és a dir, el servidor rep información de tots els clients conectats però cada client només rep la información del servidor. Un altre arquitectura podría ser *peer to peer* però en aquest cas s'ha descartat donades les seves limitacions tan en nombre d'usuaris conectats a la xarxa p2p i els problemas de "delay" , a continuació es mostra una figura que mostra l'estructura general :



Com es pot veure a la figura hi ha un dispositiu que tè el rol de servidor i envía les dades a tots els clients conectats, en canvi cada client només envia les dades al servidor. Això és a causa de els retards que es produirien durant la transmissió donat que per a què la informació d'un client arribés als demès clients s'hauría d'enviar de client X a servidor i de servidor a client Y ocasionant un retard en la actualització de la posició del jugador X visible en el client Y i per altre banda, es congestiona més l'ample de banda de la connexió.

Si s'analitza aquesta estructura es pot extraure la següent conclusió : el servidor veurà en el seu display (escenari de joc) ell mateix i a tots els clients conectats ,i per altre banda cada client només veurà ell mateix i al servidor (però no al reste de clients).

Més endavant s'analitzen les limitacions d'aquesta arquitectura en quant als retards durant la transmissió i la utilització de l'ample de banda de la xarxa al créixer el nombre de clients conectats.

4.4.2 ESTABLIMENT DE CONNEXIÓ:

L'establiment de conexió és el primer objectiu durant el procès d'implementació de la jugabilitat online, aquesta tasca és complexe donat que s'han de gestionar els sockets corresponents tan en el servidor com a tots els clients i es necessari adquirir les direccions IP dels dispositius. Aquest procès es pot dividir en els següents pasos :

1. Crear i obrir sockets en el servidor i en els clients.

2. Obtenir direcció IP del servidor i dels clients a partir del router.

3. Establir conexió entre el servidor i els clients amb les direccions IP manualment.

És un procès sencill però molt important donat que si aquest pas no funciona ja no es podrán comunicar de cap manera els dispositius. És per aquest motiu que durant l'establiment es mostren missatges informatius per pantalla on es pot veure si es produeixen fallades durant tot el procès.
Per establir la conexió de manera gràfica amb l'objectiu de facilitar al màxim la tasca a l'usuari s'ha afegit una funcionalitat al menú principal que mostra una finestra per crear el servidor i els clients amb un menú d'ajuda explicant els pasos a seguir per l'usuari.

4.4.3 PAS DE MISSATGES I TRANSMISSIÓ DE DADES :

La comunicació/transmissió de dades entre el servidor i els clients està basada en flags , és a dir, no es necessari migrar codi cada flag tindrà un codi i uns paràmetres associats. Els tipus de flags definits són :

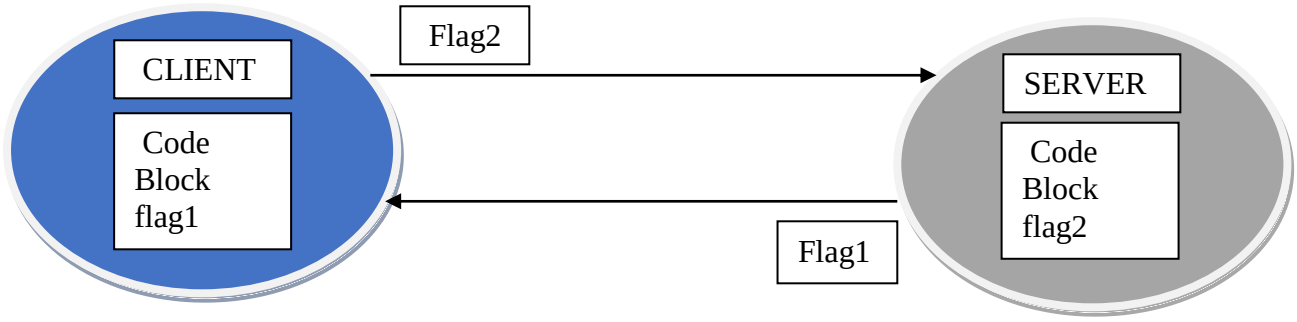
1. ***FLAG_MESSAGE_SERVER_MOVE_PLAYER i FLAG_MESSAGE_CLIENT_MOVE_PLAYER_CLIENT:***
Flag associat a la velocitat i posició del jugador , els paràmetres (dades) que representa són: (Vx,Vy),(Px,Py) i vector director.

2. ***FLAG_MESSAGE_SERVER_PLAYER_SELECTED i FLAG_MESSAGE_CLIENT_PLAYER_SELECTED:***
Aquesta variable representa el jugador seleccionat en el dispositiu no local, per exemple desde el servidor seria el jugador seleccionat al client. Els paràmetres que representa són : ID jugador

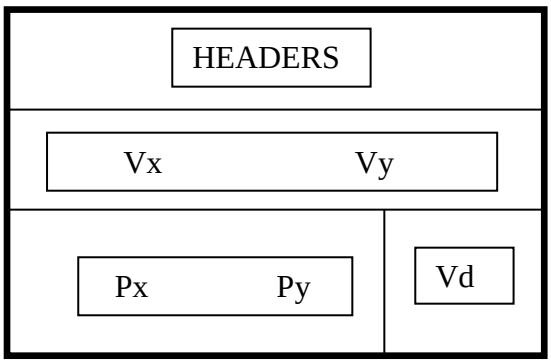
4. ***FLAG_MESSAGE_SERVER_PLAYER I FLAG_MESSAGE_CLIENT_PLAYER:*** Flag associat a paràmetres especials.

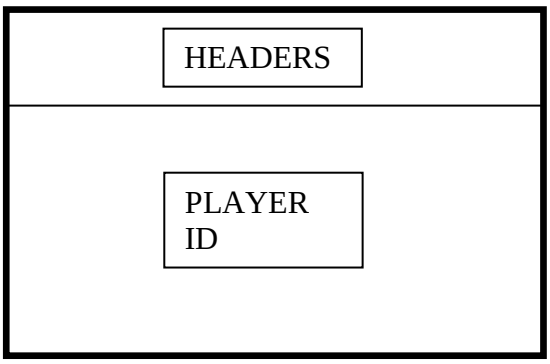
Els flags associats al moviment dels jugadors s'envien intercaladament de client a servidor i de servidor a client cada 0.15 segons per actualitzar a "temps real" en cada dispositiu la

posicio a l'escenari I la velocitat del moviment. En canvi els flags associats al jugador seleccionat només s'envien una sola vegada al tocar el display dins l'escenari la primera vegada, a continuació es mostra una figura que intenta mostrar el funcionament del pas de missatges :



La idea d'aquesta metodología és no haber de migrar codi, empaquetar codi es costòs i dificulta la transmissió per tant l'objectiu será tenir el mateix codi en totes les aplicacions executant-se en el servidor i en els clients de manera que que algunes "parts" del codi només s'executaràn o bé en client o bé en el servidor segons els flags que es rebin.
Com es pot veure a la figura quan el servidor envía el flag de tipus "1" al client , el client executa el bloc de codi associat al flag (el bloc de codi sempre és una funció) amb els paràmetres rebuts per xarxa i de client a servidor exactament el mateix. Segons el tipus de flag les funcions que s'executen són unes o altres i els paràmetres/dades que s'han d'encapsular i enviar són diferents, d'aquesta manera els paquets amb les dades encapsulades són petits reduïnt la probabilitat d'error durant la transmissió. A continuació es presenten les estructures del paquets segons els flag's :

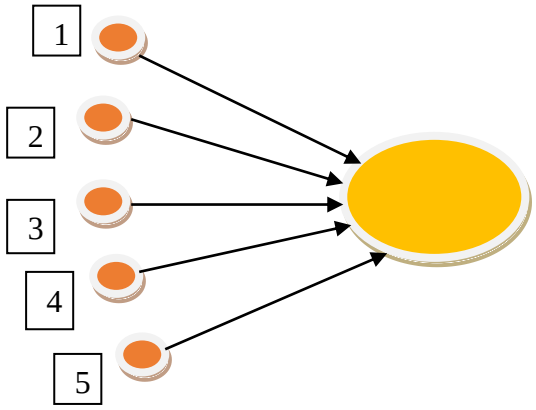




4.4.4 LIMITACIONS DE L'ARQUITECTURA I DEL DISSENY:

Per concluir la part online s'analitzaràn algunes limitacions i problemas que presenta aquesta arquitectura. Com s'ha esmentat anteriorment l'arquitectura client-servidor es facilment escalable però també té alguns inconvenients :

1. Incrementar el nombre de clients implica més còmput pel servidor i retards visuals entre els diferents clients. Suposem que tenim un servidor i 5 clients com mostra la figura, el servidor farà broadcast a tots als clients seqüencialment per tant el client 5 rebrà la información amb un *delay* respecte el client 1 provocant un retard visual entre ambdòs clients. Per altre banda també s'incrementa l'ús de la xarxa local a causa d'enviar més informació.



2. Un altre inconvenient és realizar broadcast desde un client al reste de clients donat que tots els paquets han de pasar necessàriament pel servidor provocant *delay's* no despreciables. Una posible sol.lució podría ser descentralitzar el servidor amb una conexió *fully-connected* (el servidor amb tots els clients i cada client amb el reste de clients), amb la dificultat afegida que això suposaría.

Aquestes limitacions i problemàtiques aparèixen quan creix molt el nombre d'usuaris , per a un cas particular com aquest videojoc on normalment mai hi hauran més de 2 o 3 usuaris en xarxa , aquestes limitacions no són greus.

6. PROVES I CONCLUSIONS :

Durant el procès de desenvolupament d'aquest projecte s'han fet algunes proves o test per comprovar el correcte funcionament del software i aquí es comenten els problemes detectats :

1. La connexió dels dispositius mòbils a través de la xarxa WiFi en el mode de joc *online* no estableix connexió sempre al primer intent.

2. Segons el model d'smartphone i a causa de les llibreries gràfiques AndEngine quan la longitud d'un escenari supera un màxim es veuen totes les textures de color negre.

3. Depenent del model de smartphone al colisionar amb certs objectes en alguns escenaris es produeixen sortides inesperades trencant amb l'execució de l'aplicació (és inestable) a causa del motor AndEngine. El motiu aparent és que al créixer molt el nombre de textures a renderitzar en un escenari el motor produeix fallades.

4. El shader d'aigua dinàmica segons la versió d'openGL present a la versió d'android del smartphone no funciona tot i que no produeix cap fallada durant l'execució.

5. A la modalitat *online* a causa dels retards produïts durant la transmissió de dades d'un dispositiu a un altre a través del router produeix retards visuals i incoherències de posicionament dels jugadors en els diferents dispositius.

Tot i aquests problemes és una aplicació bastant estable (sobretot en els smartphone amb una versió d'Android 2.2) on la gestió de memòria ha sigut un punt crític durant el desenvolupament donat que d'un escenari a un altre s'ha de lliberar memòria cuidadosament.

RESUM :

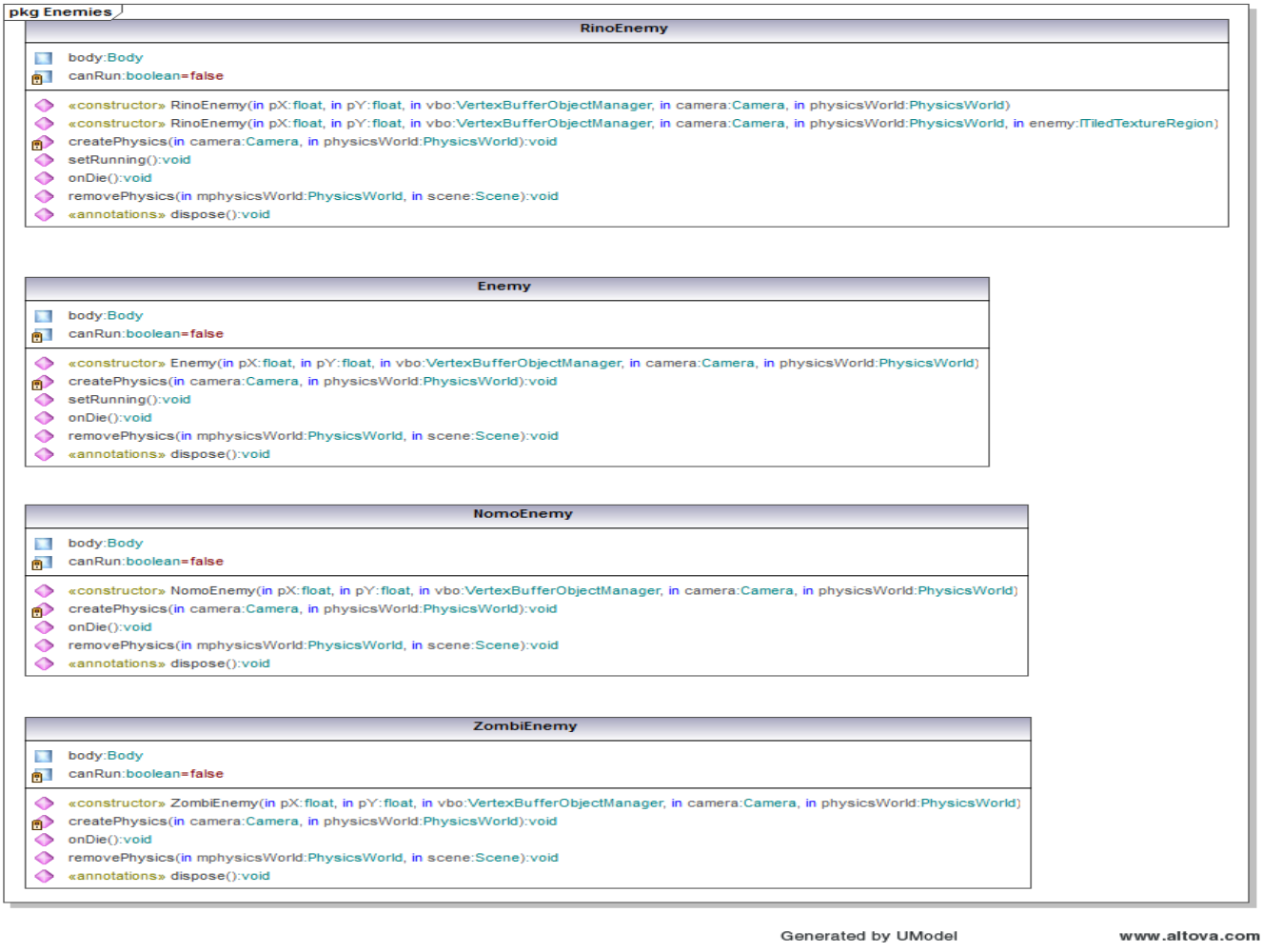
Aquest projecte de final de carrera és un videojoc 2D de plataformes , concretament un RPG, amb realitat augmentada i modalitat multijugador per sistemes operatius Android. És un joc d'entreteniment i lúdic basat en els jocs clàssics com el *mario* intentant donar una aparença més novedosa amb la realitat augmentada. Ha estat desarrollat amb el motor gràfic gratuït AndEngine i Java. La temàtica del joc està dirigida a nens tot i que no és un joc educatiu.

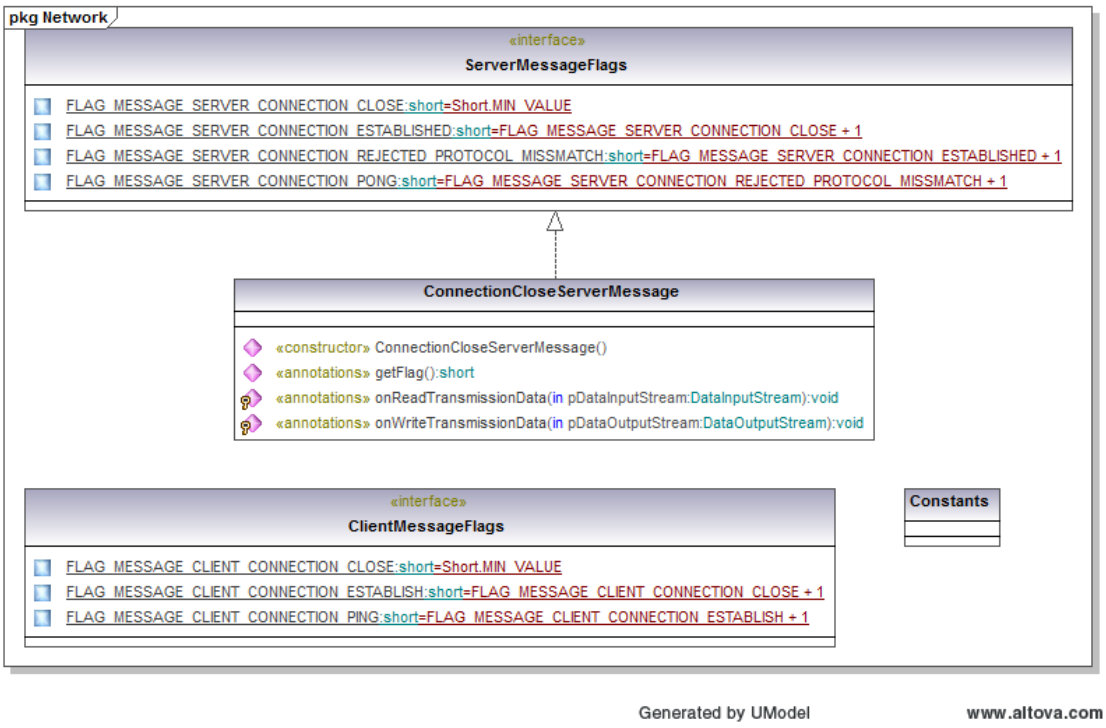
Este proyecto de final de carrera es un videojuego 2D de plataformes , concretamente un RPG, con realidad augmentada y modalidad multijugador para sistemes operativos Android. Es un juego de entretenimiento y lúdico basado en los videojuegos clásicos como el *mario* intentando dar una apariéncia más novedosa con la realidad augmentada. Ha estado desarrollado con el motor gráfico libre AndEngine y lenguage Java. La temàtica del videojuego està dirigida a niños aunque no és un juego educativo.

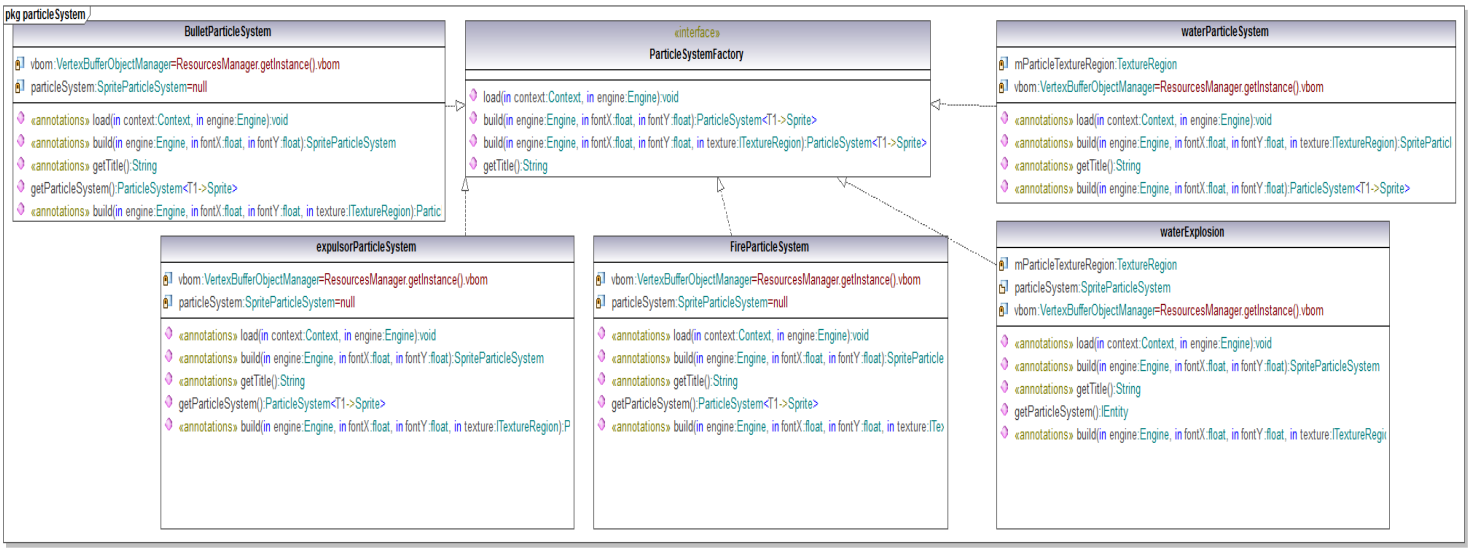
This final degree projecti t is a 2D platform videogame , specifically an RPG , with augmented reality and multiplayer modality gaming for Android operating sistems. It is an entretainment game based in clasic videogames like *mario* trying to give a diferent and original apperance with aumented reality. It has been developed with AndEngine open source graphics engine and Java programin Language. The videogame is themed for children although it is not a educative game.

6. ANEXE :

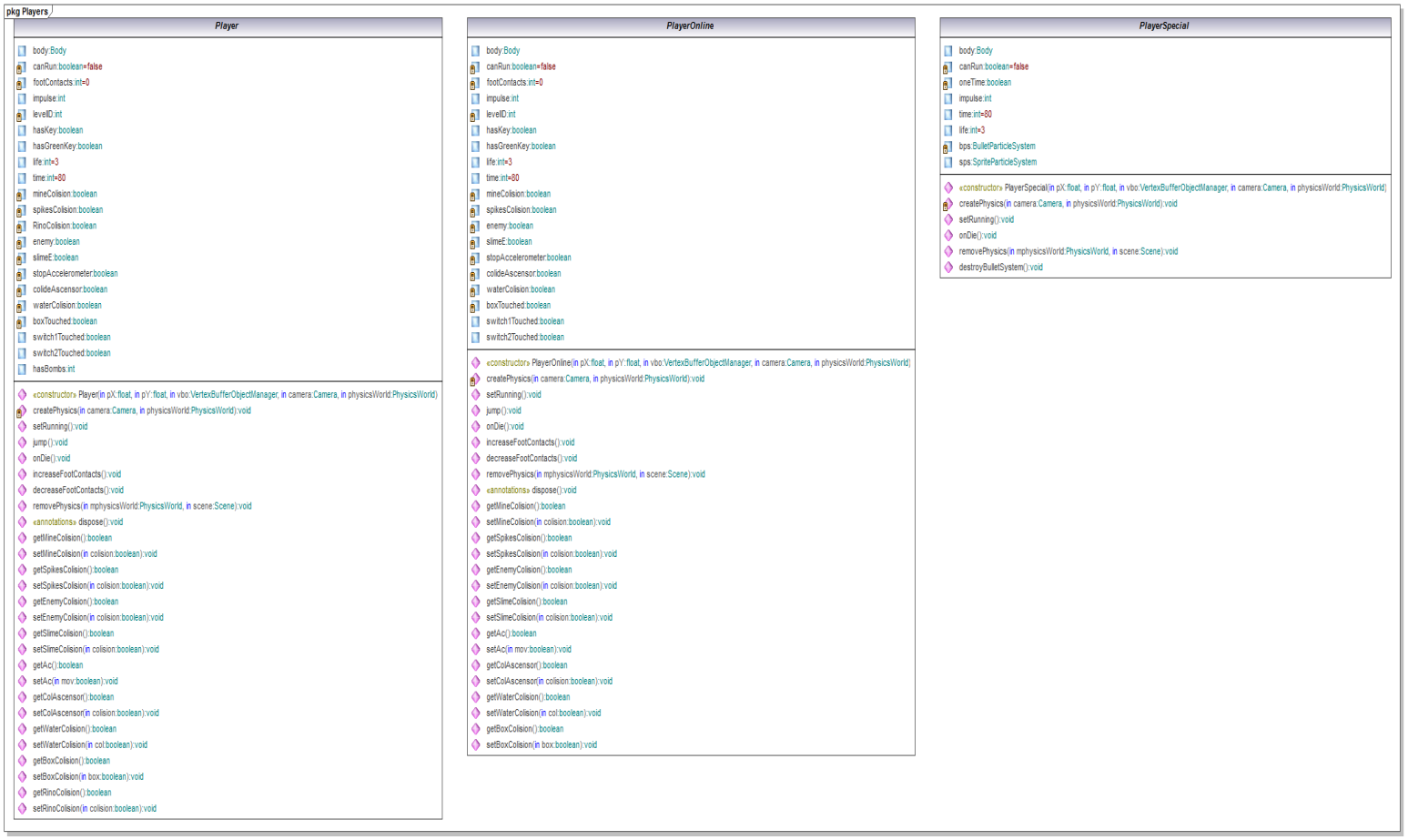
6.1 DIAGRAMES DE CLASSES PER "PACKAGE":

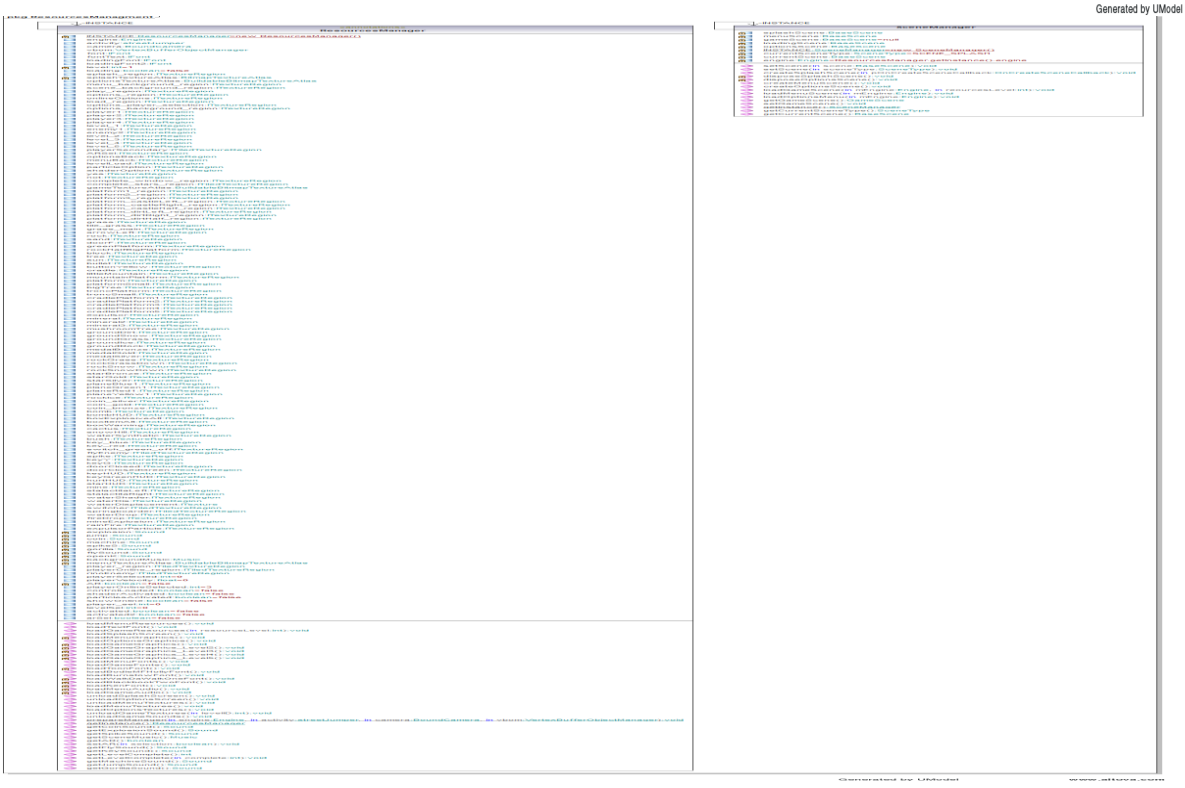

pkg Enemies
RinoEnemy
body:Body
canRun:boolean=false
«constructor» RinoEnemy(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
«constructor» RinoEnemy(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld, in enemy:ITiledTextureRegion)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
setRunning():void
onDie():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
«annotations» dispose():void
Enemy
body:Body
canRun:boolean=false
«constructor» Enemy(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
setRunning():void
onDie():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
«annotations» dispose():void
NomoEnemy
body:Body
canRun:boolean=false
«constructor» NomoEnemy(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
onDie():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
«annotations» dispose():void
ZombiEnemy
body:Body
canRun:boolean=false
«constructor» ZombiEnemy(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
onDie():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
«annotations» dispose():void
Generated by UModel
www.altova.com


pkg Network
«interface»
ServerMessageFlags
FLAG_MESSAGE_SERVER_CONNECTION_CLOSE:short=Short.MIN_VALUE
FLAG_MESSAGE_SERVER_CONNECTION_ESTABLISHED:short=FLAG_MESSAGE_SERVER_CONNECTION_CLOSE + 1
FLAG_MESSAGE_SERVER_CONNECTION_REJECTED_PROTOCOL_MISSMATCH:short=FLAG_MESSAGE_SERVER_CONNECTION_ESTABLISHED + 1
FLAG_MESSAGE_SERVER_CONNECTION_PONG:short=FLAG_MESSAGE_SERVER_CONNECTION_REJECTED_PROTOCOL_MISSMATCH + 1
ConnectionCloseServerMessage
«constructor» ConnectionCloseServerMessage()
«annotations» getFlag():short
«annotations» onReadTransmissionData(in pDataInputStream:DataInputStream):void
«annotations» onWriteTransmissionData(in pDataOutputStream:DataOutputStream):void
«interface»
ClientMessageFlags
FLAG_MESSAGE_CLIENT_CONNECTION_CLOSE:short=Short.MIN_VALUE
FLAG_MESSAGE_CLIENT_CONNECTION_ESTABLISH:short=FLAG_MESSAGE_CLIENT_CONNECTION_CLOSE + 1
FLAG_MESSAGE_CLIENT_CONNECTION_PING:short=FLAG_MESSAGE_CLIENT_CONNECTION_ESTABLISH + 1
Constants
Generated by UModel
www.altova.com

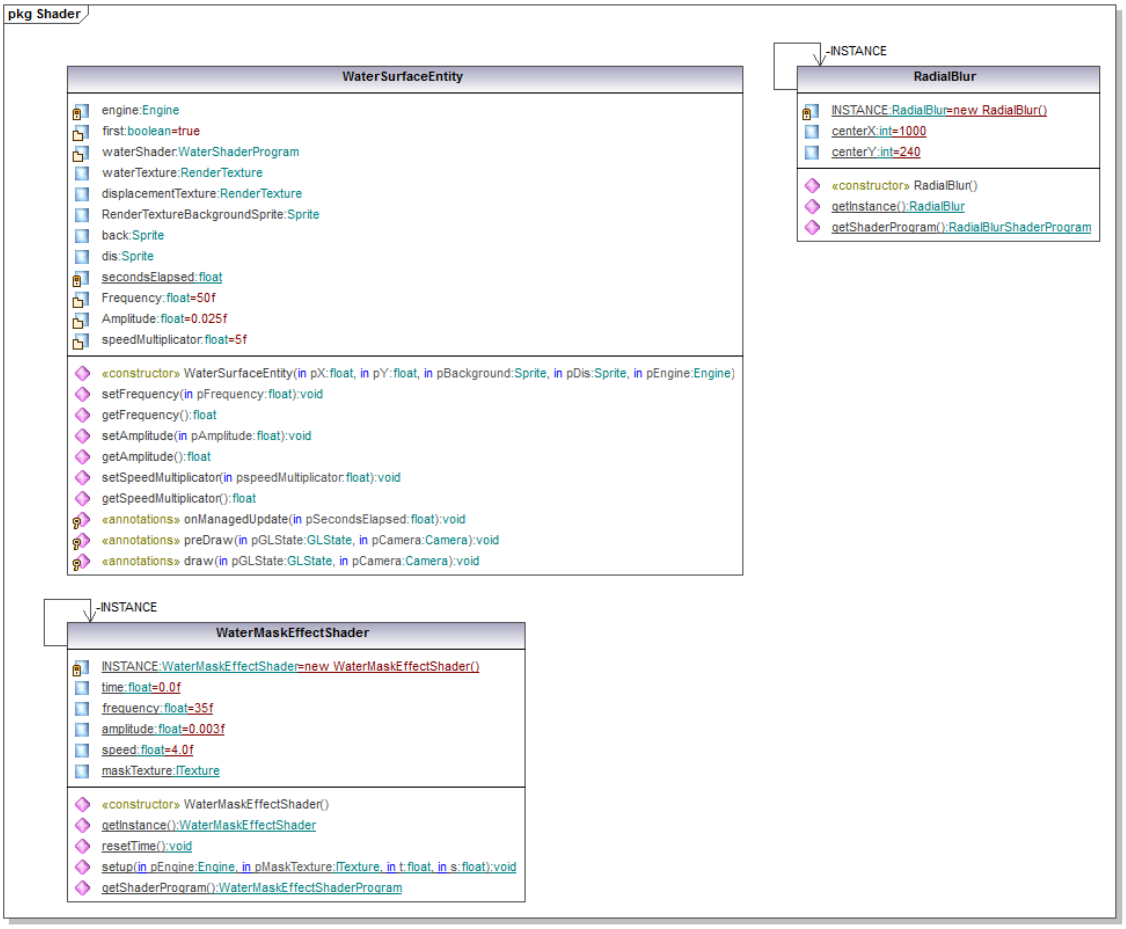

pkg particleSystem
BulletParticleSystem
vbom:VertexBufferObjectManager=ResourcesManager.getInstance().vbom
particleSystem:SpriteParticleSystem=null
«annotations» load(in context:Context, in engine:Engine):void
«annotations» build(in engine:Engine, in fontX:float, in fontY:float):SpriteParticleSystem
«annotations» getTitle():String
getParticleSystem():ParticleSystem<T1->Sprite>
«annotations» build(in engine:Engine, in fontX:float, in fontY:float, in texture:ITextureRegion):Partic
«interface»
ParticleSystemFactory
load(in context:Context, in engine:Engine):void
build(in engine:Engine, in fontX:float, in fontY:float):ParticleSystem<T1->Sprite>
build(in engine:Engine, in fontX:float, in fontY:float, in texture:ITextureRegion):ParticleSystem<T1->Sprite>
getTitle():String
waterParticleSystem
mParticleTextureRegion:TextureRegion
vbom:VertexBufferObjectManager=ResourcesManager.getInstance().vbom
«annotations» load(in context:Context, in engine:Engine):void
«annotations» build(in engine:Engine, in fontX:float, in fontY:float, in texture:ITextureRegion):SpriteParticl
«annotations» getTitle():String
«annotations» build(in engine:Engine, in fontX:float, in fontY:float):ParticleSystem<T1->Sprite>
expulsorParticleSystem
vbom:VertexBufferObjectManager=ResourcesManager.getInstance().vbom
particleSystem:SpriteParticleSystem=null
«annotations» load(in context:Context, in engine:Engine):void
«annotations» build(in engine:Engine, in fontX:float, in fontY:float):SpriteParticleSystem
«annotations» getTitle():String
getParticleSystem():ParticleSystem<T1->Sprite>
«annotations» build(in engine:Engine, in fontX:float, in fontY:float, in texture:ITextureRegion):P
FireParticleSystem
vbom:VertexBufferObjectManager=ResourcesManager.getInstance().vbom
particleSystem:SpriteParticleSystem=null
«annotations» load(in context:Context, in engine:Engine):void
«annotations» build(in engine:Engine, in fontX:float, in fontY:float):SpriteParticle
«annotations» getTitle():String
getParticleSystem():ParticleSystem<T1->Sprite>
«annotations» build(in engine:Engine, in fontX:float, in fontY:float, in texture:ITe
waterExplosion
mParticleTextureRegion:TextureRegion
particleSystem:SpriteParticleSystem
vbom:VertexBufferObjectManager=ResourcesManager.getInstance().vbom
«annotations» load(in context:Context, in engine:Engine):void
«annotations» build(in engine:Engine, in fontX:float, in fontY:float):SpriteParticleSystem
«annotations» getTitle():String
getParticleSystem():IEntity
«annotations» build(in engine:Engine, in fontX:float, in fontY:float, in texture:ITextureRegi
Generated by UModel
www.altova.com

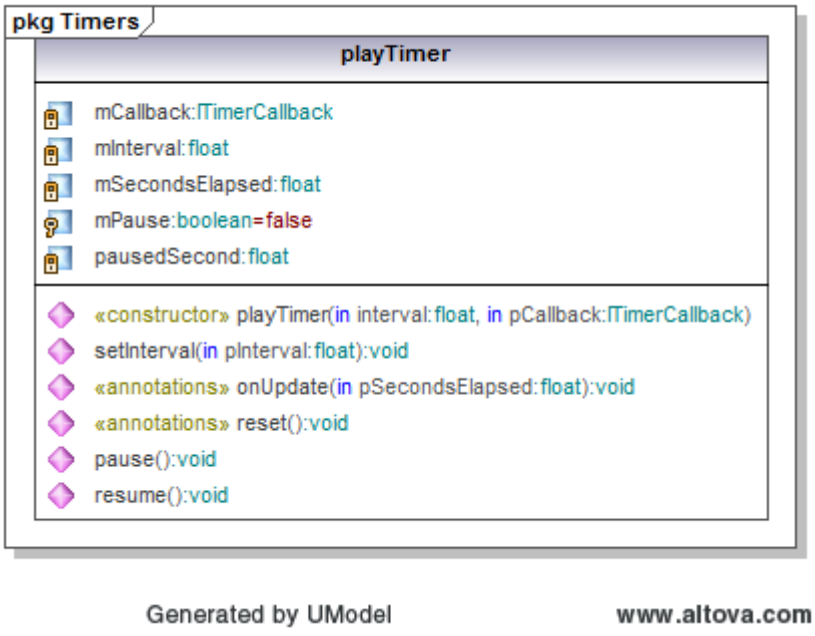

pkg PlatformJumper
streetJumper
dialog:Dialog
LOCALHOST_IP:String="127.0.0.1"
SERVER_PORT:int=4444
FLAG_MESSAGE_SERVER_MOVE_PLAYER:short=1
FLAG_MESSAGE_SERVER_PLAYER_SELECTED:short=FLAG_MESSAGE_SERVER_MOVE_PLAYER + 1
FLAG_MESSAGE_CLIENT_PLAYER_SELECTED_CLIENT:short=FLAG_MESSAGE_SERVER_PLAYER_SELECTED + 1
FLAG_MESSAGE_CLIENT_MOVE_PLAYER_CLIENT:short=FLAG_MESSAGE_CLIENT_PLAYER_SELECTED_CLIENT + 1
FLAG_MESSAGE_SERVER_PLAYER:short=FLAG_MESSAGE_CLIENT_MOVE_PLAYER_CLIENT + 1
FLAG_MESSAGE_CLIENT_PLAYER:short=FLAG_MESSAGE_SERVER_PLAYER + 1
DIALOG_CHOOSE_SERVER_OR_CLIENT_ID:int=0
DIALOG_ENTER_SERVER_IP_ID:int=DIALOG_CHOOSE_SERVER_OR_CLIENT_ID + 1
DIALOG_SHOW_SERVER_IP_ID:int=DIALOG_ENTER_SERVER_IP_ID + 1
camera:BoundCamera
resourcesManager:ResourcesManager
activated:boolean=false
mPlayerIDCounter:int
mPlayers:SparseArray<T1->Sprite>=new SparseArray<Sprite>()
mServerIP:String=LOCALHOST_IP
mSocketServer:SocketServer<T1->SocketConnectionClientConnector>=null
mServerConnector:ServerConnector<T1->SocketConnection>
mClientConnector:SocketConnectionClientConnector
mMessagePool:MessagePool<T1->IMessage>=new MessagePool<IMessage>()
«constructor» streetJumper()
initMessagePool():void
«annotations» onCreateEngineOptions():EngineOptions
«annotations» getDialog():void
«annotations» onCreateEngine(in pEngineOptions:EngineOptions):Engine
«annotations» onCreateResources(in pOnCreateResourcesCallback:OnCreateResourcesCallback):void
«annotations» onCreateScene(in pOnCreateSceneCallback:OnCreateSceneCallback):void
«annotations» onPopulateScene(in pScene:Scene, in pOnPopulateSceneCallback:OnPopulateSceneCallback):void
«annotations» onKeyDown(in keyCode:int, in event:KeyEvent):boolean
«annotations» onCreateDialog(in pID:int):Dialog
«annotations» onDestroy():void
playerSelection(in playerID:int, in pX:float, in pY:float):void
movePlayer(in pID:int, in vX:float, in vY:float, in jumpImpulse:float):void
playerLoad(in playerID:int, in pX:float, in pY:float):void
initServer():void
initServerAndClient():void
initClient():void
log(in pMessage:String):void
toast(in pMessage:String):void
«annotations» onPause():void
«annotations» onAccelerationAccuracyChanged(in pAccelerationData:AccelerationData):void
«annotations» onAccelerationChanged(in pAccelerationData:AccelerationData):void
setAccelerometerActivated(in activated:boolean):void
getConnectedDevices(in YourPhoneIPAddress:String):ArrayList<T1->InetAddress>
Generated by UModel
www.altova.com


pkg Players
Player
body:Body
canRun:boolean=false
footContacts:int=0
impulse:int
levelID:int
hasKey:boolean
hasGreenKey:boolean
life:int=3
time:int=80
mineColision:boolean
spikesColision:boolean
RinoColision:boolean
enemy:boolean
slimeE:boolean
stopAccelerometer:boolean
colideAscensor:boolean
waterColision:boolean
boxTouched:boolean
switch1Touched:boolean
switch2Touched:boolean
hasBombs:int
«constructor» Player(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
setRunning():void
jump():void
onDie():void
increaseFootContacts():void
decreaseFootContacts():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
«annotations» dispose():void
getMineColision():boolean
setMineColision(in colision:boolean):void
getSpikesColision():boolean
setSpikesColision(in colision:boolean):void
getEnemyColision():boolean
setEnemyColision(in colision:boolean):void
getSlimeColision():boolean
setSlimeColision(in colision:boolean):void
getAc():boolean
setAc(in mov:boolean):void
getColAscensor():boolean
setColAscensor(in colision:boolean):void
getWaterColision():boolean
setWaterColision(in col:boolean):void
getBoxColision():boolean
setBoxColision(in box:boolean):void
getRinoColision():boolean
setRinoColision(in colision:boolean):void
PlayerOnline
body:Body
canRun:boolean=false
footContacts:int=0
impulse:int
levelID:int
hasKey:boolean
hasGreenKey:boolean
life:int=3
time:int=80
mineColision:boolean
spikesColision:boolean
enemy:boolean
slimeE:boolean
stopAccelerometer:boolean
colideAscensor:boolean
waterColision:boolean
boxTouched:boolean
switch1Touched:boolean
switch2Touched:boolean
«constructor» PlayerOnline(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
setRunning():void
jump():void
onDie():void
increaseFootContacts():void
decreaseFootContacts():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
«annotations» dispose():void
getMineColision():boolean
setMineColision(in colision:boolean):void
getSpikesColision():boolean
setSpikesColision(in colision:boolean):void
getEnemyColision():boolean
setEnemyColision(in colision:boolean):void
getSlimeColision():boolean
setSlimeColision(in colision:boolean):void
getAc():boolean
setAc(in mov:boolean):void
getColAscensor():boolean
setColAscensor(in colision:boolean):void
getWaterColision():boolean
setWaterColision(in col:boolean):void
getBoxColision():boolean
setBoxColision(in box:boolean):void
PlayerSpecial
body:Body
canRun:boolean=false
oneTime:boolean
impulse:int
time:int=80
life:int=3
bps:BulletParticleSystem
sps:SpriteParticleSystem
«constructor» PlayerSpecial(in pX:float, in pY:float, in vbo:VertexBufferObjectManager, in camera:Camera, in physicsWorld:PhysicsWorld)
createPhysics(in camera:Camera, in physicsWorld:PhysicsWorld):void
setRunning():void
onDie():void
removePhysics(in mphysicsWorld:PhysicsWorld, in scene:Scene):void
destroyBulletSystem():void


Generated by UModel
www.altova.com

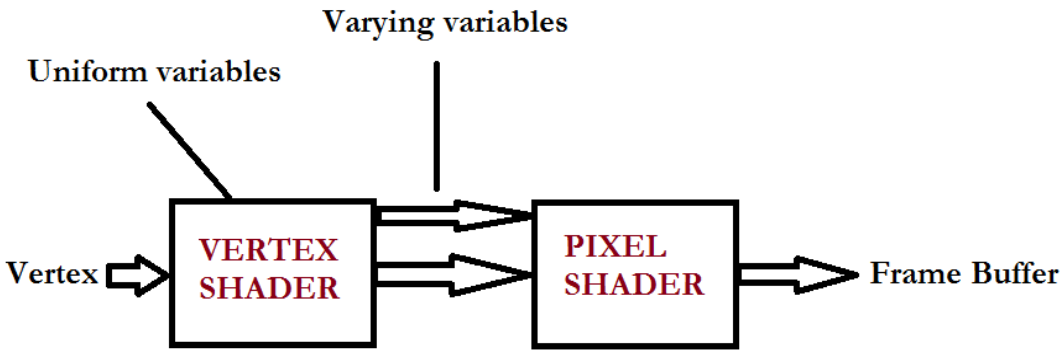

pkg Scene /
BaseScene
GameScene
LoadingScene
SplashScene
MainMenuScene
OptionsScene
LevelCompleteWindow


pkg Shader
WaterSurfaceEntity
engine:Engine
first:boolean=true
waterShader:WaterShaderProgram
waterTexture:RenderTexture
displacementTexture:RenderTexture
RenderTextureBackgroundSprite:Sprite
back:Sprite
dis:Sprite
secondsElapsed:float
Frequency:float=50f
Amplitude:float=0.025f
speedMultiplicator:float=5f
«constructor» WaterSurfaceEntity(in pX:float, in pY:float, in pBackground:Sprite, in pDis:Sprite, in pEngine:Engine)
setFrequency(in pFrequency:float):void
getFrequency():float
setAmplitude(in pAmplitude:float):void
getAmplitude():float
setSpeedMultiplicator(in pspeedMultiplicator:float):void
getSpeedMultiplicator():float
«annotations» onManagedUpdate(in pSecondsElapsed:float):void
«annotations» preDraw(in pGLState:GLState, in pCamera:Camera):void
«annotations» draw(in pGLState:GLState, in pCamera:Camera):void
INSTANCE
RadialBlur
INSTANCE:RadialBlur=new RadialBlur()
centerX:int=1000
centerY:int=240
«constructor» RadialBlur()
getInstance():RadialBlur
getShaderProgram():RadialBlurShaderProgram
INSTANCE
WaterMaskEffectShader
INSTANCE:WaterMaskEffectShader=new WaterMaskEffectShader()
time:float=0.0f
frequency:float=35f
amplitude:float=0.003f
speed:float=4.0f
maskTexture:ITexture
«constructor» WaterMaskEffectShader()
getInstance():WaterMaskEffectShader
resetTime():void
setup(in pEngine:Engine, in pMaskTexture:ITexture, in t:float, in s:float):void
getShaderProgram():WaterMaskEffectShaderProgram
Generated by UModel
www.altova.com


pkg Timers
playTimer
mCallback:ITimerCallback
mInterval:float
mSecondsElapsed:float
mPause:boolean=false
pausedSecond:float
«constructor» playTimer(in interval:float, in pCallback:ITimerCallback)
setInterval(in pInterval:float):void
«annotations» onUpdate(in pSecondsElapsed:float):void
«annotations» reset():void
pause():void
resume():void
Generated by UModel
www.altova.com

6.2 SHADERS :

A continuació s'explica l'algoritme de aigua dinàmica implementat en GLSL presentat a l'apartat d'implementació dels escenaris. Primer s'explica per sobre que és un shader i com és el seu fluxe de funcionament :



La idea d'un shader és definir un pipeline de funcionament independent del workflow del openGL (en aquest cas). La entrada de dades són tots els vèrtex de l'escenari , la primera fase del pipeline coneguda com *vertex shader* és un fragment de codi (escrit pel programador) que s'aplica a cada vèrtex , s'ha de tenir en compte que si volem passar variables de l'aplicació externa al vèrtex shader aquestes variables han de ser variables *uniform* com mostra la figura. La següent fase del pipeline conegut com fragment/píxel shader conté el fragment de codi que s'aplicarà a cada píxel de l'escenari, i finalment la sortida de dades és el color per píxel al frame buffer de pantalla.

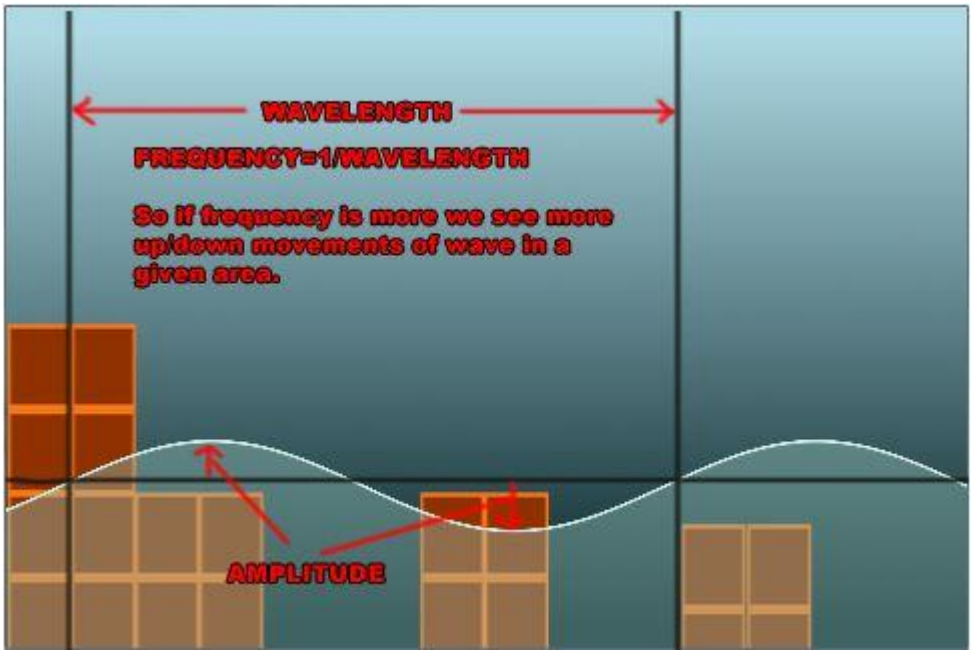
El shader que s'implementa per simular aigua dinàmica al primer escenari de joc tè el següent codi que es comenta a continuació :

```
precision lowp float
uniform lowp sampler2D
ShaderProgramConstants.UNIFORM_TEXTURE_1
uniform lowp sampler2D
ShaderProgramConstants.UNIFORM_TEXTURE_0
varying mediump vec2
ShaderProgramConstants.VARYING_TEXTURECOORDINATE
uniform float timedelta;
uniform float frequency;
uniform float amplitude;

void main()
{
```

```
vec4 v_color = vec4(1.0);
vec2 maskAlpha = texture2D(ShaderProgramConstants.UNIFORM_TEXTURE_1 +
vec2(ShaderProgramConstants.VARYING_TEXTURECOORDINATES + .x/6.0, +
ShaderProgramConstants.VARYING_TEXTURECOORDINATES + ".y/6.0)).xy;
float t = ShaderProgramConstants.VARYING_TEXTURECOORDINATES + .y +
maskAlpha.y *0.1-0.15 + (sin (+
ShaderProgramConstants.VARYING_TEXTURECOORDINATES + .x * frequency +
timedelta) * amplitude);
vec4 backgroundColor = v_color * texture2D("+
ShaderProgramConstants.UNIFORM_TEXTURE_0 +  vec2( +
ShaderProgramConstants.VARYING_TEXTURECOORDINATES + .x, t));
gl_FragColor = backgroundColor;
}
```

En aquest shader tot el codi està en el pixel shader donat que totes les operacions en aquest cas es fan per píxel. La idea és generar una funció trigonomètrica a partir d'un mapa de punts i modificar l'amplitud, freqüència i velocitat de la ona amb unes variables uniform passades desde l'aplicació principal per canviar la posició i color de la textura d'aigua per píxel.



Com es pot observar a l'imatge l'objectiu del codi és generar moviment a la textura d'aigua segons els paràmetres freqüència, amplitud i speed :

1. **Freqüència** : Un valor més alt implica generar més moviments verticals en una període d'ona.

2. **Amplitud** : Un valor més alt implica generar més ones en una área donada.

6.3 ENQUESTES :

Al finalitzar el desenvolupament del software es van passar unes enquestes a un conjunt de persones amb la intenció d'obtenir crítiques i comentaris que ajudin a millorar alguns aspectes importants del videojoc. Les conclusions que es van extraure de les enquestes van ser :

1. Falta de jugabilitat

2. Menús poc intuïtius

3. Disseny gràfic pobre

A continuació es mostra la plantilla d'enquesta que es va utilitzar :

Enquesta streetJumper :

1. Com califiqueu la jugabilitat ?

a) Molt jugable (Divertit)
b) Prou jugable
c) Acceptablement jugable
d) Aburrit
e) Molt aburrit

2. Que us semblen els grafics ?

a) Molt bons / atrauen la vostra atenció
b) "normals", res especial ni diferent
c) Molt treballats però no agraden
d) Poc treballats
e) Una caca

3. En quant als menú/s …….

a) Són molt intuitius
b) Es difícil saber que s'ha de fer per iniciar partida
c) No s'entèn absolutament res
d) Es intuitiu però podría estar millor
e) No és gens intuitiu i , a mès l'aparença és horrible

4. En general, quina nota li doneu al joc ?

a) 9
b) 7
c) 5
d) 3
e) 0

5. Com creíeu que es podría millorar ?

6. Opinions personals (tots els comentaris serán benvinguts)

7. En quant a la jugabilitat on-line :

a) Funciona correctament i augmenta la jugabilitat
b) Funciona fatal i no es pot jugar, i a mès és aburrit
c) Té masses retards de moviment i fa difícil jugar
d) És una merda
e) Està ben feta però no es gens intuitiu com establir conexió