MemoryCon: una aplicació d'Android per a potenciar la memòria dels malalts d'Alzheimer

Treball Final de Grau: Videojocs educatius

Autor: Sergi Colina Arrufat

Directors del projecte: Heliodoro Tejedor Navarro

Jordi Duch Gavaldà

Gener del 2015

FITXA DEL TREBALL FINAL

Títol del treball:	MemoryCon: una aplicació d'Android per a potenciar la memòria dels malalts d'Alzheimer
Nom de l'autor:	Sergi Colina Arrufat
Nom dels consultors:	Heliodoro Tejedor Navarro Jordi Duch Gavaldà
Data de lliurament (mm/aaaa):	01/2015
Àrea del Treball Final:	Videojocs Educatius
Titulació:	2n cicle d'Eng. Informàtica

Resum del Treball:

Aquest Treball de Fi de Grau consisteix en una aplicació per a dispositius mòbils amb sistema operatiu Android, i que està basada en un manual d'exercicis genèrics per a malalts d'Alzheimer. L'objectiu del manual —i per tant, del videojoc— no és altre que el de servir d'ajuda en l'estimulació de les capacitats cognitives dels pacients, és a dir, dels jugadors.

El joc, que s'engloba dins del grup dels *casual games*, consisteix a anar responent diferents qüestions a diverses pantalles. Cada pantalla representa un dels exercicis del manual. La dificultat dels exercicis varia, així com l'àrea cognitiva que estimula: orientació personal, temporal i espaial, memòria verbal immediata, memòria sobre el coneixement del món, capacitat d'una persona per a reconéixer colors, etc.

Per tal d'atorgar certa gamificació al joc, es determina una fita per al jugador: trobar un tresor pirata. Per cada pregunta encertada, sumem punts. Les preguntes errònies descompten punts, tot i que no es pot baixar de zero. A mida que anem sumant punts, aconseguirem uns objectes que ens ajudaran a trobar el tresor.

Adaptant un bastiment dissenyat pels autors del llibre *Beginning Android 4 Games Development*, s'ha procurat dotar a l'aplicació en tot moment d'una aparença senzilla i càlida, amb gràfics, colors, músiques i sons adequats per a intentar assolir els objectius plantejats.

Abstract:

This Degree Project is the development of an application for mobile devices with Android operating system. It is based on a generic manual with exercises for Alzheimer patients. The aim of the manual —and therefore, the video game— is simply to be helpful in stimulating cognitive abilities of patients, ie players.

The game, which is part of the group of casual games, consists of answering different questions to several screens. Each screen represents an exercise from the manual. The difficulty of the exercises varies, and each one stimulates different cognitive functions: personal, temporal and spatial orientation; immediate verbal memory; memory about about the world knowledge; one's ability to recognize different colors; etc.

In order to give some gamification to the game, the player has a goal: to find a pirate treasure. For each correct question, it adds points to the score. Wrong questions substract points, but the score can never go less than zero. As we score points, we'll get some items that will help us find the treasure.

By adapting a framework designed by the authors of the book 'Beginning Android 4 Games Development', the application has been provided with a simple and warm looking, with proper graphics, colors, music and sounds in order to achieve the proposed objectives.

Paraules clau:

MemoryCon / Android / Alzheimer / videojocs / educatius

Índex

1. Introducció

1.1 Context i justificació del Treball

Un videojoc educatiu és una eina d'aprenentatge que proporciona un valor educacional al jugador. No parlem només de la transmissió de valors socials i coneixements: el seu ús produeix reduccions en els temps de reacció i una millor coordinació psicomotriu, a més d'elevar l'autoestima dels jugadors. La curiositat, la diversió i la naturalesa del desafiament també poden afegir-se al potencial educatiu d'un joc.

Si a més a més hi comptem l'adquisició d'habilitats en resolució de problemes; negociació; anàlisi; capacitat de judici; pensament estratègic; capacitat de comunicació; creació de xarxes socials; habilitats narratives; navegació transmèdia (el procés d'explicar històries a través de diversos mitjans de comunicació i la seva exploració); patrons de pensament no lineal; atenció, visió i cognició millorades... la llista de beneficis és extensa.

Des de la proliferació dels diferents dispositius mòbils, el mercat de videojocs ha experimentat un creixement vertiginós. Al primer trimestre del 2013, Android havia guanyat la seva particular batalla amb les tauletes iOS, que fins llavors Apple dominava sense discussió:

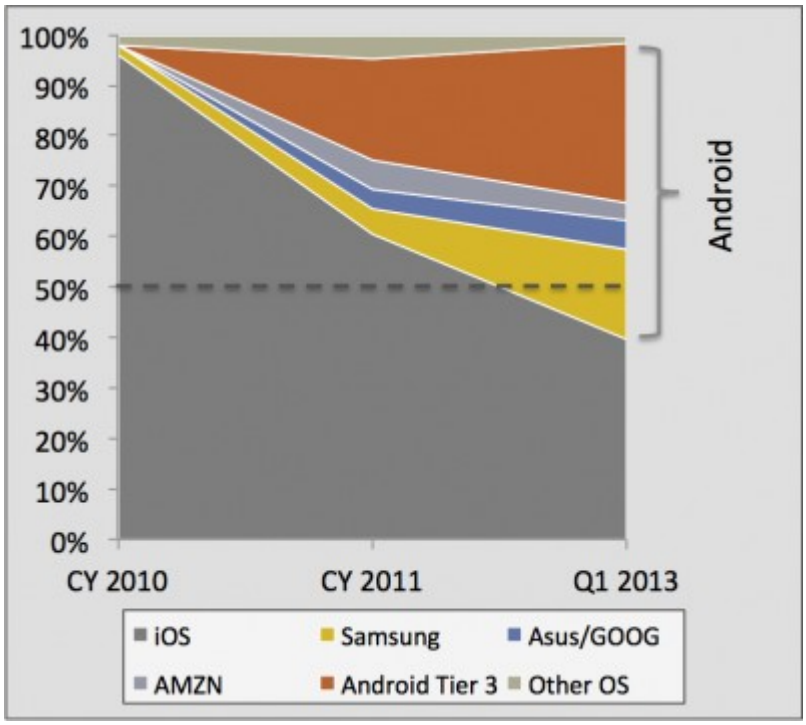


Figura 1.1: Quota global de mercat de les tauletes electròniques per sistema operatiu i venedor (font: IDC i Strategy Analytics).

	Apple iOS 7	Android 4.3	Windows Phone 8	BlackBerry OS 7	Symbian 9.5
Compañía	Apple	Open Handset Alliance	Microsoft	RIM	Symbian Foundation
Núcleo del SO	Mac OS X	Linux	Windows NT	Mobile OS	Mobile OS
Licencia de *software*	Propietaria	Software libre y abierto	Propietaria	Propietaria	Software libre
Año de lanzamiento	2007	2008	2010	2003	1997
Fabricante único	Sí	No	No	Sí	No
Variedad de dispositivos	modelo único	muy alta	media	baja	muy alta
Soporte memoria externa	No	Sí	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit	Pocket Internet Explorer	WebKit	WebKit
Soporte Flash	No	Sí	No	Si	Sí
HTML5	Sí	Sí	Sí	Sí	No
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry App World	Ovi Store
Número de aplicaciones	825.000	850.000	160.000	100.000	70.000
Coste publicar	$99 / año	$25 una vez	$99 / año	sin coste	$1 una vez
Actualizaciones automáticas del S.O.	Sí	depende del fabricante	depende del fabricante	Sí	Sí
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM	ARM
Máquina virtual	No	Dalvik	.net	Java	No
Aplicaciones nativas	Siempre	Sí	Sí	No	Siempre
Lenguaje de programación	Objective-C, C++	Java, C++	C#, muchos	Java	C++
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac	Windows, Mac, Linux

Taula 1.1: Comparativa de les principals plataformes mòbils (font: androidcurso.com)

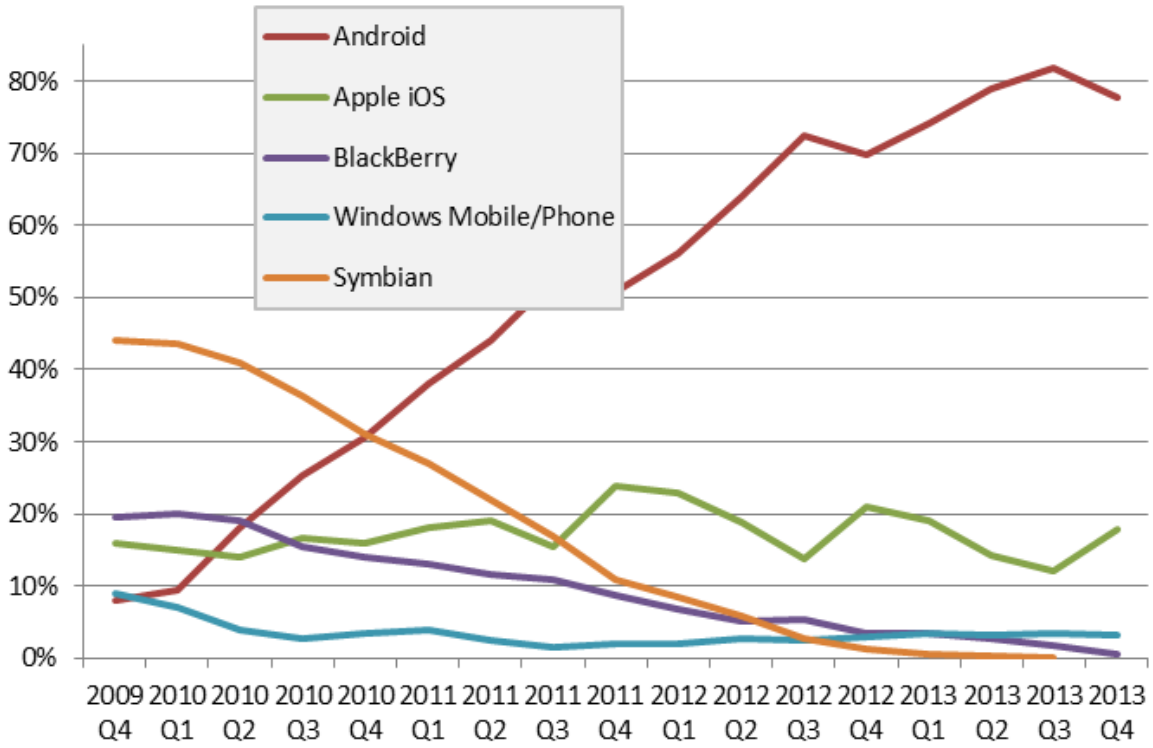


Figura 1.2: Percentatge de telèfons intel·ligents venuts al món segons el seu sistema operatius fins al darrer quart del 2013 (font: Gartner Group).

Aquest canvi de tendència s'ha degut principalment a quatre factors:

1. Ara els dispositius mòbils serveixen per a molt més que per al consum audiovisual;
2. Android ha madurat;
3. L'ascens de Samsung;
4. La democratització dels dispositius mòbils.

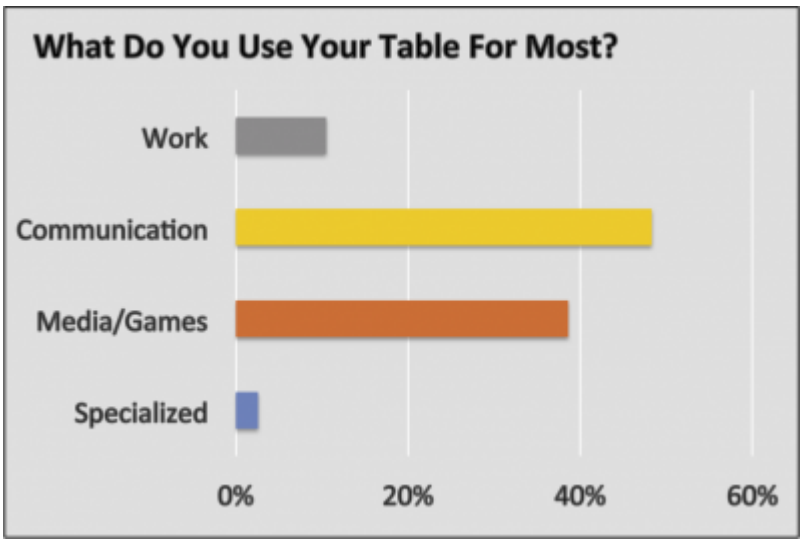


Figura 1.3: estadístiques d'ús de tauletes (Font: CivicScience.com)

1.2. Motivació i objectius del treball

Dues han estat les motivacions principals per dur a terme aquest treball de fi de grau: d'una banda, aprofundir en el coneixement de la plataforma Android (i més encara si tenim en compte que els meus coneixements de Java es limiten tan sols al que he après durant els estudis de Grau), i de l'altra, aconseguir implementar una manera més lúdica de realitzar exercicis destinats a malalts d'Alzheimer, una malaltia que li ha estat diagnosticada a un membre de la meva família, tot i que en un estadi molt inicial.

Així doncs, l'objectiu principal d'aquest Treball Final de Grau és el desenvolupament d'un videojoc educatiu de tipus casual per a dispositius Android, amb versions superiors a 1.5. Al mateix temps, i com a conseqüència, adquirir i ampliar els meus coneixements sobre el món de la programació Java i el disseny de videojocs.

1.3. Enfocament i mètode seguit

El joc consisteix en una sèrie d'exercicis individuals on l'usuari haurà d'anar triant les respostes correctes. Tenint en compte l'especial condició del jugador, s'ha procurat mantenir en tot moment una estètica agradable i entenedora, on predominin els colors primaris i secundaris, i amb les diferents zones a cada pantalla ben definides. L'objectiu és que l'usuari aprengui —recordi— constantment, per tant no té gaire sentit que passi pantalles sense conèixer les respostes correctes. Així doncs, per passar d'una activitat a una altra, l'usuari haurà hagut de contestar correctament totes les preguntes (eventualment si més no, malgrat hagi comès alguns errors).

S'ha volgut atorgar certa **gamificació** al joc, en forma d'una "recerca del tresor". En funció de la puntuació obtinguda (200, 300, 400... 1000) van apareixent una sèrie d'objectes (una pala, una brúixola, cèntims per a poder pagar la tripulació, un mapa, etc.) que ens ajudaran en la seva cerca. A mes a més, s'ha afegit un marcador de temps i un sistema de *bonus* de puntuació en funció del temps emprat en resoldre l'exercici i la dificultat d'aquest. També disposem d'una pantalla amb les puntuacions més altes al finalitzar el joc.

Mitjançant l'ús de colors adequats, sons, músiques, icones i botons s'ha volgut indicar i definir al màxim les diferents *affordances*, retroaccions, metàfores, visibilitats i possibles restriccions de cada pantalla.

Val a dir, però, que en aquest cas en concret és desitjable que l'usuari tingui una persona al costat per tal que l'experiència sigui més enriquidora. Encara que algun dels exercicis poden ser resolts de manera individual pel pacient, és necessària la supervisió en tots ells de la persona responsable per comprovar que ho fa correctament.

1.4 Planificació del projecte

S'adjunta l'arxiu *Pla temporal.mpp*, realitzat amb el programari *Microsoft Project 2010*, on es pot comprovar la temporalització i el diagrama de Gantt del projecte.

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Preparació Pla Projecte	16 días	mié 17/09/14	mié 08/10/14		
Elecció de tecnologies	3 días	mié 17/09/14	vie 19/09/14		
Elecció de coneixements teòrics	3 días	mié 17/09/14	sáb 20/09/14		
Elecció temàtica	4 días	lun 22/09/14	jue 25/09/14		
Configuració entorn IDE Eclipse	2 días	vie 26/09/14	lun 29/09/14		
Inici TFG	0 días	mié 08/10/14	mié 08/10/14		
Disseny del videojoc	5 días	mié 08/10/14	lun 13/10/14	**6**	
Disseny gràfic	5 días	mié 08/10/14	lun 13/10/14		
Cerca d'imatges	2 días	mié 08/10/14	jue 09/10/14		
Maquetació Adobe Photoshop	3 días	vie 10/10/14	lun 13/10/14	9	
Reunions	1 día	jue 09/10/14	jue 09/10/14		
Reunió metgessa CAP Ribes	1 día	mié 08/10/14	mié 08/10/14		
Revisió Planificació	3 días	mié 08/10/14	vie 10/10/14		
Cerca de documentació	3 días	jue 09/10/14	sáb 11/10/14	12	
Redacció PAC1 - Disseny del videojoc	2 días	sáb 11/10/14	lun 13/10/14		
Entrega PAC1	0 días	lun 13/10/14	lun 13/10/14	15	
Versió parcial del joc	37 días	mar 14/10/14	dom 30/11/14	**16**	
Disseny gràfic	11 días	mar 14/10/14	mar 28/10/14		
Atles de textures	2 días	mar 14/10/14	mié 15/10/14		
Maquetació Adobe Photoshop	2 días	mar 14/10/14	mié 15/10/14		
Implementació pantalles	28 días	jue 16/10/14	sáb 22/11/14		
Exercici 1	5 días	jue 16/10/14	mié 22/10/14		
Exercici 2	3 días	jue 23/10/14	lun 27/10/14	22	
Exercici 3	2 días	mar 28/10/14	mié 29/10/14	23	
Exercici 4	2 días	jue 30/10/14	vie 31/10/14	24	
Exercici 5	6 días	lun 03/11/14	lun 10/11/14	25	
Exercici 6	2 días	mar 11/11/14	mié 12/11/14	26	
Exercici 7	3 días	jue 13/11/14	lun 17/11/14	27	
Puntuació final	1 día	mar 18/11/14	mar 18/11/14	28	
Primera entrega PAC2 - versió parcial del joc (arxiu apk)	0 días	sáb 22/11/14	sáb 22/11/14		
Realització vídeo explicatiu	1 día	lun 24/11/14	lun 24/11/14		
Segona entrega PAC2 - versió parcial del joc (arxiu de vídeo)	0 días	mié 26/11/14	mié 26/11/14	36	
Revisió requeriments, segons indicacions dels consultors	1 día	sáb 29/11/14	sáb 29/11/14	37	
Reimplementació pantalles	5 días	mar 25/11/14	sáb 29/11/14		
Exercici 1	2 días	mar 25/11/14	mié 26/11/14		
Exercici 2	2 días	mar 25/11/14	mié 26/11/14		
Exercici 3	2 días	mar 25/11/14	mié 26/11/14		
Exercici 4	4 días	mié 26/11/14	sáb 29/11/14		
Exercici 5	4 días	mié 26/11/14	sáb 29/11/14		
Exercici 6	4 días	mié 26/11/14	sáb 29/11/14		
Exercici 7	4 días	mié 26/11/14	sáb 29/11/14		
Realització vídeo explicatiu	1 día	dom 30/11/14	dom 30/11/14	41	
Tercera entrega PAC2 - versió parcial del joc (arxiu de vídeo)	0 días	dom 30/11/14	dom 30/11/14	42	
Versió jugable	19 días	lun 01/12/14	jue 25/12/14		
Revisió requeriments, segons indicacions dels consultors	2 días	lun 01/12/14	mar 02/12/14		

Reimplementació pantalles	4 días	mar 02/12/14	vie 05/12/14		
Exercici 1	2 días	mar 02/12/14	mié 03/12/14		
Exercici 2	2 días	mar 02/12/14	mié 03/12/14		
Exercici 3	2 días	mar 02/12/14	mié 03/12/14		
Exercici 4	4 días	mar 02/12/14	vie 05/12/14		
Exercici 5	4 días	mar 02/12/14	vie 05/12/14		
Exercici 6	4 días	mar 02/12/14	vie 05/12/14		
Exercici 7	4 días	mar 02/12/14	vie 05/12/14		
Implementació noves pantalles	7 días	mar 09/12/14	mié 17/12/14		
Exercici 7b	3 días	mar 09/12/14	jue 11/12/14		
Exercici 7c	3 días	mar 09/12/14	jue 11/12/14		
Pantalla principal	2 días	mié 10/12/14	jue 11/12/14		
Pantalla del tresor	3 días	mié 10/12/14	vie 12/12/14		
Pantalla “how to”	2 días	mié 10/12/14	jue 11/12/14		
Pantalla introductòria	2 días	mié 10/12/14	jue 11/12/14		
Pantalla “highscores”	6 días	mar 09/12/14	mar 16/12/14		
Millora codi exercici 5	3 días	mié 17/12/14	vie 19/12/14	61	
Revisió codi (problemes de memòria VRAM)	3 días	lun 22/12/14	mié 24/12/14	61	
Realització vídeo explicatiu	1 día	jue 25/12/14	jue 25/12/14	63	
Entrega PAC3 - Versió jugable (codi font + vídeo)	0 días	jue 25/12/14	jue 25/12/14	64	
Tancament Projecte	11 días	dom 28/12/14	vie 09/01/15	**65**	
Revisió documentació	5 días	dom 28/12/14	jue 01/01/15		
Redacció memòria	5 días	vie 02/01/15	jue 08/01/15	67	
Entrega final	0 días	vie 09/01/15	vie 09/01/15	68	
Projecte Tancat	0 días	vie 09/01/15	vie 09/01/15	69	

Taula 4.1. Pla temporal del projecte

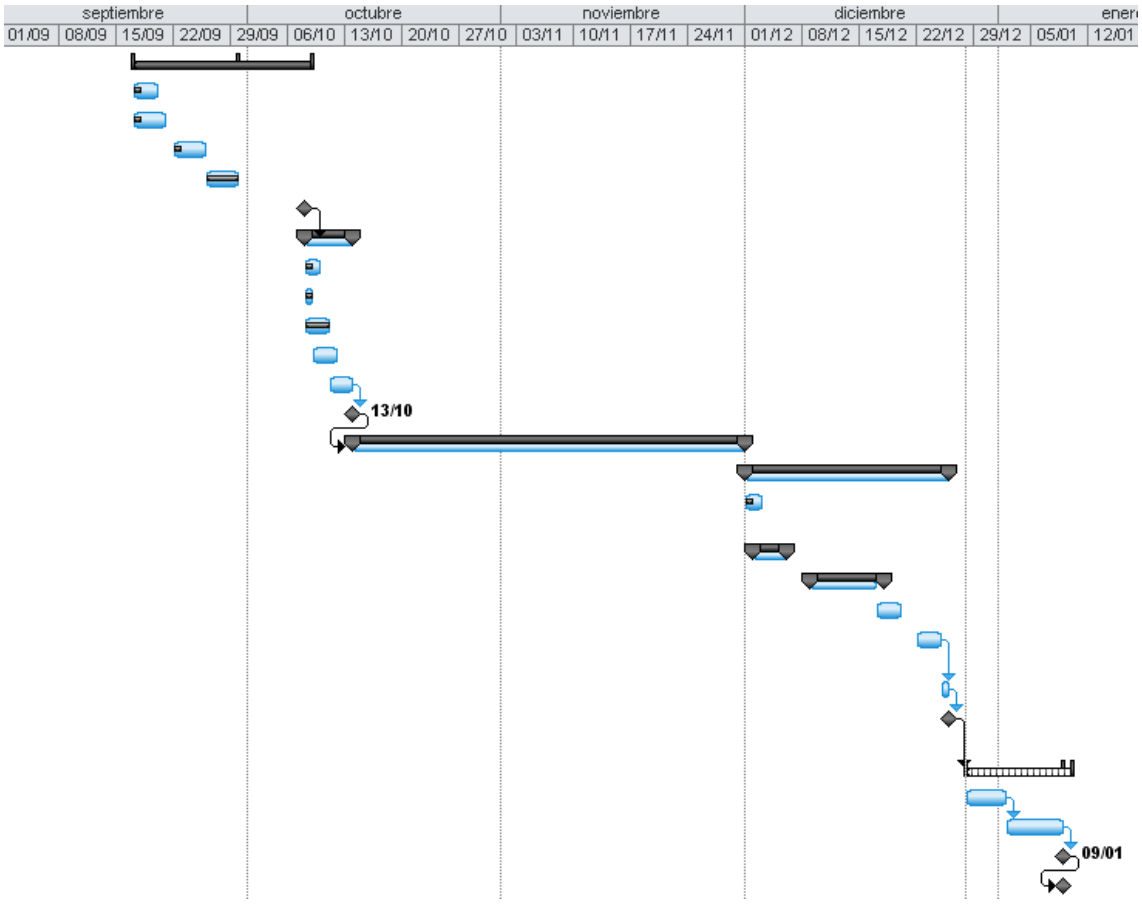


Figura 4.1. Diagrama de Gantt del projecte

1.5 Breu sumari de productes obtinguts

En aquest TFG s'ha desenvolupat un videojoc per a dispositius mòbils basats en Android. Així doncs, i en última instància, s'ha obtingut un arxiu APK (*Aplication Package File*) que es podrà instal·lar en qualsevol dispositiu a partir de la versió d'Android 1.5.

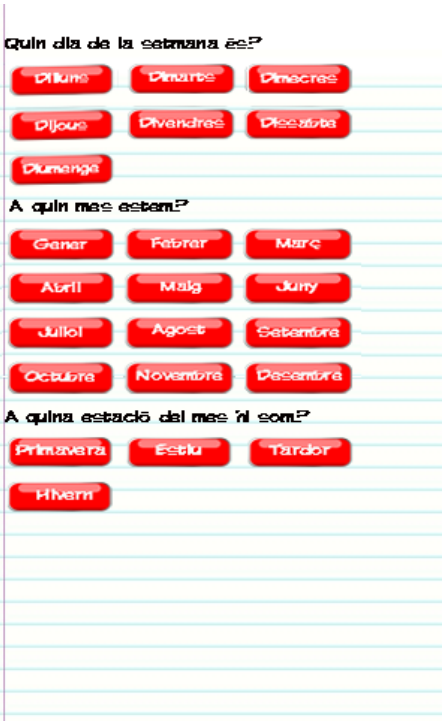
1.6 Breu descripció dels altres capítols de la memòria

Es posa especial èmfasi en les diferents parts que han conformat el procés de desenvolupament i implementació d'aquest videojoc: disseny gràfic, jerarquia de classes, anàlisi i comprensió del bastiment, estructura dels paquets, sense oblidar la descripció i explicació de les classes principals, així com de les funcions més importants dins del codi. També es fa referència a les principals eines utilitzades.

2. Disseny gràfic

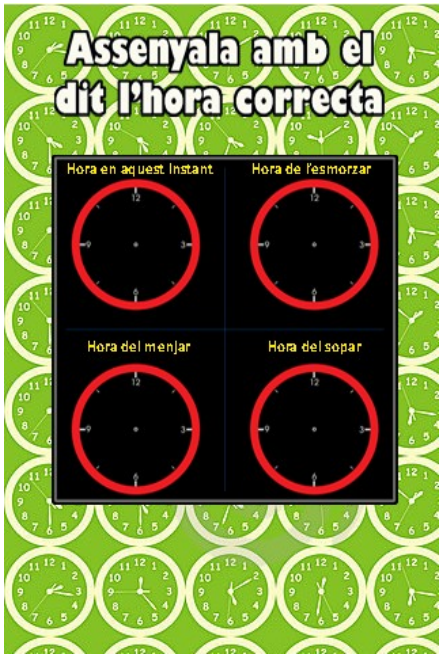
MemoryCon disposa de set activitats de joc diferents, cadascuna de les quals —a excepció de la sisena, on no s'ha considerat necessària— precedida d'una pantalla introductòria i explicativa. Totes les pantalles han estat dissenyades amb *Adobe Photoshop CS3*.

Exercici 1: pantalla d'introducció



Exercici 1: pantalla de joc

Exercici 2: pantalla d'introducció

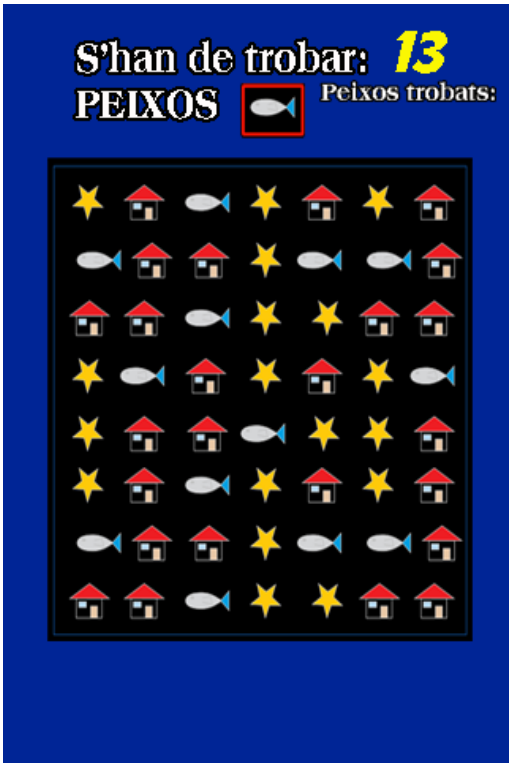


Exercici 2: pantalla de joc

Exercici 3: pantalla d'introducció

Exercici 3: pantalla de joc

Exercici 4: pantalla d'introducció



Exercici 4: pantalla de joc

Exercici 5: pantalla d'introducció

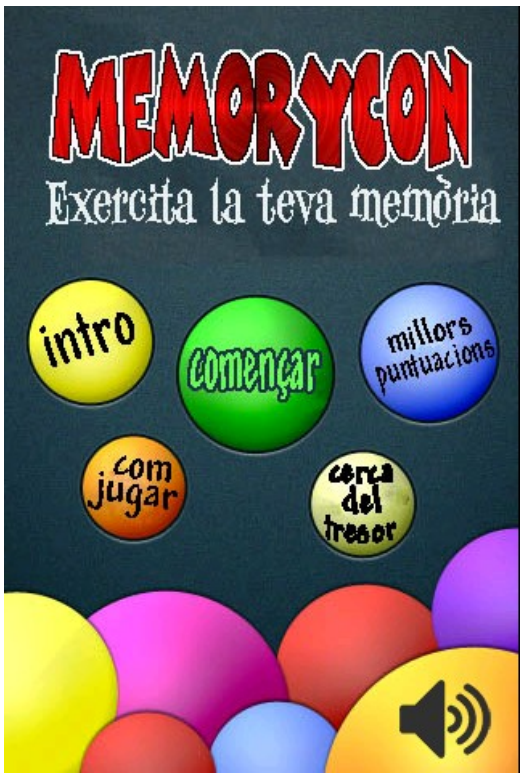
Exercici 5: pantalla de joc

Exercici 6: pantalla de joc

Exercici 7: pantalla d'introducció

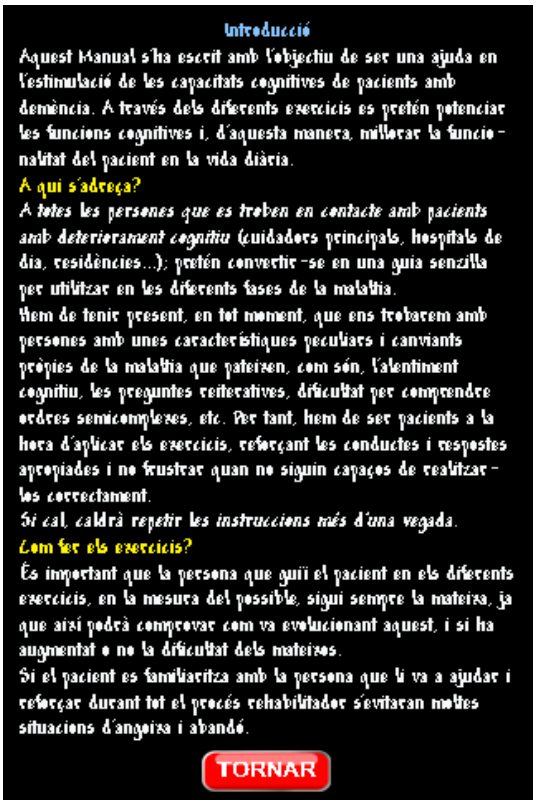
Exercici 7: pantalla de joc

A més, compta amb una pantalla principal...



...des de la qual podem accedir a:

- Una petita **introducció** al manual "Ejercicios para potenciar la memoria de los enfermos de Alzheimer" (vegeu bibliografia).



- Una breu explicació sobre **com jugar** les diferents activitats:

- La pantalla amb les **millors puntuacions**:

- La **pantalla del tresor**, on van apareixent els diferents objectes —en funció de les puntuacions assolides— que ens ajudaran en la seva recerca:

- El control de **so**:

- I finalment, prement a **Començar**, podem donar inici al joc, accedint a la primera activitat.

Un cop acabada la partida, tenim la pantalla de puntuació total...

Des de la qual, prement a *Finalitzar*, accedim a la pantalla de millors puntuacions.

2.1. Atles de textures

Cada cop que fem una crida a que es carregui una imatge a pantalla (*renderitzat*) consumim molta memòria de la GPU. El que se sol fer és agrupar diferents imatges en atlas de textures (també anomenats atles d'imatges, o *sprite sheets*).

Un atles de textures no és més que una gran textura que conté totes, o una gran part, de les nostres textures. La idea és que només haguem de carregar una textura, crear llavors totes les dades de cada objecte i en última instància utilitzar només una crida de renderitzat. D'aquesta manera el procés és molt més ràpid, i no es consumeix tanta memòria.

Malauradament, tan sols he fet servir *sprites* per a alguns botons i els diferents tipus de números (vegeu imatge esquerra). S'haurien d'haver fet servir també per als diferents components de la recerca del tresor, per als `bonus.png`, les icones d'encert i error, els `bravo.png`, i alguns botons addicionals. Com a resultat, a darrera hora l'aplicació ha experimentat alguns problemes de memòria (*vegeu apartat de conclussions*).

En el nostre cas, per tal de "retallar" els diferents *sprites*, fem crides als següents mètodes:

```
GameScreen.drawText(Graphics g, String line, Pixmap pix, int x, int y)
```

```
Graphics.drawPixmap(Pixmap pixmap, int x, int y, int srcX, int srcY, int srcWidth, int srcHeight);
```

3. Desenvolupament i implementació

El desenvolupament de *MemoryCon* s'ha dut a terme des de l'IDE **Eclipse Juno**, on s'ha instal·lat el **plugin ADT** (*Android Development Tools*). Aquest plugin permet aprofitar les eines de l'**Android SDK** (un kit de desenvolupament necessari per a programar i implementar tot tipus d'aplicacions per a Android) des de l'Eclipse. D'aquesta manera, i amb l'**AVD** (*Android Virtual Device*) **Manager**, podem anar provant la nostra aplicació des de la mateixa màquina virtual al nostre ordinador, sense necessitat d'instal·lar-la cada vegada en un dispositiu mòbil Android:

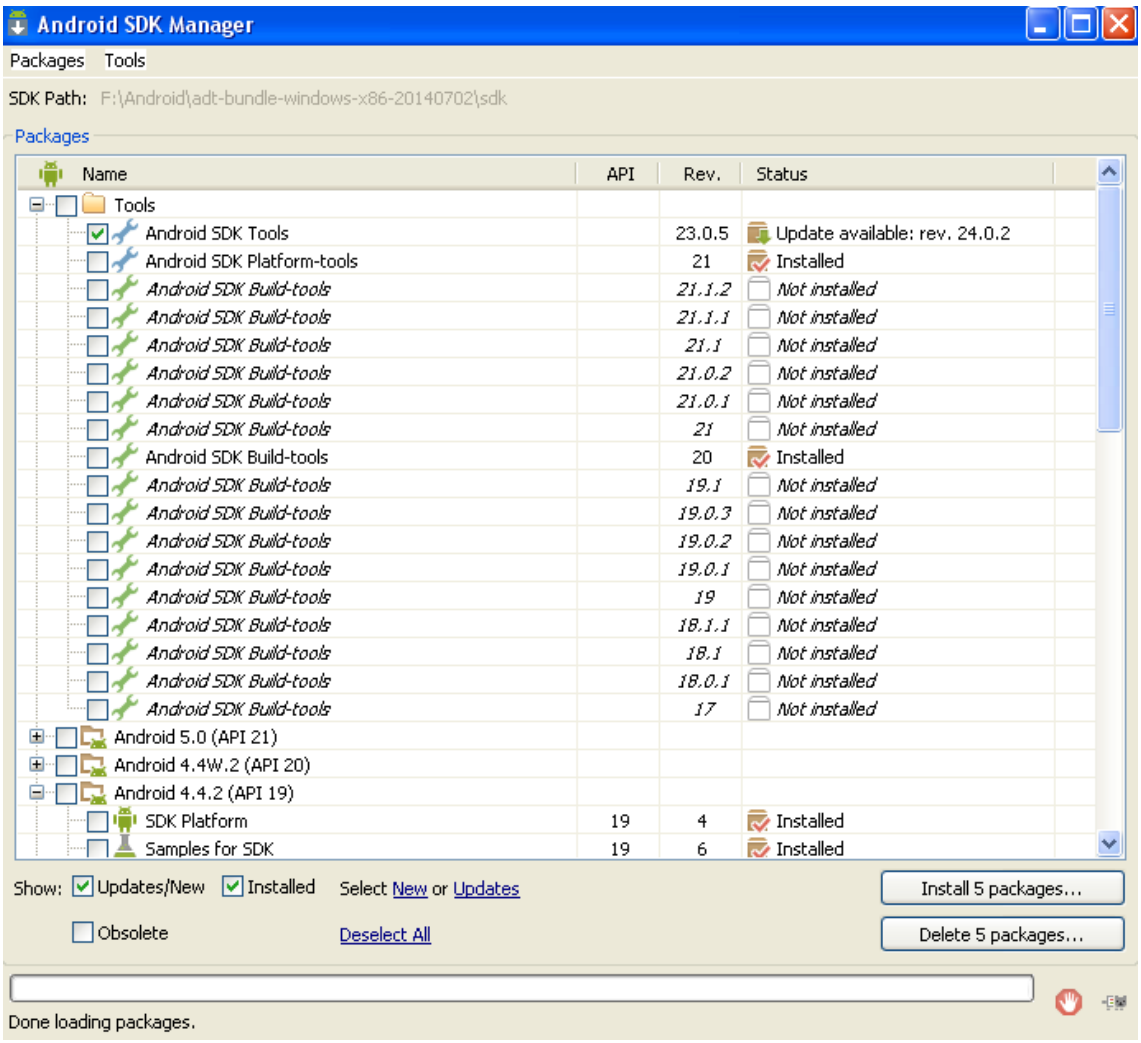


Figura 6.1: L'Android SDK Manager

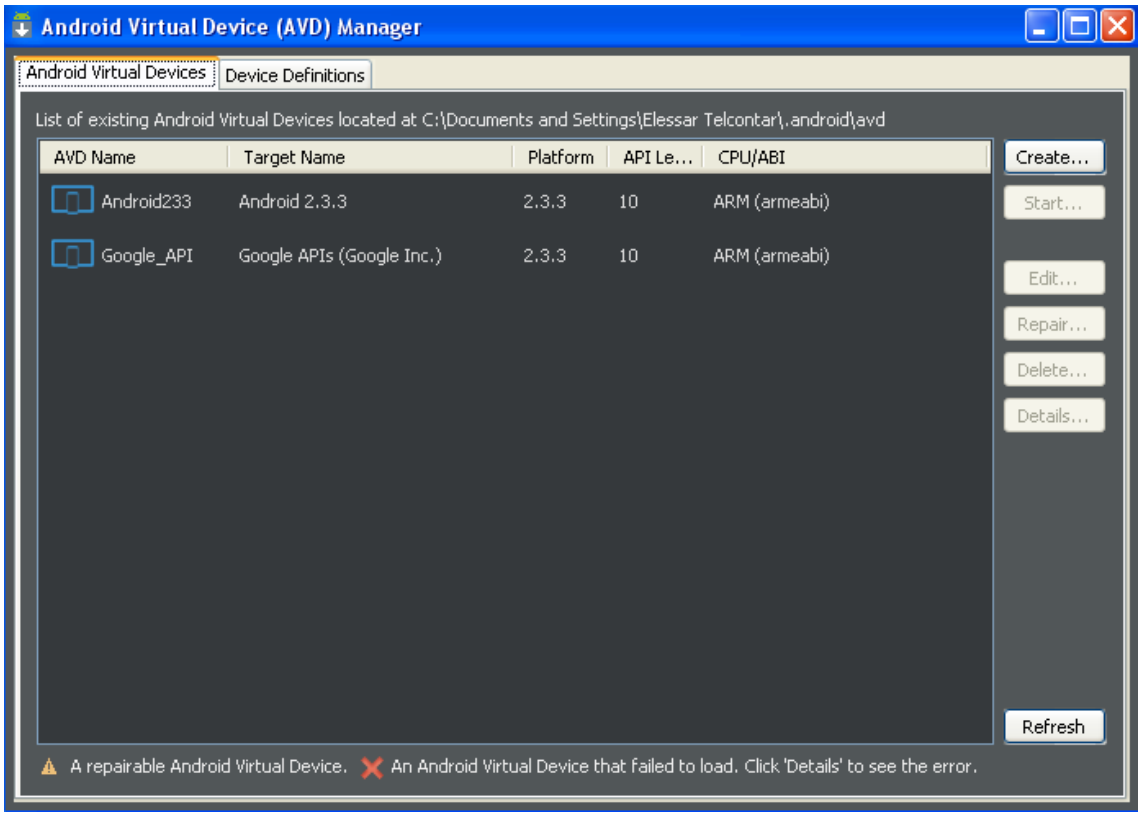


Figura 6.1: L'AVD Manager

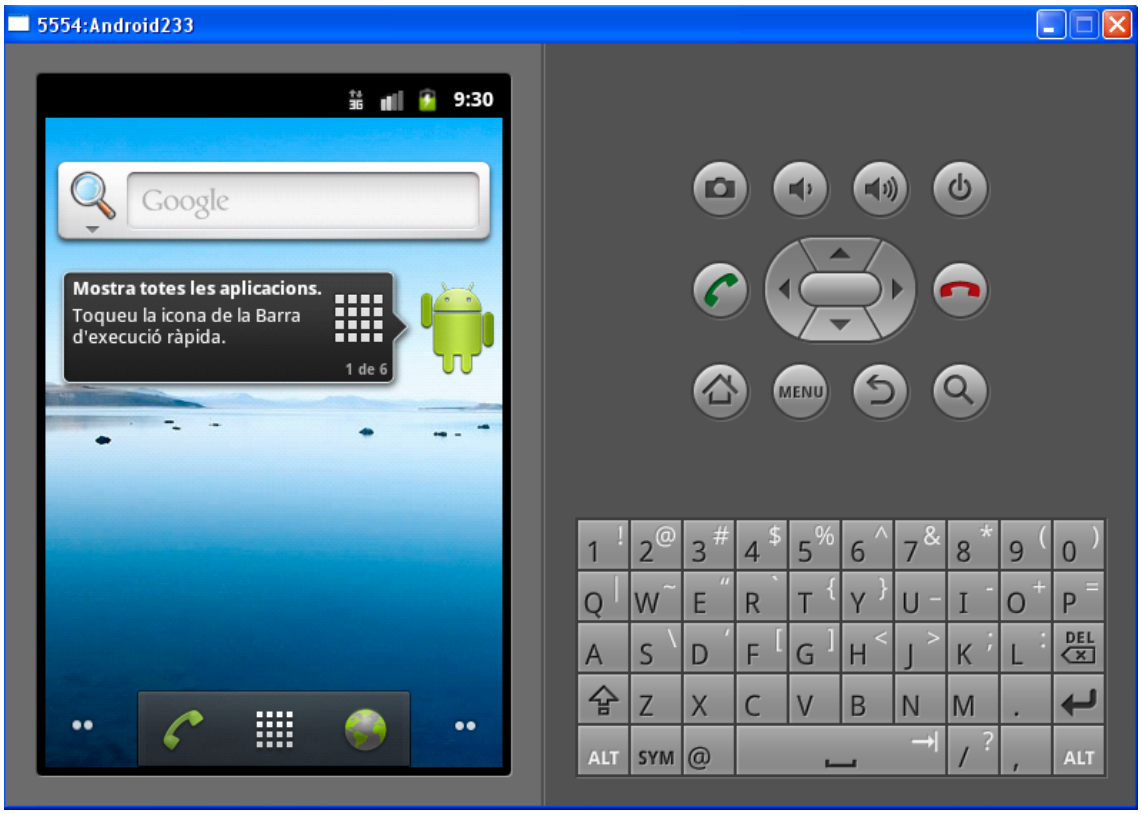


Figura 6.2: La màquina virtual

3.1. Estructura general

Al *Package Explorer* de l'Eclipse podem veure l'estructura del nostre projecte (***tfg***):

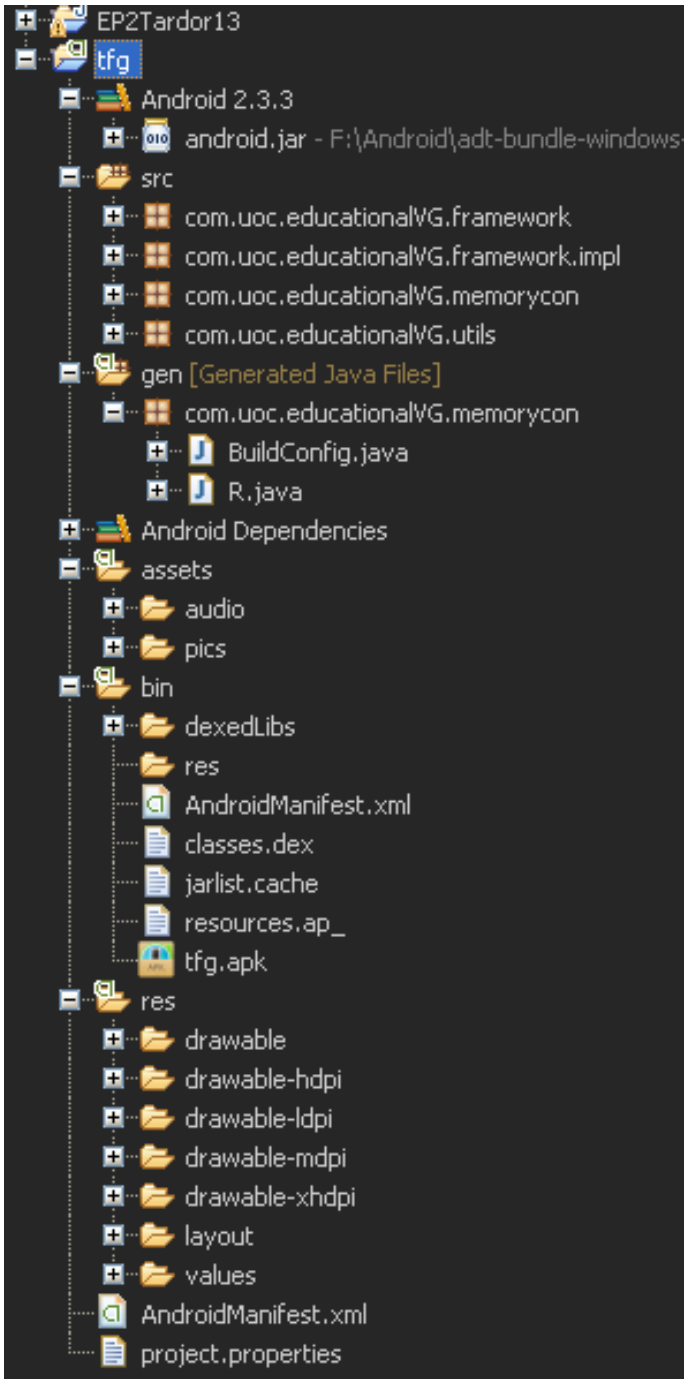


Figura 6.3: L'explorador de paquets d'Eclipse

Analitzem-lo una mica més en detall:

- `AndroidManifest.xml` descriu la nostra aplicació. Defineix quines activitats i serveis comprèn la nostra aplicació, a quina versió mínima d'Android es pot executar

(en teoria), i els permisos que necessita (per exemple, l'accés a la targeta SD o a Internet).

Una aplicació Android pot consistir en una multitud de **components** diferents:

- *Activities*: els components visibles; presenten una interfície d'usuari amb el qual interactuar. Poden estar en tres estats: *Running*, *Paused* i *Stopped*.

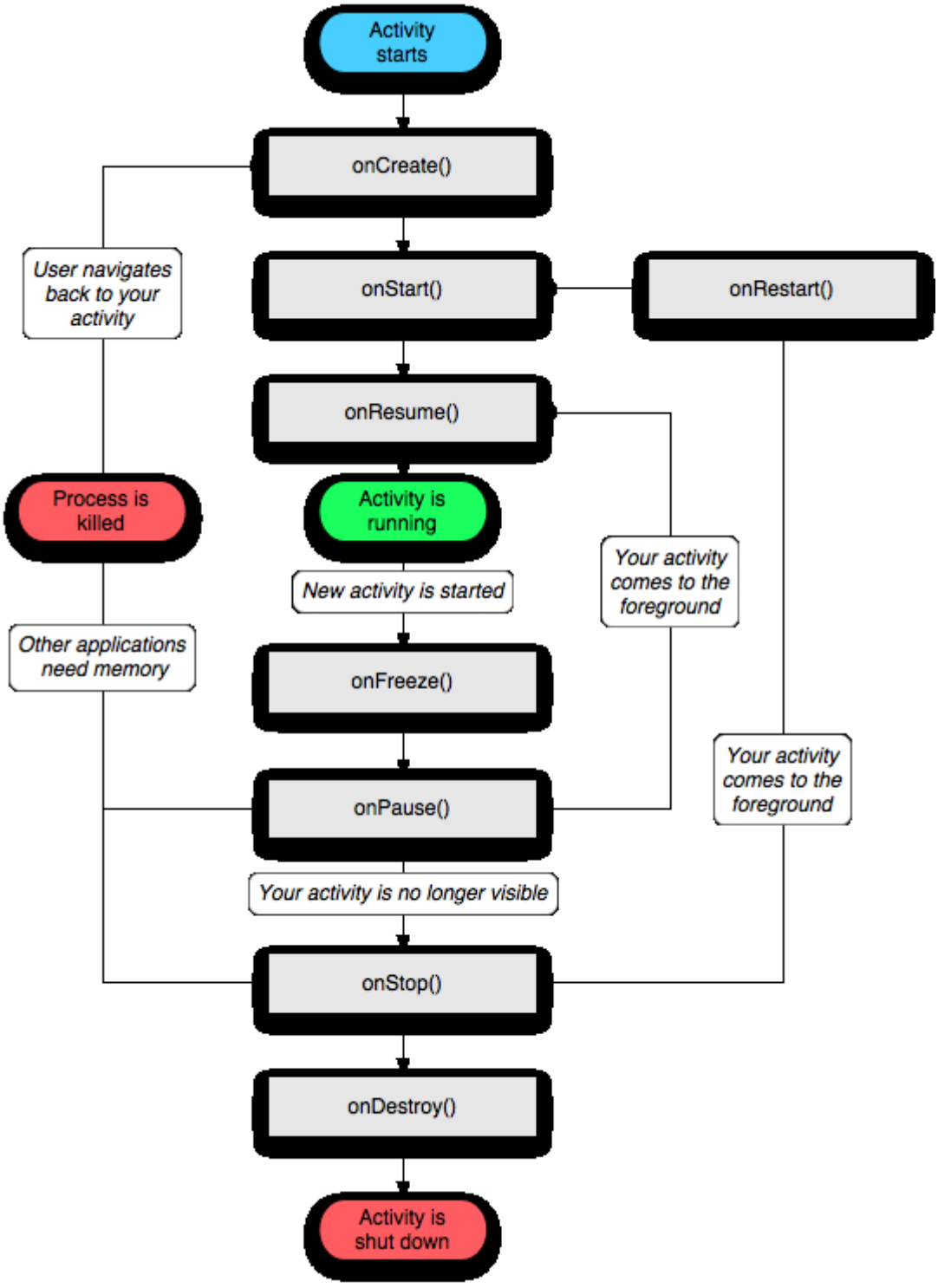


Figura 6.4: cicle de vida d'una Activity (font: stackoverflow.com)

- *Services*: Es tracta de processos que funcionen en segon pla i que no tenen una interfície d'usuari visible. Per exemple, un servei pot ser responsable de demanar a un servidor de correu si hi ha nous correus electrònics.

- *Content providers*: Aquests components fan que part de les dades de l'aplicació estiguin disponibles per a altres aplicacions.

- *Intents*: missatges creats pel sistema o per les mateixes aplicacions. A continuació, es transmeten a qualsevol part interessada. Els intents ens poden notificar certs esdeveniments del sistema, com ara que la targeta SD s'ha retirat, o que s'ha connectat un cable USB. Els *intents* també són utilitzats pel sistema per iniciar components de la nostra aplicació, com ara les *activities*. També podem generar els nostres propis *intents* de demanar a altres aplicacions que realitzin una acció, com obrir una galeria de fotos per mostrar una imatge o iniciar l'aplicació de càmera per fer una foto.

- *Broadcast receivers*: reaccionen a *intents* específics, i podrien executar una acció, com l'inici d'una activitat específica o l'enviament d'un altra *intent* al sistema.

El **fitxer de manifest** serveix per a molt més que un simple definició dels components d'una aplicació. La següent llista resumeix les parts rellevants d'un arxiu de manifest en el context del desenvolupament del joc:

- La versió de l'aplicació que apareix i s'utilitza en l'Android Market.

- Les versions d'Android en què la nostra aplicació es pot executar.

- Els perfils de maquinari que la nostra aplicació requereix (és a dir, *multitouch*, resolucions de pantalla específiques, o suport per a OpenGL ES 2.0).

- Permisos per a l'ús de components específics, com escriure a la targeta SD o l'accés a la pila d'Internet.

- `default.properties` té diversos ajustaments a nivell de sistema. No ens ha de preocupar, ja que el plugin ADT plugin s'encarregarà de modificar-lo quan sigui necessari.

- `src/` conté tots els arxius de codi Java. Tingueu en compte que el paquet té el mateix nom que el que s'ha especificat en l'assistent de projecte Android.

- `gen/` conté arxius de codi font Java generats pel sistema d'estructura Android. No s'han de modificar ja que, en alguns casos, es regeneren automàticament.

- `assets/` és on s'emmagatzemen tots els arxius que la nostra aplicació necessita (per exemple, arxius de configuració, arxius d'àudio, imatges, etc.).

- `res/` conté els recursos que necessita la nostra aplicació, com ara icones en diferents resolucions (per als diferents tamanys de dispositius Android) per als llançadors de la

nostra aplicació, *strings* per a la seva internacionalització, i *layouts* d'interfície d'usuari definits a través de XML.

- Android 2.3.3 ens diu que estem implementant sobre un *target* d'Android 2.3.3. Això és en realitat una dependència en la forma d'un arxiu JAR estàndard que allotja les classes de l'Android 1.5 API.

La vista de l'explorador de paquets amaga un altre directori, anomenat `bin/`, que conté el codi compilat i preparat per al seu desplegament a un dispositiu o emulador (l'arxiu APK).

3.2. Diagrama de classes

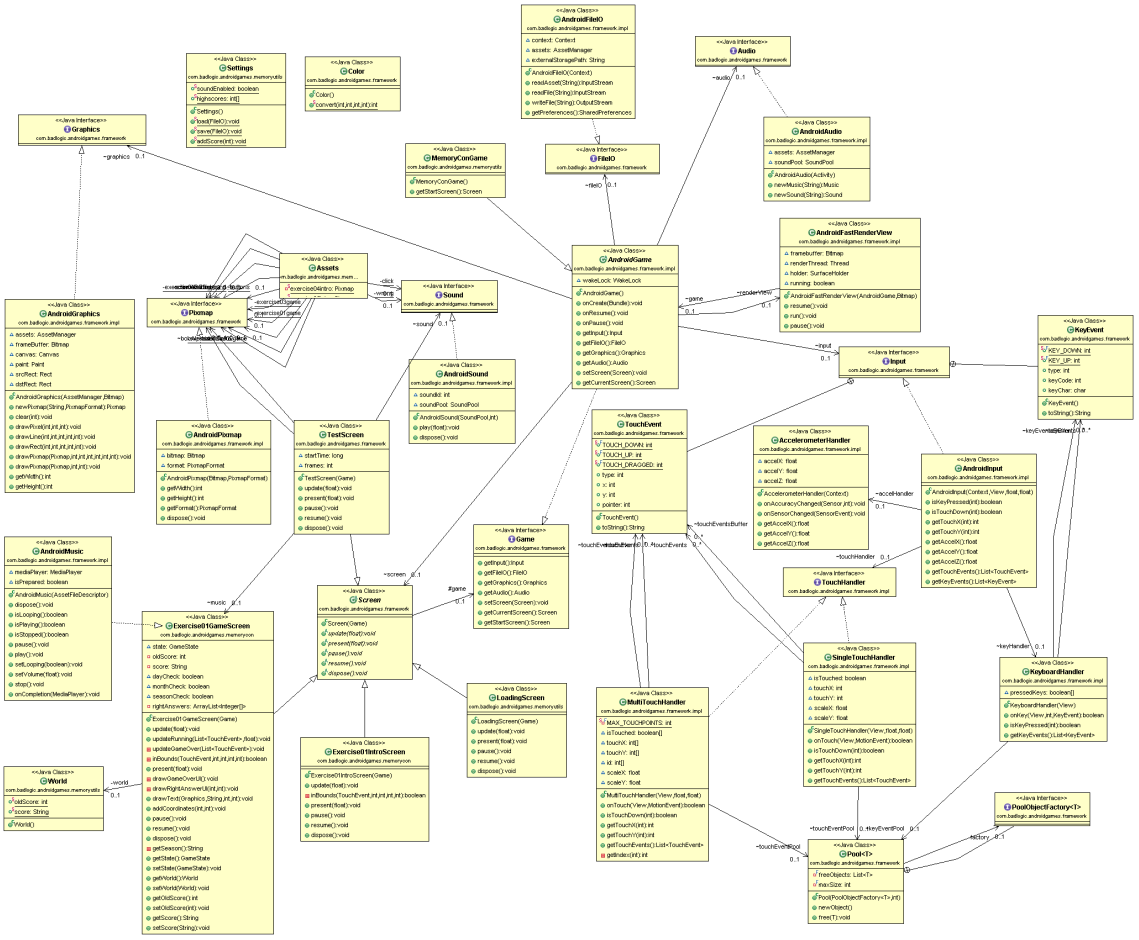
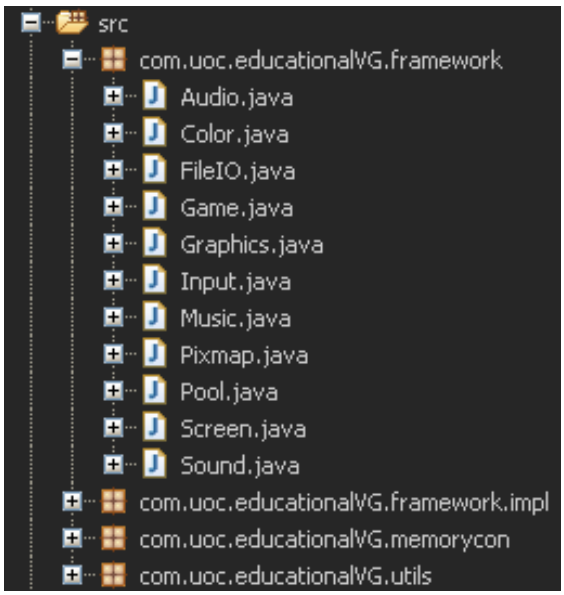


Figura 6.5: Diagrama de classes UML de l'aplicació MemoryCon

3.2.1. Anàlisi del bastiment (*framework*)

Primerament, analitzem les classes del nostre *framework* (obtingut del llibre *Beginning Android 4 Games Development*):



- `Input`

  Aquesta interfície està relacionada amb el mòdul de gestió de finestres. Es realitza un seguiment dels *inputs* de l'usuari (és a dir, quan toca la pantalla, les pulsacions de teclat, perifèrics, o les lectures de l'acceleròmetre). En Android, tenim tres mètodes d'entrada principals: pantalla tàctil, teclat/*trackball*, i acceleròmetre. La pantalla tàctil pot generar tres esdeveniments: *touch down* (quan un dit toca la pantalla), *touch drag* (el dit s'arrossega per la pantalla) i *touch up* (quan el dit s'aixeca de la pantalla).

- `FileIO`

  Volem un mecanisme senzill d'accés a arxius. Especifiquem un nom de fitxer i obtenim un *stream* (*InputStream* o *OutputStream*) a canvi. Els *assets* seran llegits des de l'arxiu APK de la nostra aplicació, i els arxius es poden llegir i escriure a la targeta SD (el dispositiu d'emmagatzematge extern).

- `Audio`

  Definim els següents **requisits**:

  - Necessitem una forma de **carregar arxius d'àudio** per a la reproducció en *streaming* i per a la reproducció des de la memòria.
  - Necessitem una manera de **controlar la reproducció d'àudio** en *streaming*.
  - Necessitem una manera de **controlar la reproducció d'àudio** totalment carregat.

  La interfície `Audio` ens permetrà crear noves instàncies de `Music` i `Sound`. Una instància de `Music` representa un arxiu d'àudio en *streaming*. Una instància de `Sound` representa un efecte de so curt que mantenim carregat completament en la memòria. Els mètodes `Audio.newMusic()` i `Audio.newSound()` prenen un nom de fitxer com a argument i llencen una `IOException` en cas que el procés de càrrega falli (per exemple, quan l'arxiu especificat no existeix o està malmès). Els noms d'arxiu es refereixen a arxius d'*assets* a l'arxiu APK de la nostra aplicació.

- `Color`

  El color (amb un model RGB).

A l'hora de dissenyar les interfícies per al nostre **mòdul de gràfics** (interfícies `Graphics` i `Pixmap`), i definir la seva funcionalitat, hem de tenir en compte que hem de ser capaços de realitzar les següents operacions:

- Carregar les imatges des del disc, i emmagatzemar-les en la memòria per a "dibuixar-les" més endavant.
- "Netejar" el *framebuffer* (una àrea reservada de la VRAM on s'emmagatzema cada píxel per a que sigui mostrat per pantalla) amb un color perquè puguem esborrar el que encara és allà des de l'últim *frame*. De fet, el que "esborrem" és el *framebuffer* de la interfície d'usuari, no el *framebuffer* "real".
- Situar un píxel al *framebuffer* en una ubicació específica, i amb un color específic.
- Dibuixar línies i rectangles al *framebuffer*.
- Dibuixar imatges carregades prèviament al *framebuffer*. Voldrem dibuixar bé la imatge completa, o bé parts d'ella. També hem de ser capaços de dibuixar imatges amb i sense barreja (*blending*).
- Obtenir les dimensions del *framebuffer*.

Així doncs,

- `Graphics`
  - El mètode `Graphics.newPixmap()` carrega una imatge donada en format JPEG o PNG. El `Pixmap` resultant pot tenir diferents formats (RGB565, ARGB4444) per a controlar el consum de memòria de la càrrega de les nostres imatges). El nom de fitxer especifica un *asset* a l'arxiu APK de la nostra aplicació.
  - El mètode `Graphics.clear()` esborra tot el *framebuffer* amb el `color` especificat. Tots els colors del nostre *framebuffer* seran especificats com a valors ARGB8888 de 32 bits.
  - El mètode `Graphics.drawPixel()` situa un píxel a (x, y) en el *framebuffer* amb el color especificat. Les coordenades fora de la pantalla són ignorades. D'això se'n diu *clipping*.
  - El mètode `Graphics.drawLine()` és anàleg al mètode `Graphics.drawPixel()`. Especifiquem el punt d'inici i el punt final de la línia, juntament amb un color. S'ignorarà qualsevol part de la línia que estigui fora de la trama del *framebuffer*.

- El mètode `Graphics.drawRect()` dibuixa un rectangle al *framebuffer*. La (x, y) especifica la posició de la cantonada superior esquerra del rectangle al *framebuffer*. Els arguments `width` i `height` especifiquen el nombre de píxels en x i y, i el rectangle s'omple a partir de (x, y). L'argument `color` és el color que s'utilitza per omplir el rectangle.

- El mètode `Graphics.drawPixmap()` dibuixa porcions rectangulars d'un Pixmap al *framebuffer*. Les coordenades (x, y) especifiquen la posició de la cantonada superior esquerra del `Pixmap` de destinació en el *framebuffer*. Els arguments `srcX` i `srcY` especifiquen la cantonada superior esquerra corresponent a la regió rectangular que s'utilitza des del `Pixmap`, determinada en el propi sistema de coordenades del `Pixmap`. Finalment, `srcWidth` i `srcHeight` especifiquen la mida de la porció que es pren des del `Pixmap`.

- Per últim, els mètodes `Graphics.getWidth()` i `Graphics.getHeight()` retornen l'amplada i l'altura del *framebuffer* en píxels.

Tots els mètodes de dibuix, excepte `Graphics.clear()`, realitzaran automàticament la barreja (*blending*) per a cada píxel.

- `Pixmap`

  - Els mètodes `Pixmap.getWidth()` i `Pixmap.getHeight()` retornen l'amplada i l'alçada del `Pixmap` en píxels.

  - El mètode `Pixmap.getFormat()` retorna el `PixelFormat` amb què el `Pixmap` s'emmagatzema a la memòria RAM.

- `Game`

  Una implementació d'aquesta interfície ha de:

  1) Configurar el component de la finestra i la interfície d'usuari i connectar-ho amb *callbacks* perquè puguem rebre esdeveniments de finestres i d'entrades.

  2) Iniciar el *thread* del bucle principal.

  3) Fer un seguiment de la pantalla actual, i dir-li que s'actualitzi i es presenti a si mateixa en cada iteració del bucle principal (cada *frame*).

  4) Transferir els esdeveniments de finestra (per exemple, fer una pausa o reprendre) des del *thread* d'interfície d'usuari fins al *thread* del bucle principal i passar-los a la pantalla actual perquè aquesta pugui canviar el seu estat en conseqüència.

  5) Donar accés a tots els mòduls que hem desenvolupat anteriorment: `Input`, `FileIO`, `Graphics` i `Audio`.

- El mètode `Game.setScreen()` ens permet fixar la pantalla actual del joc. Aquests mètodes s'executaran una vegada, juntament amb tota la creació interna del *thread*, la gestió de finestres, i la lògica de bucle principal que demanarà constantment a la pantalla actual que es presenti i s'actualitzi.

- El mètode `Game.getCurrentScreen()` retorna la pantalla activa en aquell moment.

Més endavant utilitzarem una classe abstracta anomenada `AndroidGame` per implementar la interfície `Game`. Aquesta classe abstracta implementarà tots els mètodes excepte `Game.getStartScreen()`. Aquest mètode serà un mètode abstracte. Si creem la instància d'`AndroidGame` per al nostre joc, l'estrendrem i reemplaçarem el mètode `Game.getStartScreen()`, retornant una instància a la primera pantalla del nostre joc. `AndroidGame` es deriva de `Activity`, que serà instanciada automàticament pel sistema operatiu Android, quan un usuari iniciï el nostre joc.

- `Screen`

`Screen` és una classe abstracta en lloc d'una interfície; així podrem implementar-la sense haver de repetir massa codi.

```
public abstract classScreen {
      protected final Game game;
      public Screen(Game game) {
      this.game = game;
      }
      public abstract void update(floatdeltaTime);
      public abstract void present(floatdeltaTime);
      public abstract void pause();
      public abstract void resume();
public abstract void dispose();
}
```

En aquesta classe, el constructor rep la instància de `Game` i l'emmagatzema en un atribut `final` al qual es pot accedir des de totes les subclasses. A través d'aquest mecanisme, aconseguim dues coses:

- Podem accedir als mòduls de baix nivell de `Game` per tal de reproduir àudio, dibuixar a la pantalla, obtenir els *inputs* de l'usuari, i llegir i escriure arxius.

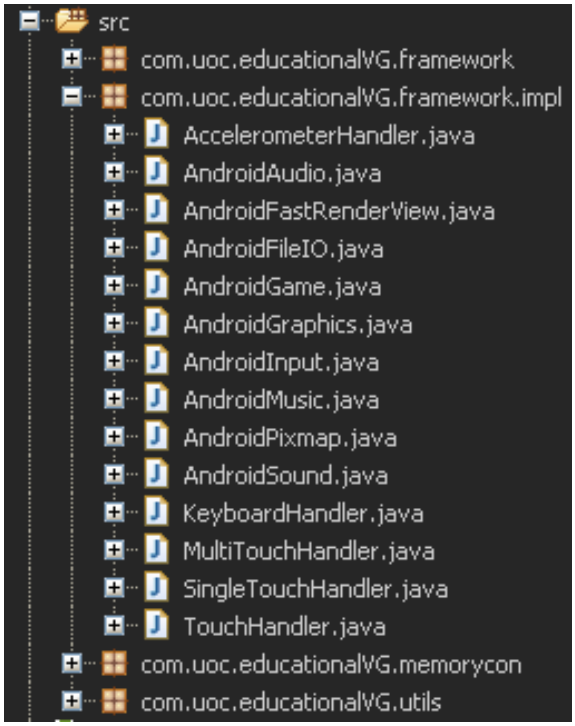
- Podem establir una nova `Screen` actual tot invocant `Game.setScreen()` quan convingui (per exemple, quan es prem un botó que activa una transició a una nova pantalla). Això ens permetrà fer transicions entre pantalles.

Els mètodes `Screen.update()` i `Screen.present()` actualitzen l'estat de la pantalla i la presenten d'acord amb l'estat actual, respectivament. La instància de `Game` els cridarà una vegada per a cada iteració del bucle principal.

Els mètodes `Screen.pause()` i `Screen.resume()` seran cridats quan el joc es posi en pausa o es reprengui. Això, de nou, és dut a terme per la instància de `Game` i s'aplica a la pantalla activa en aquell moment. *Nota: finalment no ha calgut fer servir aquests mètodes, a excepció d'un pause() a Exercise07GameScreen.java, per qüestions de persistència).*

El mètode `Screen.dispose()` serà cridat per la instància de `Game` en cas que es cridi a `Game.setScreen()`. La instància de `Game` es desfarà de la pantalla actual a través d'aquest mètode i per tant donarà a `Screen` una oportunitat per alliberar tots els recursos del sistema (per exemple, *assets* de gràfics emmagatzemats en `Pixmaps`) per fer espai per als recursos de la nova pantalla a la memòria. La crida al mètode `Screen.dispose()` és també l'última oportunitat per a una pantalla per assegurar-se que es guardarà tota la informació que necessiti persistència.

3.2.2. Implementació de les interfícies del bastiment



- `AndroidFileIO`

  Aquesta classe implementa la interfície `FileIO`, la qual, recordem, contenia quatre mètodes: un per aconseguir un `InputStream` per a un *asset*, un altre per obtenir un `InputStream` per a un arxiu en l'emmagatzematge extern, un tercer que retorna un

`OutputStream` per a un arxiu en el dispositiu d'emmagatzematge extern, i un darrer que aconsegueix les `SharedPreferences` per al joc .

Així doncs, implementem la interfície `FileIO`, emmagatzemem el `Context` (que és la porta d'entrada a gairebé tot en Android: des de crear nous objectes, fins a accedir implícitament a components, passant per l'accés a recursos estàndard), emmagatzemem un `AssetManager`, el qual obtenim del `Context`, guardem la ruta de l'emmagatzematge extern, i implementem els quatre mètodes basats en aquesta ruta. Finalment, es passa a través de les `IOExceptions` per saber si hi ha alguna cosa irregular en les crides.

La nostra implementació de la interfície `Game` contindrà una instància de `AndroidFileIO` i la retornarà a través de `Game.getFileIO ()`. Això també vol dir que la nostra implementació de `Game` haurà de passar a través del `Context` perquè la instància `AndroidFileIO` es posi a treballar.

No comprovem si l'emmagatzematge extern disponible. Si no està disponible, o si ens oblidem d'afegir el permís adequat a l'arxiu de manifest, obtindrem una excepció, de manera que la comprovació d'errors és implícita.

- `AndroidAudio`

La implementació de la interfície `Audio` té un `AssetManager` i una instància de `Soundpool`. L'`AssetManager` és necessari per a carregar els efectes de so des dels arxius d'assets cap al `SoundPool` en una crida a `AndroidAudio.newSound()`. La instància d'`AndroidAudio` també administra el `Soundpool`.

Hi ha dues raons per les quals passem l'`Activity` del nostre joc en el constructor: d'una banda, ens permet establir el control de volum, i de l'altra ens dóna una instància d'`AssetManager`, que emmagatzemarem en el corresponent atribut de classe. El `SoundPool` està configurat per reproduir 20 efectes de so en paral·lel, la qual cosa s'adequa a les nostres necessitats.

El mètode `newmusic()` crea una nova instància `AndroidMusic`. El constructor de la classe pren un `AssetFileDescriptor`, que utilitza per crear un `MediaPlayer` intern. El mètode `AssetManager.openFd()` llança una `IOException` en cas que alguna cosa vagi malament, l'atrapem i la rellancem com una `RuntimeException`.

Finalment, el mètode `NewSound()` carrega un efecte de so d'un asset en el `SoundPool` i retorna una instància `AndroidSound`. El constructor d'aquesta instància pren un `SoundPool` i l'ID de l'efecte de so que li assigna el `AndroidSound`. De nou, llancem una `IOException` en cas que alguna cosa vagi malament, l'atrapem i la rellancem com una `RuntimeException`.

- AndroidSound

  A través dels mètodes `play()` i `dispose()`, simplement emmagatzemem el `SoundPool` i l'ID de l'efecte de so carregat per a la seva posterior reproducció i eliminació. Gràcies a l'API d'Android, aquesta implementació és senzilla.

- AndroidMusic

  La classe `AndroidMusic` emmagatzema una instància `MediaPlayer` juntament amb un atribut booleà (`isPrepared`), ja que només podrem cridar a `MediaPlayer.start()` / `stop()` / `pause()` quan el `MediaPlayer` estigui preparat.

  La classe `AndroidMusic` implementa les interfícies `Music` i `OnCompletionListener`. Aquesta darrera interfície és un mitjà per saber quan en un `MediaPlayer` s'ha aturat la reproducció d'un arxiu de música. Si això succeeix, el `MediaPlayer` necessita estar preparat de nou abans que puguem invocar qualsevol dels altres mètodes. El mètode `onCompletionListener.onCompletion()` podria ser cridat des d'un *thread* separat, i ja que col·loquem l'atribut `isPrepared` en aquest mètode, cal assegurar-se que està a resguard de modificacions simultànies.

  En el constructor, creem i preparem el `MediaPlayer` a partir de l'`AssetFileDescriptor` que se li passa, i activem el *flag* `isPrepared`, al mateix temps que registrem la instància `AndroidMusic` com un `OnCompletionListener` amb el `MediaPlayer`. Si alguna cosa surt malament, llancem una `RuntimeException`.

  Els mètode `dispose`() comprova si el `MediaPlayer` encara està reproduïnt i, si és així, el deté. En cas contrari, la crida a `MediaPlayer.release()` llançarà una excepció de temps d'execució.

  Els mètodes `isLooping()`, `isPlaying()` els proporciona la classe `MediaPlayer`. El mètode `isStopped()` utilitza l'indicador `isPrepared`, que indica si s'ha aturat el `MediaPlayer`.

  El mètode `pause()` simplement comprova si la instància `MediaPlayer` està reproduïnt; en cas afirmatiu, crida al seu mètode `pause()`.

  El mètode `play()` és una mica més complicat. Si ja estem reproduïnt, simplement fem un *return* de la funció. A continuació tenim bloc *try-catch* en què comprovem si el `MediaPlayer` ja està preparat sobre la base del nostre *flag*; si convé, el preparem. Si tot va bé, cridem al mètode `MediaPlayer.start()`, que iniciarà la reproducció. Això es porta a terme en un bloc sincronitzat, ja que estem utilitzant el *flag* `isPrepared`, que podria activar-se en un *thread* independent perquè estem implementant la interfície `OnCompletionListener`. En cas que alguna cosa vagi malament, es llença una `RuntimeException`.

Els mètodes `setLooping`() i `setVolume`() poden ser cridats en qualsevol estat del MediaPlayer i delegats llavors als mètodes respectius de `MediaPlayer`.

El mètode `stop()` atura el `MediaPlayer` i estableix l'indicador `isPrepared` en un bloc sincronitzat.

Finalment, el mètode `OnCompletionListener.onCompletion()` estableix el *flag* `isPrepared` en un bloc sincronitzat perquè els altres mètodes no comencin a llançar excepcions del no-res.

- `Pool`

  Els mètodes `getTouchEvents()` i `getKeyEvents()` retornen llistes de `TouchEvents` i `KeyEvents` llistes. En els nostres gestors d'esdeveniments de teclat i tàctils, constantment es creen instàncies d'aquestes dues classes i s'emmagatzemen en llistes que són internes als gestors. El sistema d'entrada d'Android llença molts d'aquests esdeveniments quan es prem una tecla o un dit toca la pantalla, de manera que constantment es creen noves instàncies que són recollides pel *garbage collector* (recol·lector d'escombraries) a intervals curts. Per evitar això, implementem un concepte conegut com a *instance pooling*. En lloc de crear repetidament noves instàncies d'una classe, simplement reutilitzem instàncies creades prèviament. La classe `Pool` és una manera d'implementar aquest comportament.

- `AccelerometerHandler`, `KeyboardHandler`, `TouchHandler`, `MultiTouchHandler`, `SingleTouchHandler`

  Gestionen l'acceleròmetre, el teclat, i la pantalla tàctil.

- `AndroidInput`

  Implementa la interfície `Input`. Coordina els diferents *inputs*, delegant al *handler* corresponent.

- `AndroidGraphics`

  Implementa la interfície `Graphics`. Dibuixarà píxels, línies, rectangles i `Pixmaps` al *framebuffer*. Farem servir un `Bitmap` com a *framebuffer* i dirigirem totes les crides dibuix al Bitmap a través d'un Canvas.

  També és responsable de la creació d'instàncies de `Pixmap` a partir d'arxius d'*assets*. Per tant, també necessitarem un altre `AssetManager`.

- `AndroidFastRenderView`

  Aquesta classe deriva de la classe `SurfaceView`, permet renderitzar contantment la `SurfaceView` a través d'un `Canvas`. A més:

- Manté una referència a una instància de `Game` a partir de qual pot obtenir la Screen activa. Constantment fem crides als mètodes `Screen.update()` i `Screen.present()` des del thread del `FastRenderView`.
- Es realitza un seguiment en temps real entre *frames*, el qual es passa a la pantalla activa.

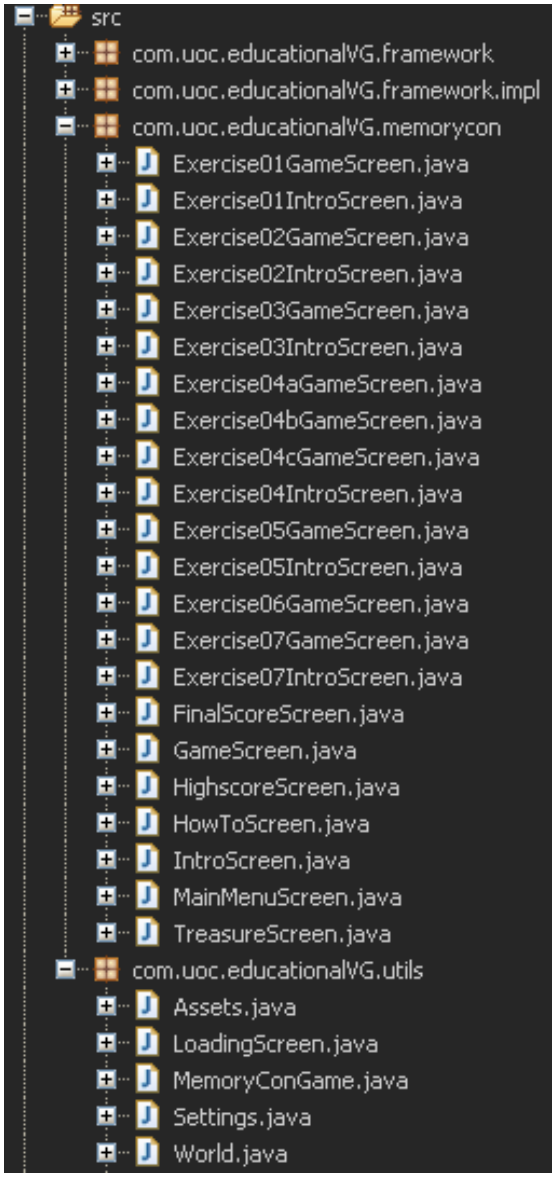
Necessitem aquest *framebuffer* artificial perquè la instància de `AndroidGraphics` dibuixi, i ho faci a la `SurfaceView`, la qual s'escala si és necessari.

- `AndroidGame`

Aquesta classe, que implementa la interfície `Game`, acabarà de lligar-ho tot, fent servir les classes creades prèviament. Aquesta és una llista de les seves responsabilitats:

- Dur a terme la gestió de finestres. En el nostre context, això significa la creació d'una activitat i una `AndroidFastRenderView`, i el maneig del cicle de vida de l'activitat d'una manera neta.
- Utilitzar i gestionar un `WakeLock` manera que la pantalla no s'atenuï.
- Crear instàncies i lliurar referències a `Graphics`, `Audio`, `FileIO` i `Input` a les parts interessades.
- Administrar pantalles i integrar-les amb el cicle de vida de l'activitat.
- El nostre objectiu general és tenir una sola classe anomenada `AndroidGame` de la qual podem derivar. Volem implementar el mètode `Game.getStartScreen()` més tard per començar el nostre joc de la següent manera.

3.2.3. Utilitats i pantalles de joc



- `MemoryConGame`

  L'`Activity` principal. La nostra aplicació necessita un punt d'entrada principal, també coneguda com l'activitat per defecte en Android. Fem una crida a aquesta activitat `MemoryConGame` i deixem que es derivi d'`AndroidGame`, la classe que hem implementat per executar el nostre joc, i que més endavant serà la responsable de la creació i execució de la nostra primera pantalla.

  El mètode `getStartScreen()` ens tornarà una instància de la classe `LoadingScreen`.

- `Assets`

  Des de la `LoadingScreen` carreguem tots els *assets* del nostre joc. Però, on els guardem? Per emmagatzemar-los, crearem una classe amb un munt de membres estàtics que comprenen tots els `Pixmaps`, `Sounds` i `Musics`.

- `Settings`

  Permetem l'usuari activar o desactivar el so. Des de `Settings` farem que tant aquesta elecció com el registre de puntuacions més altes quedin guardades (mètode `save()`) de manera persistent a la memòria externa (la targeta SD). Així, quan es torni a executar el joc una altra vegada, el dispositiu "recordarà" (mètode `load()`) aquests ajustaments.

  A més, disposem del mètode `addScore()` que afegeix una nova puntuació a les cinc puntuacions més altes, classificant-les de nou en funció del valor que s'afegeixi.

- `LoadingScreen`

  Des d'aquesta classe, que deriva de la classe `Screen` —i que per tant conté un constructor que pren una instància de `Game`—, carreguem en memòria els diferents *assets*.

  Val a dir que ja cap al final de la implementació de tota l'aplicació, vaig tenir problemes de memòria a la càrrega d'algunes imatges. Això es podia haver evitat fent servir de manera més adequada —i extensa— els diferents atles de *sprites* (vegeu apartat 5.1). Per una qüestió de temps, vaig solucionar-ho reduint els RGB565 a ARGB4444, malgrat la pèrdua de resolució.

- `World`

  Aquesta classe tan sols existeix per a guardar les variables `oldScore` i `score`. Malgrat no ser una bona pràctica, m'he vist obligat a fer-les globals (l'aplicació no anava bé només a base de *getters* i *setters*).

- `Exercise*GameScreen`, `Exercise*IntroScreen`, `MainMenuScreen`, `etc.`

  Les subclasses corresponents a les diferents pantalles del joc. Deriven de la superclasse `GameScreen`, la qual té els següents mètodes comuns:

  - `inBounds()`: comprova si l'usuari ha situat el dit a la zona correcta de la pantalla.
  - `drawText()`: dibuixa números a la pantalla, estrets dels atles de *sprites* `numbers.png`, `numbers2.png` i `numbersDigital.png` (per a la puntuació, els *highscores* i el marcador de temps, respectivament).
  - `getBonus()`: calcula els punts de bonificació, en funció del temps emprat en resoldre l'exercici i la dificultat d'aquest.
  - `addRightAnswers()`: afegeix les coordenades (x, y) a un `ArrayList` per tal de dibuixar posteriorment (amb el mètode `drawRightAnswerUI()`) els `ok.png` a cada resposta correcta.

  - `addRightAnswers()`: afegeix les coordenades (x, y) a un `ArrayList` per tal de dibuixar posteriorment (amb el mètode `drawWrongAnswerUI()`) els `notOk.png` a cada resposta errònia.

  Hi ha altres mètodes propis de les classes corresponents a cada exercici, però són prou autoexplicatius (vegeu codi).

4. Conclusions

Desenvolupar aquest projecte m'ha permès aprofundir força en el desenvolupament de jocs per a Android, no només a nivell de programació, sinó també en el disseny gràfic.

El fet d'haver pogut basar-me en el codi del llibre "Beginning Android 4 Games Development" ha estat crucial; altrament no crec que hagués pogut assolir els objectius inicials en aquest període de temps, i més tenint en compte el meu curt bagatge quant a programació Java es refereix.

D'altra banda, poder anar fent proves amb la màquina virtual d'Android és un gran avantatge. No només pel temps que s'estalvia, sinó perquè en combinació amb la perspectiva DDMS i el *LogCat* d'Eclipse (una vista que s'hi afegeix després d'instal·lar el plugin ADT), i tot comprovant el *log* d'errors, l'anàlisi i revisió del codi resulten més senzills.

4.1. Possibles ampliacions

D'haver tingut més temps, probablement hagués introduït alguna pantalla addicional d'exercicis. També m'agradaria fer un menú inicial d'activitats, de tal manera que no s'hagi de fer tot el joc sencer cada vegada, la qual cosa podria arribar a fer-se feixuga.

Quant als problemes de renderització, ja ha quedat clar que en una futura versió caldria aprofitar al màxim els atles de textures. I no només per qüestions de rendiment: crec que mitjançant el disseny de nous *sprites* podria arribar a fer efectes d'animació més aconseguits (per exemple, a la pantalla dels peixos, en lloc de dibuixar una icona de vist i plau verda, podria aconseguir l'efecte d'un peix saltant, amb dues o tres imatges diferents, i tornar a la seva posició inicial; o bé a la pantalla del tresor, on per a cada nou objecte que aparegui es podria afegir un efecte de so).

També caldria homogeneïtzar una mica més les pantalles de joc: per una qüestió d'espai, de vegades m'he vist forçat a traslladar la posició d'alguns botons o icones d'unes pantalles respecte d'altres (per exemple, a l'exercici 7).

5. Eines utilitzades

Planificació:

- Microsoft Project Professional 2010

Disseny gràfic:

- Adobe Photoshop CS3

Programació / implementació:

- IDE Eclipse Juno
- Android Development Tools
- Android SDK
- Java SE Development Kit 8

Edició de vídeo:

- Camtasia Studio 8
- BB FlashBack Express

Redacció de documents:

- Microsoft Office Word 2007

L'aplicació ha estat provada sobre una tauleta ePad de 7'' amb Android 1.6 i un telèfon mòbil Samsung Galaxy Mini amb Android 4.1.2.

6. Glossari

.apk Arxiu del paquet d'aplicacions per a Android (*Android Application Package File)*. Cada aplicació Android està compilada i empaquetada en un sol arxiu que inclou tots els arxius del codi de l'aplicació (arxius *.dex*), recursos, *assets* (actius) i l'arxiu de manifest (*Android manifest file*). Així doncs, un *.apk* és qualsevol applicació que podrem instal·lar en el nostre dispositiu.

.dex Arxiu compilat del codi de l'aplicació Android. Els programes per a Android es compilen en arxius .dex (*Dalvik Executable*), que són comprimits al seu torn en un únic arxiu .apk al dispositiu. Els arxius .dex es poden crear mitjançant la traducció automàtica d'aplicacions compilades escrits en el llenguatge de programació Java.

acció (*action*) Una descripció d'alguna cosa que un remitent vol que un *intent* faci. Una acció és un valor de cadena assignat a un *intent*. Els strings de les Action poden ser definides per Android o per un altre desenvolupador. Per exemple, `android.intent.action.VIEW` per a una adreça web URL, o `com.example.rumbler.SHAKE_PHONE` per a una aplicació personalitzada perquè vibri el telèfon.

acceleròmetre sensor que mesura l'acceleració i la forces induïdes per les gravetat.

activitat (*activity*) Representa una pantalla en una aplicació, amb suport de codi Java, derivada de la classe `Activity`. En general, una activitat està representada de manera visible per una finestra de pantalla completa que pot rebre i gestionar els esdeveniments d'interfície d'usuari i realitzar tasques complexes, a causa de la finestra (`Window`) que utilitza per a representar la seva finestra. Una `Activity` també pot ser flotant o transparent.

adb (*Android Depuration Bridge*) Una aplicació de depuració de línia d'ordres que s'inclou amb l'SDK d'Android. Proporciona eines per a navegar pel dispositiu, eines de còpia en el dispositiu, i redirecció de ports per a la depuració. Si desenvolupem en Eclipse utilitzant el plugin ADT, adb s'integra en l'entorn de desenvolupament.

Android manifest Vegeu **arxiu de manifest**.

aplicació Des d'una perspectiva de components, una aplicació Android consta d'una o més activitats, serveis, oients, i receptors d'*intents*. Des d'una perspectiva de l'arxiu de codi font, una aplicació per Android es compon de codi, recursos, actius (*assets*), i un manifest únic. Durant la compilació, aquests arxius estan empaquetats en un sol arxiu anomenat arxiu de paquet d'aplicació (.apk).

ART *Android Runtime*, o temps d'execució d'Android. És el responsable d'executar les aplicacions en Android. ART és el successor de *Dalvik Runtime*.

arxiu de manifest Un arxiu XML que cada aplicació ha de definir, per descriure el nom del paquet d'aplicació, versió, components (activitats, filtres *intent*, serveis), les biblioteques importades, les diverses activitats, etc.

atles de textures Una gran imatge que conté una col·lecció, o "atles", de sub-imatges, cadascuna de les quals és una textura a una part d'un objecte 2D o 3D. Les sub-textures es poden representar mitjançant la modificació de les coordenades de textura de l'objecte en els atles. En una aplicació on s'utilitzen moltes petits textures, sovint és més eficient emmagatzemar les textures en un atles que serà tractat com una sola unitat pel maquinari de gràfics. També anomenats *sprite sheets* o atles d'imatges.

Dalvik Màquina virtual de la plataforma Android. Només executa arxius en el format *Dalvik executable* (.dex). La màquina virtual d'Android per defecte fins a l'arribada d'ART.

DDMS *Dalvik Debug Monitor Service*, una aplicació GUI de depuració que s'inclou amb l'SDK. Proporciona captures de pantalla, bolcat de registre, i capacitats d'anàlisi del procés. Disponible a l'Eclipse si s'instal·la el plugin ADT.

Drawable Un recurs visual compilat que pot ser utilitzat com un fons, títol, o una altra part de la pantalla. Els recursos Drawable es compilen com a subclasses de `android.graphics.drawable`.

framebuffer S'anomena *framebuffer* a una categoria de dispositius gràfics, que representen cada un dels píxels de la pantalla com ubicacions a la memòria d'accés aleatori. També se l'anomena així en l'àrea dels sistemes operatius, als dispositius que fan servir o aparenten usar aquest mètode d'accés a dispositius gràfics.

intent Un objecte missatge que es pot utilitzar per posar en marxa o comunicar-se amb altres aplicacions o activitats de forma asíncrona.

intent filter Un objecte de filtre que l'aplicació declara en el seu arxiu de manifest, per indicar al sistema quins tipus d'*Intents* cadascun dels seus components està disposat a acceptar i amb quins criteris.

layout Un arxiu XML que descriu el disseny d'una pantalla d'`Activity`.

llenç (*canvas*) Una superfície de dibuix que s'encarrega de la composició dels bits contra un objecte `BitMap` o `Surface`. Té mètodes de dibuix estàndard de mapes de bits, línies, cercles, rectangles, textos, etc., i està vinculat a un `Bitmap` o `Surface`. Canvas és la forma més senzilla i fàcil de dibuixar objectes 2D a la pantalla. Però no és compatible amb l'acceleració de maquinari, al contrari que OpenGL ÉS. La classe base és `Canvas`.

Logcat Un llistat dels missatges emesos pel telèfon. Molt útil per als programadors per tal de trobar quina és la causa dels errors.

proveïdor de continguts (*content provider*) Una capa d'abstracció de dades que es pot utilitzar per exposar de forma segura les dades de la nostra aplicació a altres aplicacions. Es basa en la classe `ContentProvider`.

receptor broadcast (*broadcast receiver*) Un broadcast és un missatge que qualsevol aplicació pot rebre. A Android es poden utilitzar receptors per interceptar aquests missatges.

recursos Components no programables de l'aplicació que són externs al codi compilat de l'aplicació, però que es poden carregar des del codi d'aplicació utilitzant un format de referència coneguda. Per exemple: *strings* d'interfície d'usuari, components de *layout* de la interfície d'usuari, gràfics o altres arxius multimèdia. Els recursos d'una aplicació s'emmagatzemen sempre en les subcarpetes `res/*` del projecte.

renderització (*rendering*) Terme usat en argot informàtic per referir-se al procés de generar una imatge a partir d'un model 2D o 3D.

servei Un objecte de la classe `Service` que s'executa en segon pla (sense la presència de cap interfície gràfica) per realitzar diverses accions persistents, com ara la reproducció de música o la supervisió de l'activitat de la xarxa.

SDK Un kit de desenvolupament de programari o SDK (sigles en anglès de *Software Development Kit*) és un conjunt d'eines de desenvolupament que permet a un programador crear aplicacions per a un sistema concret.

superfície Un objecte de tipus `Surface` que representa un bloc de memòria que es composa a la pantalla. Una `Surface` alberga un objecte `Canvas` per a dibuixar, i ofereix diversos mètodes d'ajuda per dibuixar capes i canviar la mida de la superfície. No ha d'usar aquesta classe directament, sinó `SurfaceView`.

SurfaceView Un objecte `View` que envolta una superfície de dibuix, i exposa els mètodes per especificar la mida i el format de forma dinàmica. Una `SurfaceView` proporciona una manera de dibuixar independentment del *thread* de la interfície gràfica per a les operacions d'ús intensiu de recursos (com els jocs o les vistes prèvies de la càmera), però amb el cost d'utilitza memòria addicional. `SurfaceView` dóna suport tant gràfics de Canvas com de OpenGL ÉS. La classe base és `SurfaceView`.

vista (*View*) L'objecte més important a l'hora de muntar una interfície gràfica. D'ell estendran la majoria de components que utilitzem. Bàsicament consisteix en una àrea rectangular de la pantalla que admetrà determinades funcionalitats i esdeveniments d'interacció. Una `View` és una classe base per a la majoria dels components de disseny d'una activitat o de la pantalla de diàleg (quadres de text, finestres, etc.). Rep trucades del seu objecte pare (`ViewGroup`) per dibuixar-se a si mateix, i li informa sobre a quina posicó de la pantalla vol ser-hi i quin tamany vol tenir

VRAM *Video Random Access Memory* (VRAM) és un tipus de memòria RAM que utilitza el controlador gràfic per poder manejar tota la informació visual que li envia la CPU del sistema. La principal característica d'aquesta classe de memòria és que és accessible de forma simultània per dos dispositius. D'aquesta manera, és possible que la CPU gravi informació en ella, mentre es llegeixen les dades que seran visualitzats en el monitor en cada moment.

UI Sigles en anglès per a *User Interface*: la interfície gràfica d'usuari.

ViewGroup Un objecte contenidor que agrupa un conjunt de `Views` fills. El `ViewGroup` és responsable de decidir on es col·loquen els View fills i quin tamany tindran, així com per fer crides a cada un per dibuixar-se a si mateix quan calgui. Alguns `ViewGroup` són invisibles i només per al disseny (*layout*), mentre que altres tenen una interfície d'usuari intrínseca (per exemple, un caixa amb una llista i un *scroll* de desplaçament).

Widget Element que pot afegir-se a l'escriptori del nostre dispositiu Android, oferint informació i determinades funcionalitats (segons el que el desenvolupador dugui a terme). Per exemple, una caixa de text, un menú emergent...

Window En una aplicació d'Android, un objecte deriva de la classe abstracta `Window`, la qual especifica els elements d'una finestra genèrica, com ara l'aparença (títol de text de la barra, la ubicació i el contingut dels menús, etc.). `Dialog` i `Activity` utilitzen una implementació d'aquesta classe per fer una finestra.

Fonts emprades per a la confecció del glossari:

http://www.elandroidelibre.com/2014/09/glosario-android-80-conceptos-que-deberias-conocer.html

https://developer.android.com/guide/appendix/glossary.html

http://en.wikipedia.org

7. Bibliografia

EDITORIAL JUST IN TIME S.L. (2005). *Ejercicios para potenciar la memoria de los enfermos de Alzheimer*.

EDTECH DIGEST (2012). *Trends | Infographic: How Video Games Are Changing Education* [en línia]. *https://edtechdigest.wordpress.com/2012/12/07/trends-infographic-how-video-games-are-changing-education* [data de consulta: 26/12/2014]

GRIFFITHS, M. (2002). *The educational benefits of videogames*. [en línia]. http://sheu.org.uk/sites/sheu.org.uk/files/imagepicker/1/eh203mg.pdf [data de consulta: 26/12/2014]

MALYKHINA, E. (2014). *Fact or Fiction?: Video Games Are the Future of Education*. [en línia]. http://sheu.org.uk/sites/sheu.org.uk/files/imagepicker/1/eh203mg.pdf [data de consulta: 26/12/2014]

HIXON, TODD (2013). *Why Android Is Winning The Tablet Wars* [en línia]. http://www.forbes.com/sites/toddhixon/2013/05/14/why-android-is-winning-the-tablet-wars/ [data de consulta: 26/12/2014]

YOUNG, M. F.; SLOTA, S.; CUTTER, A. B.; JALETTE, G; MULLIN, G. LAI, B.; SIMEONI, Z.; TRAN, M.; YUKHYMENKO, M. (2012). *Our Princess Is in Another Castle : A Review of Trends in Serious Gaming for Education*. [en línia]. http://rer.sagepub.com/content/82/1/61 Review of Educational Research [data de consulta: 27/12/2014]

WIKIPEDIA. *Texture atlas*. [en línia]. http://en.wikipedia.org/wiki/Texture_atlas [data de consulta: 27/12/2014]

ZECHNER, M.; GREEN, R. (2011). *Beginning android 4 games development*. New York: Apress Media LLC.