Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

VIDEOJOC 2D PER ANDROID

Héctor Govinda Segura Riaza

Director: Eloi Puertas Prats
Realitzat a: Departament de
Matemàtica Aplicada i
Anàlisi. UB

Barcelona, 23 de gener de 2015

ÍNDEX

ÍNDEX

AGRAÏMENTS

Als amics de debò que m'han ajudat, ja sigui moralment o amb treball artístic i d'idees. Sincerament, no tinc moltes paraules que donar-vos, però sí tinc abraçades que compartir amb vosaltres. Per sobre de tothom, al meu cercle més proper: Al Sergio, que sempre has estat disposat a ajudar-me amb música i creació i retoc d'imatges, ets un artista! A l'Edu, que sempre sap com treure'm potencial, fins i tot quan jo no en tinc ganes, gràcies per les teves idees esbojarrades però genials. A la Natalia, pel seu entusiasme en treballar junts i pels seus dibuixos, ets una altra artista. Al Dani, "el meu Dani", qui sempre em sorprèn amb entusiasme, originalitat i bogeria de la bona.

A l'Eloi, el meu tutor, gràcies per la teva guia, la teva claredat, i el teu humor que et mostra sempre disponible per ajudar.

Als meus pares, que sempre m'han recolzat i inculcat a treballar al màxim sense preocupar-me pel fracàs; aquesta és realment la clau de l'èxit.

A tots els professors que m'han ajudat dia a dia i vocacionalment en el meu camí en aquesta carrera, on he après molt més que informàtica. A tots els companys que algun cop m'hagin ofert o m'hagin provocat un somriure.

Al Jaume, DEP.

A tots, moltes gràcies!

SUMMARY (English)

In the era of smart mobile devices, the videogames for those devices promote the two dimensions against 3D one more time.

At development level, this is due to the lower cost of production of 2D sprites instead of 3D models, the lower consumption of resources that the computer needs to run the application (memory, processing) which forces the use of a more powerful 3D graphics engine, more complex and therefore expensive.

At user level, the 2D still prevails today due to the strong aesthetics and simplicity that can be achieved with this type of graphics which can get a player to not realize that he is playing with graphics of two dimensions. Furthermore, the fact that a game on a mobile consumes fewer resources means more memory and, especially, more battery, a very valuable resource in mobile devices.

It is for these reasons that this 'TFG' is a 2D videogame, set in a post-apocalyptic world where a zombie virus has spread on humanity and the protagonist must escape alive in search of a cure or kill all the zombies that bump into his path. It is an ability game where the hero should go over the level, shooting the zombies that appear around the map using two fingers (multi-touch system), one for the movement and another for shooting to a direction.

The game is designed to be played on smartphones with *Android* (*Google*) operative system from *Android 4.1 JellyBean* version to the latest version on the market nowadays, *Android 5.0.2 Lollipop*.

RESUM (català)

En plena era de dispositius mòbils intel·ligents, els videojocs que els acompanyen tornen a enaltir les dues dimensions en contra del 3D.

A nivell de desenvolupament això és degut al menor cost de producció de *sprites* 2D en comptes de models 3D, al menor consum de recursos que té l'ordinador al executar la aplicació (memòria, processament) i a que el 3D et força a utilitzar un motor gràfic més potent, complex i, per tant, costós.

A nivell d'usuari, el 2D encara triomfa avui dia degut a la forta estètica i simplicitat que es pot aconseguir amb aquest tipus de gràfics on, fins i tot, es pot aconseguir que un jugador no se'n adoni que està jugant amb gràfics de dues dimensions. A més a més, el fet que un videojoc consumeixi menys recursos al seu mòbil es tradueix en més memòria i, sobretot, més bateria, un be molt preuat en dispositius mòbils.

És per aquestes raons que aquest TFG és un videojoc en 2D, ambientat en un món postapocalíptic on un virus zombi s'ha estès per la humanitat i el protagonista ha de fugir amb vida en busca d'una cura o bé matar a tots els zombis que es creuin al seu camí. És un joc d'habilitat on s'haurà de recórrer el nivell disparant a zombis que van apareixent pel mapa utilitzant dos dits (sistema *multi-touch*), un per moure's i l'altre per disparar cap a una direcció.

El joc està dissenyat per jugar-se en smartphones amb sistema operatiu *Android* (*Google*), des de la versió *Android 4.1 JellyBean* fins l'última versió al mercat avui dia, *Android 5.0.2 Lollipop*.

1. MOTIVACIÓ I ANTECEDENTS

A nivell personal, durant la carrera en el grau d'enginyeria informàtica es fan assignatures de tot tipus però el que sempre he tingut clar és que a mi m'agrada el desenvolupament de software. Partint d'aquesta base, hi ha molts camps de desenvolupament que em motiven i en els que, per tant, volia enfocar-me quan vaig decidir quin TFG fer. Dos d'aquests camps eren els videojocs i els dispositius mòbils, així que vaig fusionar els dos conceptes amb ganes d'aprendre més sobre ells. Aquesta és la principal causa d'aquest treball.

A nivell general, els videojocs són un mercat que mou a milions d'usuaris. En aquest treball s'ha volgut aconseguir "el videojoc perfecte", és a dir, aquell videojoc que tingui tots els elements que al gran públic més agraden. Les prioritats a l'hora de pensar el projecte han sigut les de fer un joc innovador, divertit i dinàmic de jugar. S'ha volgut fer un videojoc que no existeixi cap igual el qual tingui totes les bondats d'altres videojocs en un sol i, a més, que tingui detalls completament nous i mai vistos en altres.

Quan un videojoc no convenç al públic sol per les següents raons:

- **Falta d'innovació.** Dóna la sensació a l'usuari que un videojoc ja l'ha jugat abans perquè hi ha d'altres que són iguals o molt semblants.
- **Falta d'objectius clars.** Dóna la sensació a l'usuari que no sap perquè fa les coses, els nivells són massa llargs, hi ha massa repetició d'accions, hi ha estancament en les accions i llocs, o l'objectiu plantejat és tan llunyà que l'usuari s'avorreix i sol donar-se el conseqüent abandonament.
- **Falta d'emoció.** Si és molt fàcil de passar, si els enemics són fàcils de vèncer, si l'usuari no arriba a entendre com funciona el joc (hi ha jocs complexes que ni tan sols tenen tutorials), o si no arriba a sentir empatia amb el que passa a la pantalla.

1. MOTIVACIÓ I ANTECEDENTS

Ben cert és que hi ha jocs de tot tipus, i no tots han de ser totalment actius ni totalment reflexius però tots han de motivar-te a guanyar.

- **Falta d'estètica.** Si un joc és lleig, té gràfics dolents i irreals, la banda sonora i els sons són molestos o el que es veu no concorda amb el que passa. No és una condició necessària per tenir un bon videojoc però sí que és un afegit tremendament important per al gran públic.
- **Falta de continguts.** Si un videojoc es ven incomplet, amb continguts extres per descarregar de pagament, amb contingut premium (de pagament) que desequilibra el joc entre els jugadors que paguen i els que no paguen. Tot i que tenir contingut extra hauria de ser un avantatge, la majoria de cops es tergiversa el significat per vendre't un joc incomplet perquè després paguis més per poder jugar a tot el joc o per poder jugar al mateix nivell de la resta de jugadors, en el cas dels multijugadors.

No caure en aquests errors no és fàcil però tampoc és impossible, i la història de l'èxit dels videojocs n'és el testimoni:

- **Innovació.** Ja sigui pels medis físics amb els que s'interactuen com els botons, joysticks, comandaments sense fils, càmeres que capten els moviments, pantalles tàctils, ulleres de realitat virtual, etc.

  O bé al software del joc com, per exemple, el que van significar el *Pong*, el *Pac-Man*, o els primers que van fer un joc en 3D, van afegir la possibilitat de *modding* (fer els teus propis nivells (*Doom*)), van crear la vista en primera persona, els primers videojocs online, els primers jocs famosos de gènere com el rol (*Final Fantasy*, *Dragon Quest*), estratègia (*Cytron Masters*), lluita (*Warrior*), els primers jocs amb realitat augmentada, i un llarguíssim etcètera.

1. MOTIVACIÓ I ANTECEDENTS

- **Objectius clars.** Els objectius han d'existir a curt i mitjà termini. Per norma general no s'ha de mostrar el final del videojoc per tal que el jugador tracti d'arribar allà, obviant els jocs casuals on no hi hagi cap fil conductor entre nivells. Tot i això se'ls pot temptar donant a conèixer a l'arxienemic o donant la sensació que poden guanyar el joc quan encara vagin per la meitat.

  L'ésser humà és més capaç de dur a terme objectius més simples en el present o futur proper que no pas objectius complexes pel futur llunyà, que no vol dir que no sàpiga en quina direcció camina en el seu camí vital i que no tingui objectius llunyans. Un problema gran s'ha de partir en problemes petits i més simples. De la mateixa manera, un joc ha de plantejar objectius simples (per missions, nivells, mapes, capítols, amb petites recompenses, etc.) per tal d'arribar a objectius més grans. Tot sense caure en la repetició d'accions excessiva, que deriva en monotonia i que sovint és una estratagema utilitzada per tal de fer veure que el joc és més del que en realitat és.
- **Emocionant.** L'usuari ha d'arribar a patir les desgràcies del protagonista i a gaudir les seves victòries. Per això la dificultat ha de ser l'adequada (ni molt fàcil ni molt complicat), la usabilitat de les accions ha de ser el més natural possible i el feedback visual i auditiu ha de ser la correcta segons la presa de decisions de l'usuari. El flux del joc ha d'anar variant entre moments dificultosos amb altres més trivials o repetitius, amb petites pauses que donin la possibilitat de descansar el cervell d'estímuls i tenir moments per pensar estratègies, revisar inventari, millorar habilitats, estudiar combinacions d'accions, comprar o vendre objectes, etc.
- **Estètica.** De vegades es té el concepte erroni que cal que hi hagi una superproducció de molts diners per fer un joc amb bons gràfics quan no sempre és així, alguns cops la simplicitat ben dissenyada guanya a la complexitat o l'hiperrealisme. Per això fan

1. MOTIVACIÓ I ANTECEDENTS

falta bons artistes, dissenyadors gràfics i informàtics que siguin capaços de fer que un videojoc deixin de ser línies de codi i arxius per ser, en major o menor escala, una obra d'art. Cada acció que es dui a terme dins el joc ha de tenir la seva tornada visual i/o sonora (i d'altres si escau com, per exemple, vibratòria) de la manera més versemblant possible dins la corrent artística que s'hagi escollit pel videojoc (estil còmic americà, medieval màgic europeu, cyberpunk, rol japonès, western futurista, o el que sigui).

- **Continguts.** Un videojoc ha de vendre's complet, a no ser que es vulgui vendre una expansió important, que seria com una falsa segona part de l'anterior. Un contingut de pagament mai no ha de representar un desequilibri excessiu amb el contingut gratuït o de fàbrica. Els *DLC* i les petites actualitzacions són un bon instrument per correccions de *bugs*, millores de rendiment, re-equilibri del joc o per augmentar el joc. El videojoc ha de tenir una durada mínima o bé proporcionar la possibilitat de poder-lo jugar repetidament amb conseqüències diferents cada cop que es jugui. Aquesta segona possibilitat s'aplica més als multijugadors.

Havent definit què hauria i què no hauria de ser un bon videojoc, a continuació s'explicarà el disseny del videojoc d'aquest TFG, que s'ha tractat d'ajustar a aquestes definicions.

També és necessari esmentar l'ajuda rebuda en forma de brainstorm que van permetre millores del disseny del joc (*game design*) i l'ajuda artística, tant per desenvolupar alguns *sprites* com per crear tota la banda sonora des de zero. Les eines emprades per aquestes funcions són l'*Adobe Photoshop* pels sprites, l'*Ableton Live* per la producció musical i l'*Audacity* pel retoc de sons.

2. ANÀLISI I DISSENY DEL JOC

El disseny del videojoc creat és d'un joc gairebé complet, d'una fase prèvia a la versió 1.0. És per aquesta raó que des d'un principi es podia considerar un projecte de màxims, on no se sabia exactament fins on arribaríem però sí que no s'acabaria per raons de falta de temps i de personal treballant-hi. Tot i això s'havia de tenir clar com seria el videojoc en la seva versió més propera a la final o el que era segur és que amb una idea imprecisa i etèria no s'arribaria a fer un bon treball.

2.1. EINES DE TREBALL

L'IDE escollit per desenvolupar el codi ha sigut *Android Studio*.

Els dispositius on córrer l'aplicació per realitzar el *debug* han sigut diversos dispositius virtuals gràcies l'emulador d'Android Studio i diversos mòbils reals, el principal dels quals és un *Nexus 4* (*LG*).

La versió mínima d'Android per córrer l'aplicació és la 4.1 JellyBean i funciona fins l'última versió al mercat avui dia, *Android 5.0.2 Lollipop*.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.7%
4.1.x	Jelly Bean	16	19.2%
4.2.x		17	20.3%
4.3		18	6.5%
4.4	KitKat	19	39.1%

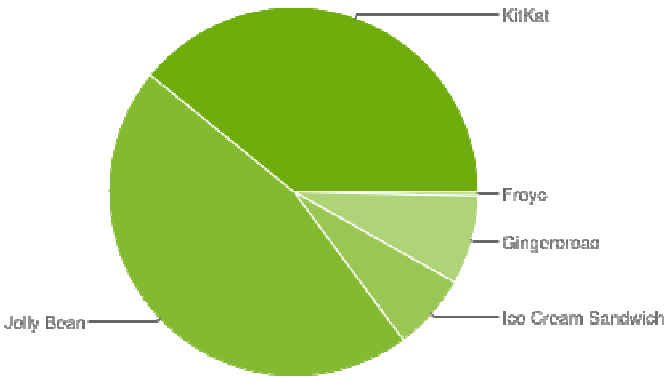

Figura 1. Distribució oficial de versions d'Android al 5 de gener de 2015. Qualsevol versió amb distribució menor al 0,1% no es mostra.

2.1. EINES DE TREBALL

A la figura 1 podem observar que la nostra aplicació pot executar-se aproximadament en el 85,1 % dels dispositius Android del mercat actual.

Espanya	% Novembre 2014
Android	85,9
iOS	9,7
Windows Phone	4,1
Altres	0,3

UE5	% Novembre 2014
Android	66,8
iOS	23,8
Windows Phone	8,3
Altres	1,2

Figura 2. Taula de quota de mercat de plataformes durant 3 mesos fins novembre del 2014.

UE5 (figura 2) correspon als cinc grans mercats de la unió europea: Gran Bretanya, Alemanya, França, Itàlia i Espanya. Observem que hi ha una majoria de sistemes *Android* a Espanya amb un 85,9 %, és a dir, segons les estadístiques, la nostra aplicació podria córrer al 73,1 % dels dispositius mòbils d'Espanya i al 56,9 % dels de la *UE5*.

2.2. DISSENY DEL JOC

A continuació definirem el disseny del joc en la seva versió pre-final.

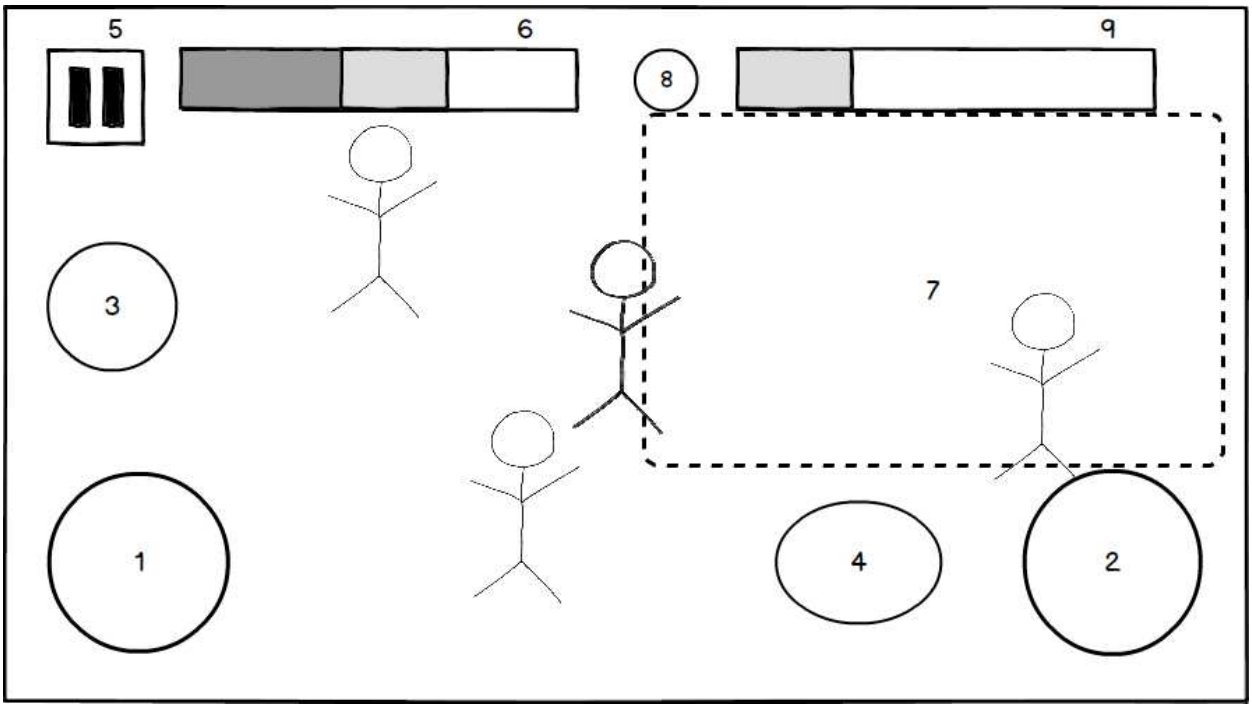

Figura 3. Disseny a paper del videojoc jugant-se.

Llegenda del disseny del joc (figura 3):

1. Joystick virtual pel moviment de l'heroi/protagonista.
2. Joystick virtual per la direcció dels trets.
3. Botó de selecció d'armes i objectes.
4. Botó per recarregar l'arma actual.
5. Botó de pausa.
6. Barra de vida.
7. Àrea de lliscaments amb el dit (*fling*) per fer atacs cos a cos en la direcció en què es faci el lliscament.
8. Informació sobre els diners virtuals acumulats.
9. Barra d'experiència acumulada per cada nivell de l'heroi.

2.2. DISSENY DEL JOC

A la figura 3, l'heroi és l'actor més remarcat i la resta són zombis.

El joystick per disparar (2 de la llegenda) es pot deixar premut i es va disparant cap on es vagi apuntant amb la cadència de trets i dany de les bales determinada per l'arma.

El botó de selecció d'armes i objectes (3) obriria un petit menú on seleccionar l'objecte desitjat en cada moment, que es classifiquen en: armes cos a cos (palanques, bats de criquet, matxets, etc.), armes de foc a distància (pistoles, metralletes, escopetes, fusells de franctirador), explosius (mines, granades, etc.) i objectes de cura de salut (farmacioles, menjar, analgèsics, adrenalina, etc.).

El botó de pausa (5) retorna al menú principal de l'aplicació.

L'heroi podria tenir una barra de vida normal i una barra extra de vida temporal (6). La temporal s'aniria reduint amb el temps o amb atacs rebuts i augmentaria, per exemple, si es prengués algun objecte com analgèsics, adrenalina o similars.

Dins l'àrea d'atacs cos a cos (7) es fan lliscaments amb el dit i l'heroi fa un atac d'empenta en la direcció on vagi el dit, amb poc dany cap al zombi però que li provoca un moviment de retrocés que alliberaria el camí de fugida de l'heroi si es veu atrapat de zombis o per tal d'evitar el seu atac.

Els diners virtuals (8) servirien per comprar objectes especials com roba (visualment), petites millores d'armes, armes especials (només visualment), música, etc. S'aniria guanyant molt de mica en mica dins el joc o bé amb diners reals.

L'experiència (9) s'aconsegueix matant zombis, passant-se nivells o aconseguint reptes opcionals que es plantejarien a l'usuari. Per cada nivell que pugés l'heroi podria aconseguir millors armes i objectes.

L'heroi tindria un "estat fúria" temporal, que permetria atacs especials que només podria utilitzar un cop per cada campanya i que s'activaria sacsejant el mòbil de dalt a baix. Aquest estat fúria consistiria en un gran augment de l'aguant, al rebre cops, i de dany d'atacs cos a

2.2. DISSENY DEL JOC

cos fets a l'àrea de lliscament (7 de la llegenda), gràcies al qual pràcticament mataria zombis d'un atac.

Hi haurien diversos tipus de zombis, alguns amb major vida, major velocitat de moviment o d'atac que d'altres. Tots tenen un cert camp de visió, és a dir, que si ets més ràpid i hàbil que ell, podries arribar a fugir-ne. Hi haurien zombis especials que podrien fer atacs de diferent forma com vomitar-te àcid, atacar-te amb armes, tenir molta més vida, llençar-te objectes, ser molt més ràpid, etc. Aquests, al ser especials, apareixerien molt menys sovint i en moments determinats.

El joc tindria campanyes amb diversos nivells cadascuna, de dificultat creixent. Per exemple, una campanya podria ser: Nivell 1 al metro Plaça Espanya, nivell 2 a la avinguda Reina Maria Cristina, nivell 3 a les pujades de la font màgica i nivell 4 al palau del MNAC.

Els requisits per passar cada nivell poden variar, per exemple, es pot demanar simplement arribar a cert punt sa i estalvi, arribar-hi havent matat un mínim de zombis, recollir certs objectes escampats pel mapa, etc.

El joc tindria una classificació (ranking) online de diverses estadístiques per tal de fomentar la competitivitat entre els jugadors. A més, tindria un sistema de trofeus proporcionats per *Google Play Games* pels punts d'experiència i per missions concretes diàries i d'altres perennes.

3. IMPLEMENTACIÓ

El videojoc té com a nom *Zombis*. El codi (nom de les entitats, mètodes i classes) i els textos dins el joc estan en anglès i els comentaris en català.

Figura 4. Icona i logotip del videojoc.

El llenguatge de programació utilitzat pel desenvolupament ha sigut exclusivament el Java, que és el llenguatge base pel desenvolupament d'aplicacions a Android. No hem utilitzat cap tipus de llibreria ni ajuda externa, tot el videojoc ha sigut programat amb l'*Android SDK* en la seva base nadiua, atorgant-nos així més control sobre el nostre codi al mateix temps que més treball i consciència del que desenvolupàvem.

El mètode de desenvolupament ha sigut el de *scrum*, és a dir, d'una reunió setmanal o bisetmanal per comprovar els progressos i de proposta de nous objectius creant, d'aquesta forma, el videojoc de forma incremental.

3.1. ARQUITECTURA DEL JOC

Diagrama amb les classes de l'aplicació *Zombis* i els seus mètodes més distintius. Cal dir que la classe `Audio` només està ben representada en la que està just a baix de `Constants`. Tot i que les altres representacions (amb fons gris) són la mateixa classe que aquesta, només està copiada d'aquesta manera per qüestió de comoditat visual amb les relacions (fletxes).

3.1. ARQUITECTURA DEL JOC

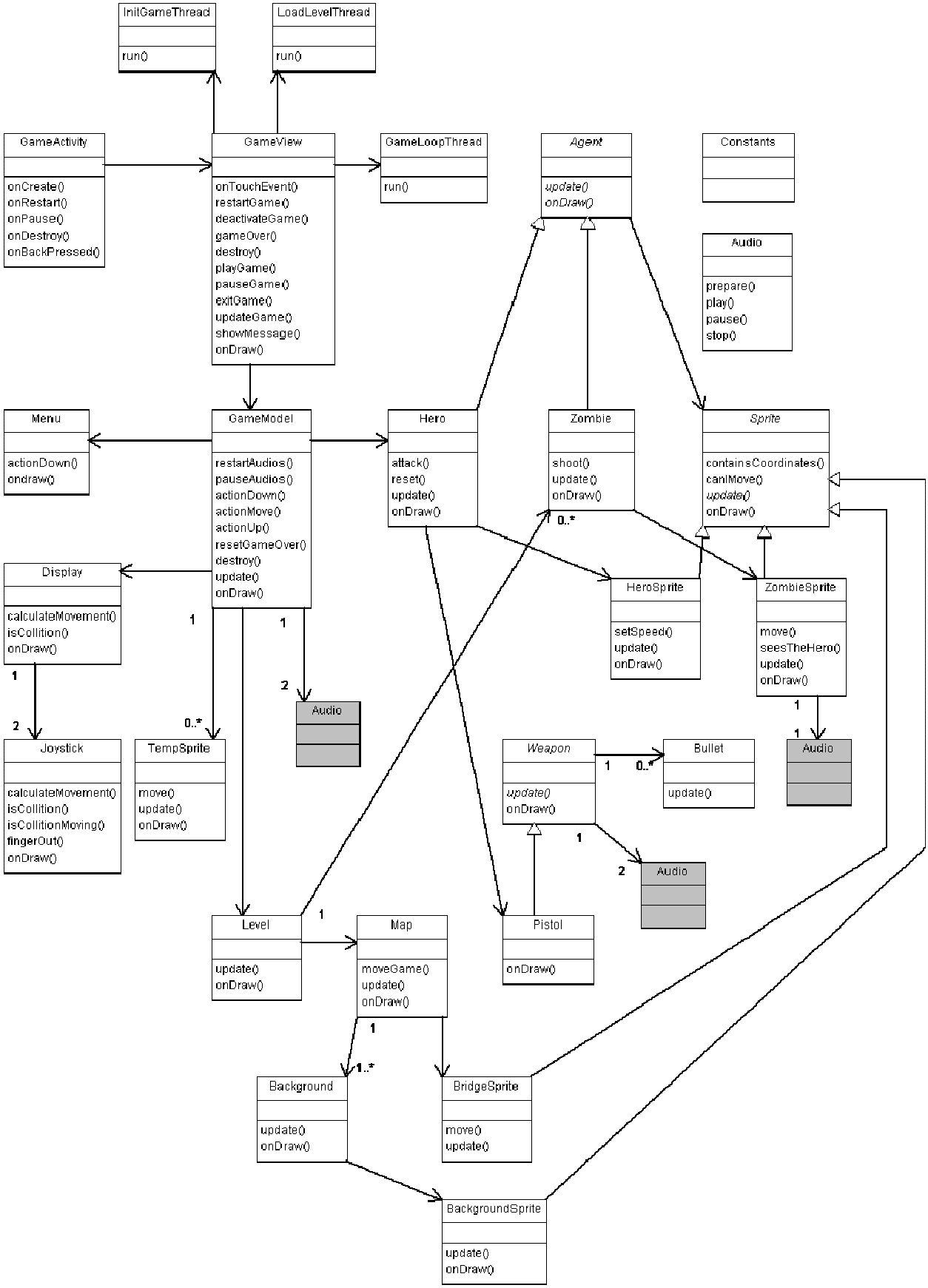

Figura 5. Diagrama de classes general de l'aplicació.

3.2. CARACTERÍSTIQUES BÀSIQUES D'ANDROID

A continuació explicarem, sense entrar en detalls excessius, components bàsics d'Android necessaris per entendre el nostre videojoc.

3.2.1. ACTIVITY

Una *Activity* és un component que se li proporciona una pantalla on poder dibuixar la seva *GUI* (sigui en una vista en *XML* o lliure com la nostra) amb la que un usuari pot interactuar amb el fi de fer alguna cosa.

Normalment, una aplicació té múltiples activitats que es limiten entre elles i una d'elles és marcada com la principal, és a dir, la *Activity* pare de tota la resta (en un sistema de *stack* o pila, també conegut com *LIFO*), des d'on es crea i destrueix l'aplicació sencera. Una activitat pot crear una altra per realitzar una altra tasca com prendre una fotografia, consultar un mapa, navegar per internet, etc.

En la nostra aplicació només tenim una activitat, la principal, en la classe anomenada `GameActivity`.

A continuació passarem a explicar com funcionen les parts que més ens importen sobre el cicle de vida de les activitats.

3.2.1. ACTIVITY

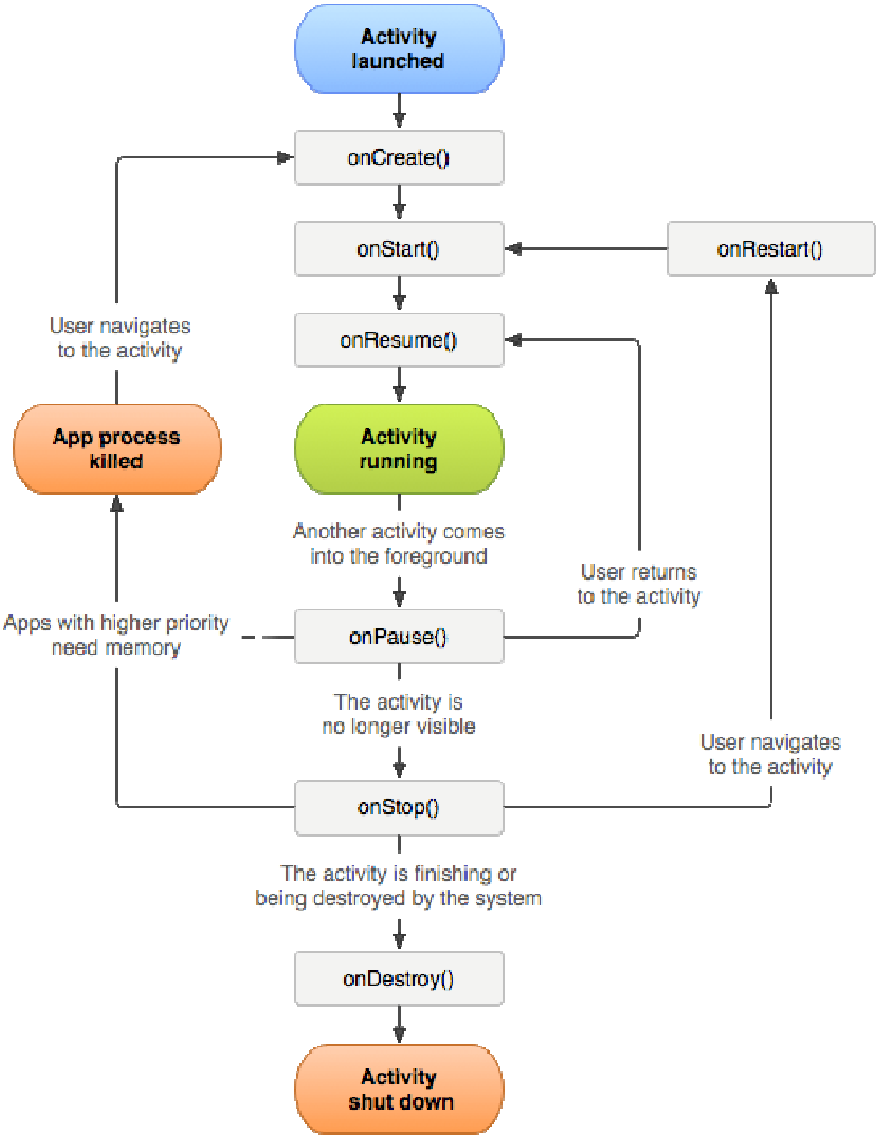

Figura 6. Cicle de vida d'una Activity.

Els esdeveniments (d'ara endavant *events*) del cicle de vida d'una *Activity* (figura 6) són mètodes que es poden sobreescriure. Els que ens interessen a la nostra aplicació són:

- `onCreate()`: Es crida quan es crea l'activitat, des d'aquí generem la *View* (en el nostre cas estenem de *SurfaceView* a la nostra classe `GameView`) i la vinculem a l'activitat amb `setContentView(view)`.

- `onPause()`: Es crida quan una activitat prèvia comença a cridar-se o bé simplement de camí a `onStop()`, és a dir, inclou quan l'aplicació deixa d'estar al front per passar a segon pla.
- `onResume()`: Es crida just quan l'activitat començarà a interactuar amb l'usuari.
- `onDestroy()`: Es crida quan l'aplicació comença a destruir-se, ja sigui per cridar a `finish()` o perquè el SO acabi l'activitat per requeriments de memòria.

3.2.2. SURFACEVIEW

En el *SurfaceView* tenim accés al *holder* de la pantalla (*SurfaceHolder*), que és la interfície que permet controlar la mida, els píxels, i els canvis que hi hagin a la pantalla del mòbil. Conté els events més simples de tocar la pantalla. Hem afegit un *callback* a aquest recipient per poder saber quan s'ha acabat de crear la superfície amb `surfaceCreated()` i poder començar a utilitzar la pantalla i, de fet, poder començar a carregar les dades i jugar.
A més, al *Surfaceholder* tenim accés al *Canvas*, que és la classe amb la qual podem dibuixar per pantalla. Abans de dibuixar s'ha de demanar el seu cadenat (*lock*) per a poder posar-nos a editar els píxels. Un cop acabem d'editar s'ha d'alliberar el cadenat. És en aquest *Canvas* on anem dibuixant els nostres *sprites 2D*, que no deixen de ser imatges en formats *JPEG* i *PNG* (que permet el canal alfa de transparència).

3.3. FUNCIONAMENT GENERAL DEL VIDEOJOC

3.3.1. GAME LOOP

Gairebé tots els videojocs funcionen en un bucle "infinit", sovint anomenat *game loop*, que s'acaba que ho fa el mateix joc. Dins aquest bucle només es fa una crida per actualitzar el joc, `update()`. Aquesta crida el que fa és actualitzar tot el model del joc segons els inputs de l'usuari, que en el nostre cas són l'efecte de tocar la pantalla i de prémer el botó d'enrere (`onBackPressed()` dins l'*Activity* principal), a més d'actualitzar-lo segons la IA dels agents i les animacions. Degut que aquests inputs són *events*, hem d'haver guardat prèviament (en el moment de l'event) la seva informació per poder utilitzar-la al moment de l'actualització. A continuació s'han de mostrar tots els efectes de l'actualització dibuixant-los a la pantalla amb `onDraw(canvas)`. Aquestes crides faran moltes més per totes les parts del model estenent-se en forma d'arbre (o graf dirigit, ja que de vegades es torna a branques de major alçada).

Degut també que l'actualització no pot ser bloquejant perquè hem de recollir els inputs de l'usuari de forma correcta i perquè podrien haver casos de saturació de càlcul, aquest bucle ha de ser en un fil o *thread* apart, en el nostre cas és a la classe `GameLoopThread`, que implementa *Runnable*. En aquesta classe fem la crida corresponent amb un temporitzador per a dormir el *thread* amb `sleep(milliseconds)`, que ens proporciona una actualització homogènia en el temps segons els 25 FPS que tenim marcats. És a dir, en mil·lisegons, en cada iteració trigarem, com a mínim, i en el millor dels casos també com a màxim a no ser que hi hagi molta falta de memòria:

$$1000ms/25\,fps = 40ms.$$

3.3.2. ESTATS DEL JOC

Tots els estats i el seus efectes es poden veure amb més detall a `GameLoopThread`:

- *Running*: Estat que indica si s'està executant el *game loop*.
- *Active*: Estat que indica si el joc està actiu, o sigui, l'aplicació està en primer pla o, pel contrari, l'aplicació està en segon pla.
- *Loading*: Estat que indica si el joc s'està carregant. Només s'activa quan carreguem el joc sencer, és a dir, quan volem carregar el model sense cap nivell. Es mostra una pantalla de càrrega.
- *Loading Level*: Estat que indica si s'està carregant un nivell del joc. Es mostra una pantalla de càrrega.

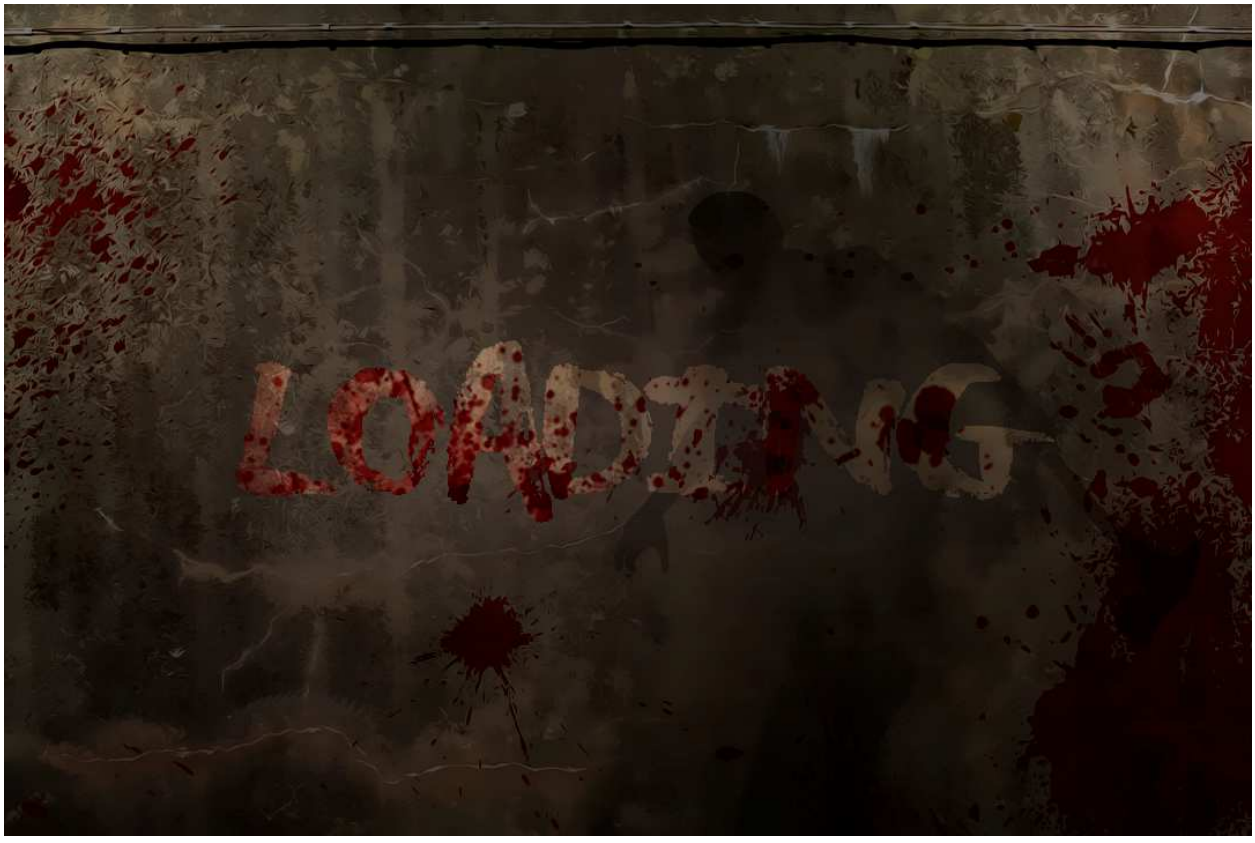

Figura 7. Pantalla de càrrega amb els estats *Loading* i *Loading Level*.

3.3.2. ESTATS DEL JOC

- *Paused*: Estat que indica si es té el joc pausat. Es mostra el menú principal del joc.

Figura 8. Pantalla de menú del joc amb l'estat *Paused*.

- *Game Over*: Estat que indica si l'heroi és mort i, en conseqüència, el joc ha acabat o bé si el joc s'ha acabat per haver "guanyat". Es mostra una pantalla de mort durant quatre segons que es pot saltar prement el botó enrere del dispositiu mòbil. Posteriorment es retorna al menú principal.

3.3.2. ESTATS DEL JOC

Figura 9. Pantalla de joc acabat de l'estat *Game Over*.

El flux entre els estats és el següent (pseudocodi):

```
Mentre Running
      Si Active, llavors
            Si Loading o Loading Level, llavors
                  Dibuixar pantalla de Loading;
            Sinó, Si Game Over, llavors
                  Mostrar pantalla de Game Over;
            Sinó, Si Paused, llavors
                  Mostrar menú;
            Sinó, llavors
                  Actualitzar el joc;
                  Mostrar el joc;
```

3.3.3. LÒGICA GENERAL DEL JOC

Quan es crea l'activitat principal (`GameActivity`), es crea i vincula la *view* a aquesta classe. Quan a la vista ja tenim disponible la superfície de la pantalla a l'*event* `surfaceCreated()` dins el *callback* del *holder* (`GameView`), assignem l'estat *running* de `GameLoopThread` a vertader; l'estat *loading* també era vertader, per defecte. Aquí comença oficialment el bucle del joc, i el primer que es fa és carregar les dades inicials del model del joc (excepte les del nivell del joc), la classe `GameModel`, mentre s'ensenya la pantalla de càrrega. Aquesta inicialització del model la fem en un nou *thread*, a la classe `InitGameThread`. En cas de fer-la sense un *thread* separat del joc tindríem una saturació de càlcul que faria que l'aplicació es saltés més 150 *frames*, o imatges i el feedback visual seria molt dolent per l'usuari, ja que el mòbil es quedaria com bloquejat sense mostrar cap pantalla de càrrega. Un cop acaba la càrrega, el *thread* d'inicialització acaba, *loading* passa a ser fals i es mostra el menú de la classe `Menu`, ja que *paused* també era vertader per defecte. En cas que es premés el botó *back*, o enrere, del mòbil mentre es carrega, l'aplicació es destrueix.

En el menú podem donar-li al botó *Exit* per destruir l'aplicació, o podem donar-li al botó *PLAY* per començar a jugar. En el segon cas, faríem una càrrega del nivell corresponent, que es representa en la classe `Level`, mitjançant un nou *thread* d'una manera molt semblant com vam fer anteriorment per carregar `GameModel`, aquest cop en la classe `LoadLevelThread`, activant-se l'estat *loading level* fins que s'acabi d'inicialitzar el nivell en concret.

A partir d'aquí comença el joc en sí. Si es prem el botó enrere mentre jugues, s'activa *paused* i, per tant, tornes al menú i el joc queda pausat fins que tornis a donar-li a *PLAY*, ja que el joc no s'està actualitzant mentre estigui pausat, els estats són exclusius entre ells. Si a l'heroi se li acaba la vida, s'activa l'estat *game over* i *paused* per tornar al menú quan acabi

el primer estat, que succeeix després de passar quatre segons o fins que es prem el botó enrere del mòbil. Si l'heroi acaba el joc, també s'activa l'estat *game over*, degut a que falla en la missió de trobar una cura per la infecció zombi.

Si l'aplicació queda en segon pla pel motiu que sigui, l'estat deixaria de ser *active* i s'activaria *paused* (i *active*) quan tornés a estar en primer pla.

3.3.4. ARBRE D'OBJECTES DEL MODEL DEL JOC

A continuació exposarem els diferents objectes del model del joc per entendre què son i com estan organitzats entre ells. D'aquí es pot extreure el camí aproximat d'actualització del joc, ja que gairebé tots aquests objectes tenen un, o més, mètodes `update()` i mètodes `onDraw(Canvas canvas)`. Per entendre-ho millor potser cal tenir present el diagrama de classes (figura 5).

Objectes importants de `GameModel`:

- `Menu`: El menú de pausa del joc.
- Llista de `TempSprite`: [0..n] objectes de *sprites* temporals, és a dir, *sprites* peribles en el temps, que poden ser taques de sang quan un zombi rep un tret de pistola, o un cadàver d'un zombi mort, quan se li acaba la vida.
- `Agent` heroi: Aquest agent s'inicialitza amb la classe `Hero`, fent ús del polimorfisme que ens proporciona Java al ser una classe que hereta (`extends`) d'`Agent`.
- `Level`: El nivell actual del joc.
- `Display`: El *display* o també anomenat interfície de joc d'usuari, que inclou botons, joysticks i *sprites* d'informació.

3.3.4. ARBRE D'OBJECTES DEL MODEL DEL JOC

Objectes importants de `Hero`:

- `Weapon`: que s'inicialitza amb la classe `Pistol`, que és la pistola inicial de l'heroi. Així, en cas d'aconseguir un altre arma faríem ús el polimorfisme.
- `Sprite`: Tot i que en realitat és la classe `Agent` qui té un objecte `Sprite`, que, en aquest cas, s'inicialitza com `HeroSprite`.

`Weapon` té una llista de [0..n] `Bullet`, que representen les bales disparades que encara segueixen veient-se per la pantalla.

Objectes importants de `Level`:

- `Map`: Mapa d'aquest nivell.
- Llista d'`Agents` zombis: Cada agent s'inicialitza amb la classe `Zombie`.

Objectes importants de `Map`:

- Matriu de `Background`: Diferents fons del mapa, que està tessel·lat en una graella quadrada, o sigui, té el mateix número de files que columnes.
- `BridgeSprite`: Pont per acabar el nivell actual i anar al següent si n'hi ha.

`Background` té un objecte `BackgroundSprite`, on es troba el *sprite* del fons en concret.

`Zombie` hereta d'`Agent`, que té un objecte `Sprite` que s'inicialitza com `ZombieSprite`.

`Display` té dos objectes `Joystick`, que són els dos controladors de l'heroi per l'usuari, el de moviment a través del mapa (situat a l'esquerra de la pantalla de joc) i el de disparar cap a la direcció escollida (situat a la dreta de la pantalla de joc). Al display també hi ha els objectes corresponents per a mostrar la resta d'elements, que corresponen als números 4, 5 i 6 del disseny (figura 3).

Només faltaria esmentar `Audio`, que representa un àudio en general, ja sigui el so d'un tret de bala o la cançó del menú.

Per finalitzar, `Constants`, que és una classe no es pot instanciar degut al seu constructor privat, on hi ha emmagatzemada la gran majoria de constants que es van utilitzant al llarg de l'aplicació.

3.4. FUNCIONAMENT ESPECÍFIC DELS ELEMENTS DEL JOC

3.4.1. SISTEMA D'IDENTIFICACIÓ DE DITS

A la classe `GameView`, que estén `SurfaceView`, sobreescrivim el mètode `onTouchEvent(MotionEvent event)`, que és el *listener* de l'*event* de tocar la pantalla amb el dit. Tindrem en compte els estats del joc perquè no hi hagin efectes no desitjats, com ara prémer botons del menú mentre estem jugant o viceversa. L'objecte `MotionEvent` proporcionat és qui ens donarà tota la informació desitjada per nosaltres:

- Quin *event* (acció) s'està processant. Obtenim la màscara de l'acció amb `event.getActionMasked()` i mitjançant un *switch* anem a les accions que ens interessi:
    - `MotionEvent.ACTION_DOWN`, quan el primer dit prem la pantalla.
    - `MotionEvent.ACTION_POINTER_DOWN`, quan a partir del segon dit cap endavant prem la pantalla.
    - `MotionEvent.ACTION_MOVE`, quan un dit ja premut, es mou de coordenades.
    - `MotionEvent.ACTION_UP`, quan l'últim dit s'aixeca de la pantalla.
    - `MotionEvent.ACTION_POINTER_UP`, quan un dit que no sigui l'últim s'aixeca de la pantalla.

3.4.1. SISTEMA D'IDENTIFICACIÓ DE DITS

- L'índex de l'acció en concret dins l'*array* intern de *pointers*, un per cada dit actuant en aquest moment en concret, amb `event.getActionIndex()`.
- L'identificador únic de cada dit premut a la pantalla que ens permet saber qui fa què, amb `event.getPointerId(index)`, on `index` és el de l'explicació al punt anterior.
- Buscar l'índex d'un dit dins l'array intern mitjançant el seu identificador amb `event.findPointerIndex(id)`.
- Les coordenades del dit a la pantalla amb `event.getX()`, `event.getY()` o `event.getX(index)` i `event.getY(index)`.

Degut a que necessitem un sistema per identificar quin dit prem quin joystick pel nostre sistema *multi-touch*, hem creat un *hash map* on s'emmagatzemen *<identificador : coordenades>* de cada dit.

Quan arriba un *event* de *down* o *pointer_down* s'afegeix un nou element al *hash*, i quan arriba un *event* de *up* o *pointer_up*, s'elimina l'element concret de la *hash*. Quan arriba un *event* de *move*, hem de cercar dins la *hash* el punt que hagi canviat les seves coordenades per saber qui s'ha mogut i poder actuar en conseqüència.

3.4.2. JOYSTICKS VIRTUALS

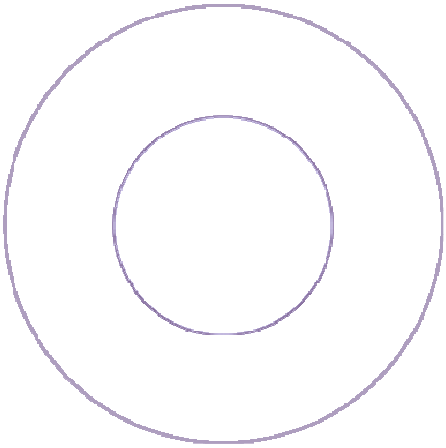

Figura 10. Joysticks del joc (pel moviment i per disparar).

Els dos joysticks funcionen amb la mateixa classe `Joystick`, tot i tenir comportaments diferents. Un joystick està composat, com es pot veure a la figura 10 de dos cercles semitransparents, un gran i un petit que mesura la meitat del gran. El cercle petit mai es surt del gran però pot moure's per tota la seva àrea.

El joystick té un estat binari, *pressed*, que indica si en un moment determinat està premut per algun dit o no ho està. En cas que ho estigui, el cercle petit es situa, o bé centrat sobre el dit si el cercle petit cap al cercle gran sense sortir-se, o bé el més a prop del dit si el cercle petit es surt del cercle gran si el petit es centrés sobre el dit. En cas que no estigui premut, el cercle petit es situa centrat just enmig del gran.

El funcionament dels joysticks es divideix en 2 parts principals:

1. Comprovar col·lisió al prémer el dit contra la pantalla.
2. Actualitzar el cercle petit si s'estava prement el joystick.

3.4.2. JOYSTICKS VIRTUALS

La primera part es duu a terme al mètode `isCollition(int x, int y, int id)`, on es passen les coordenades (x, y) específiques on s'ha premut de la pantalla i l'identificador del dit en concret per diferenciar entre els dos joysticks si ja estaven premuts.

Volem saber si les coordenades on s'ha premut estan dins del cercle gran. Per això, fem una translació del sistema de coordenades per tal de col·locar les coordenades (x, y) relativament al centre del cercle gran, és a dir, que el centre del cercle sigui l'origen de coordenades, simplement restant a (x, y) les coordenades del centre del cercle.

Seguidament, calculem la distància del punt (x', y') al centre del cercle mitjançant el teorema de Pitàgores pel triangle que es forma entre el punt i el centre:

$$d^2 = x'^2 + y'^2$$

Llavors, si la distància és menor o igual que el radi gran, vol dir que s'està dins el cercle, en cas contrari s'està fora.

Aquest mètode de comprovació de col·lisió també s'usa amb el botó de pausa (a `Display`), que és circular.

La segona part es duu a terme al mètode `isCollition(int x, int y, int id)` i `isCollitionMoving(int x, int y)` per quan és ve d'un *event* de moviment. Degut a que el cercle petit mesura la meitat del gran, si premem dins l'àrea petita (la del cercle petit en repòs), el cercle petit es podrà moure lliurement sense arribar a sortir-se mai del gran. Si, en canvi, premem o movem el dit fora del radi petit, o sigui, dins la corona circular entre el petit i el gran o bé directament movem el dit fora del gran, haurem de calcular la nova posició del cercle petit mitjançant coordenades polars:

3.4.2. JOYSTICKS VIRTUALS

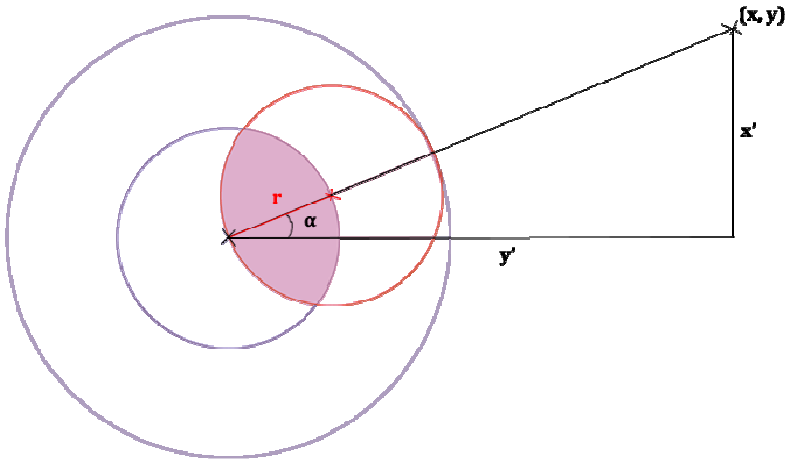

Figura 11. Càlcul de nova posició del cercle petit del joystick (en vermell).

1. Obtenim l'angle α mitjançant l'arc tangent:

$$\tan(\alpha) = \frac{y'}{x'}$$

$$\alpha = \arctan(\frac{y'}{x'})$$

2. Cerquem el punt del nou centre (x'_r, y'_r) trobant cada catet del triangle multiplicat per *r* (radi cercle petit):

$$x'_r = r * \cos(\alpha)$$

$$y'_r = r * \sin(\alpha)$$

Un cop tenim el punt, només cal dibuixar el cercle petit centrat allà cada cop.

3.4.3. ÀUDIO

Android té una llista extensa de canals de flux d'àudio com ara per l'alarma, les notificacions, les trucades, la veu, etc. En iniciar l'aplicació, hem reservat el canal de flux d'àudio de música, que és el canal per on treure l'àudio que s'utilitza en aquest cas per diverses aplicacions i videojocs, amb:

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

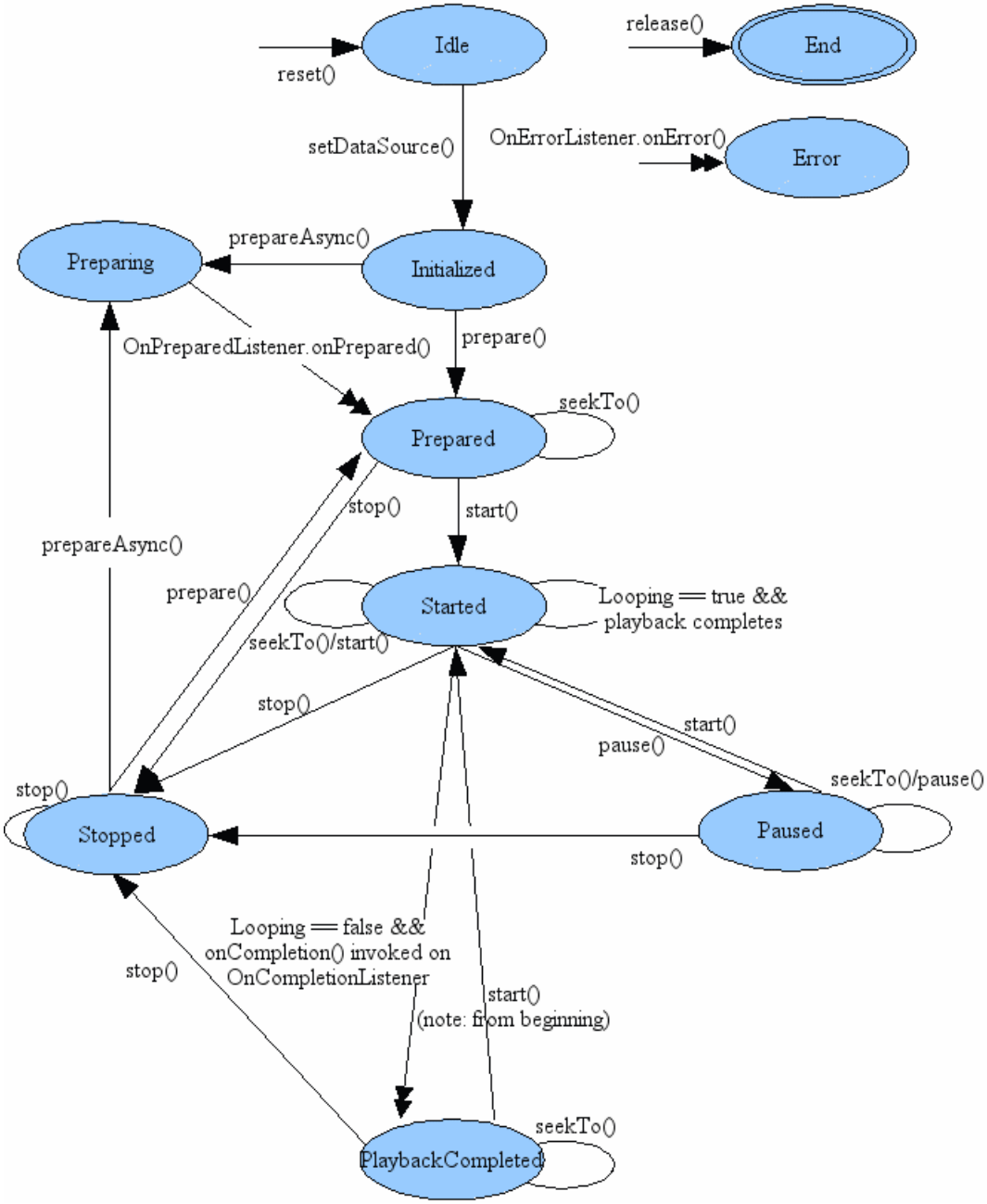

Figura 12. Diagrama d'estats de *MediaPlayer*.

3.4.4. ACTUALITZACIÓ DELS MOVIMENTS

Hem creat una classe `Audio` per poder instanciar-la on necessitéssim tenir un so. A aquesta classe tenim un objecte *MediaPlayer*, que és el reproductor de multimèdia utilitzat. La creació del reproductor amb `MediaPlayer.create()` t'estalvia tota preparació, és a dir, a partir d'aquí ja pots reproduir un so amb `start()`.

Com podem observar al seu diagrama d'estats (figura 12), si parem un àudio amb `stop()`, haurem de tornar a preparar-lo per a la seva reproducció amb `prepare()`.

Per destruir un so i alliberar la seva memòria s'ha de parar, restablir (tot i que aquesta part no es vegi clara en la imatge, s'ha de fer) i alliberar: `stop()`, `reset()`, `release()`.

Tots els àudios estan en format *OGG* (.ogg), que és un format contenidor de multimèdia de codi obert que permet un rendiment excel·lent en flux de bits (*bitstream*) a més d'una qualitat de so i una compressió més que notables.

3.4.4. ACTUALITZACIÓ DELS MOVIMENTS

L'heroi sempre es situa enmig de la pantalla, excepte en els casos extrems d'arribar a les vores del mapa, on comença a moure's cap a les cantonades.

Al `update()` de `GameModel` podem veure que primer s'actualitza el moviment del mapa (passant-li les velocitats extretes del joystick de moviment), després l'heroi i per últim la dels zombis i objectes del nivell. Això es deu a que quan l'heroi vol moure's primer ha de comprovar que el seu moviment no faci que la càmera, o la finestra que conforma la pantalla no es surti del mapa.

Si no es surt del mapa, es mouen tots els fons del mapa, els zombis i objectes del mapa, obviant sempre les velocitats individuals que puguin tenir els zombis. Cal tenir en compte que, només en aquest cas, si l'heroi es volgués moure cap a la dreta, en coordenades de

càmera en realitat l'heroi es mantindria quiet i la resta hauria de moure's cap a l'esquerra, no cap a la dreta.

Si, en canvi, amb la quantitat de moviment l'heroi es sortís del mapa (estant ell centrat enmig), s'hauria de començar a moure l'heroi cap a la vora i el mapa, els zombis i els objectes es quedarien tal qual.

Però en realitat hi ha un altre cas extrem quan, per exemple, l'heroi està situat a distància 2 de la vora de l'esquerra i el moviment indica que s'ha de moure una quantitat de 5 cap a l'esquerra. En aquest cas, el mapa, zombis i objectes s'haurien de moure una quantitat de 2 cap a la dreta i l'heroi una quantitat de 3 cap a l'esquerra.

Es per aquesta raó que primer hem de veure el moviment al mapa i extreure d'allà les possibles velocitats que haguem d'aplicar a la resta.

Apart, cal esmentar que cada `Background` té un estat *visible*, que marca si és visible en aquest moment per a la finestra de la pantalla. Si ho és, es dibuixa amb `onDraw()` a les seves coordenades corresponents, sinó, no es dibuixa. Amb això guanyen eficiència en el processament.

3.4.5. ALLIBERAMENT DE MEMÒRIA

En el moment que s'arriba a `onDestroy()` de l'activitat principal, just abans que es tanqui l'aplicació es fan crides als mètodes `destroy()`, creats a gairebé totes les classes. En aquests mètodes tenim l'alliberament de memòria de recursos que ja no s'utilitzaran, que són imatges i àudios.

L'alliberació d'àudios, abans esmentada, es fa amb `stop()`, `reset()`, `release()`. Les imatges es fan apuntar a *null* perquè, d'aquesta manera, el *garbage collector* de Java alliberi la memòria reservada per les imatges.

4. RESULTATS

L'aplicació *Zombis* ocupa 3.7 MB, i instal·lada ocupa 6 MB en memòria secundària (*SD*). En execució sense carregar el nivell, ocupa 59 MB de memòria (*RAM*) i havent carregat el nivell amb tots els zombis vius ocupa 77 MB. Sortint de l'aplicació però sense tancar el procés, ocupa residualment 17 MB. Són resultats relativament bons.

L'aplicació ha estat provada en múltiples smartphones amb èxit excepte en un mòbil concret, el *Samsung Galaxy S II*. Les raons són desconegudes ja que no s'ha tingut oportunitat d'obtenir aquest dispositiu per fer tests. En un *Samsung Galaxy S III*, en canvi, sí que funciona perfectament, o sigui que és algun problema de l'arquitectura de software d'aquest dispositiu en concret.

Visualment i sonorament tot és completament fluït i, de fet, es podria apujar els *frames per second* bastant obtenint la mateixa fluïdesa total, però 25 *FPS* ja és un bon número. En cas de voler menys consum de recursos es pot baixar els *FPS* i seguiríem obtenint una bona resposta.

Les parts de càrregues i menú ja s'han explicat com han quedat mentre es relatava el seu funcionament.

4. RESULTATS

Figura 13. Captura de pantalla del joc. L'heroi ha matat un zombi i està disparant a un altre.

Els dos joysticks reaccionen perfectament. El del moviment és gradual segons la distància del dit al centre de les circumferències, quant més llunyà més ràpid es mou, amb un cert límit. El d'apuntar dispara allà on apuntis, sense tenir en compte la distància dit-centre de joystick, però allunyar-lo facilita l'apuntar visualment per a l'usuari.

Les bales són infinites, com marca la imatge al costat del joystick dretà, i no cal recarregar l'arma, que només hi ha la pistola inicial.

L'animació de les bales és suficientment clara com per veure el recorregut que fan, desapareixen quan col·lisionen amb un zombi. Es sent un so de tret just al moment en què es dispara una bala. Les bales ataquen als zombis fent que sagnin i finalment morin deixant un cadàver al terra, ambdues imatges (sang i cadàver) es fonen lentament fins desaparèixer quan passa un cert temps.

4. RESULTATS

El botó de pausa funciona correctament, d'igual manera que prémer el botó enrere del smartphone, que porta al mateix resultat.

Al menú del joc hi ha una música i al joc hi ha una altra més ràpida.

Els fons del mapa es veuen correctament sense notar-se en cap moment el tessel·lat.

Tant l'heroi com els zombis tenen cos tangible que no deixa que els agents (heroi i zombis) es travessin entre ells, per tant si venen molts enemics cap a l'heroi poden formar-se ramats de zombis, i, de la mateixa manera, l'heroi no pot travessar els zombis.

Les animacions pels moviments dels agents al caminar corresponen a la direcció de moviment (tenen quatre direccions on mirar), excepte per l'heroi que, si està disparant, es prioritza mirar cap on dispara i no cap on camina. Les animacions al donar passes corresponen a la seva velocitat de moviment.

Els zombis es mouen cap a una direcció aleatòria un cert temps, es paren una estona un altre cert temps i tornen a moure's cap a una altra direcció aleatòria iterativament. Quan l'heroi és a un rang de visió determinat dels zombis, poden veure a l'heroi i van directes a atacar-lo.

Quan t'ataca un zombi para el seu moviment de caminar i, passat un petit temps, es veu una petita animació d'atac mentre es sent un so de mossegada. La barra de vida de l'heroi s'actualitza correctament just quan es duu la mossegada.

4. RESULTATS

Figura 14. Captura de pantalla de joc. L'heroi tracta de fugir en cerca de la cura de la infecció, però encara no pot perquè no ha complert la missió principal.

Al principi del nivell es mostra a l'usuari la missió principal del nivell en un missatge per pantalla. Per acabar el nivell s'ha de cercar i anar al pont havent complert la missió sa i estalvi. Si no has complert la missió es mostra un missatge com el de la figura 14. Si la completes, et mors degut que no hi ha cura per la infecció zombi. Només hi ha un nivell per jugar, tot i que el codi està molt ben preparat per poder crear més nivells.

4.1. INVESTIGACIÓ

Respecte l'àrea de lliscaments amb el dit (*fling*) per fer atacs cos a cos en la direcció en què es faci dit lliscament, després d'investigacions exhaustives, ens vam adonar que era impossible de realitzar de la manera en què el disseny ho requeria. La raó és la següent, per poder obtenir els *events* del moviment *fling* (no ens interessa cap més) es fa ús de la classe *GestureDetector.SimpleOnGestureListener*.

Es pot utilitzar de la següent manera:

```
private GestureDetector gestureDetector = new
     GestureDetector(gameActivity,
     new GestureDetector.SimpleOnGestureListener() {
     @Override
     public boolean onFling(MotionEvent event1, MotionEvent
          event2, float velocityX, float velocityY) {
          //instructions...
          return true;
     }
});
```

Des del mètode `onTouchEvent(MotionEvent event)` li passem l'event al detector de gestos amb `gestureDetector.onTouchEvent(event)`.

El problema és que li passem l'*event*, que conté la informació de tots els dits actius, però que per sí sol només sap retornar la informació del primer dit (down, move i up), a no ser que li demanis concretament passant-li el seu índex. És a dir, que onFling només detectarà el llançament del primer dit, que ens resulta inútil tenint en compte que normalment s'estarà prement el dit en algun joystick o botó.

4.1. INVESTIGACIÓ

Qualsevol pot veure millor que *onFling()* no és viable en *multi-touch* quan, per exemple, en una llista qualsevol com la llista d'alarmes de l'aplicació bàsica d'alarmes d'*Android*. Si fem un gest de llançament cap avall o amunt, la llista es mou a la velocitat adequada però si deixem premut un dit i amb l'altre fem un llançament, la llista no es mou adequadament, ja que no es captura el gest de llançament.

Es podria dir que ha sigut un intent d'innovació fallit.

5. CONCLUSIONS

- *Android* és, probablement, el sistema operatiu per dispositius mòbils que més triomfa al món però, tot i així, li falta més documentació oficial, segurament degut al seu constant creixement.

- Desenvolupar videojocs és una tasca que requereix tenir especialitats en múltiples camps, artística en 2D, artística en modelat 3D, en programació general, en intel·ligència artificial, en so, en el sistema operatiu, etc. Per aquesta raó, queda patent que s'ha de fer amb un equip multidisciplinari, el treball en equip és la clau per obtenir bons productes en general.

- Molts cops, per programar tasques simples l'aplicació es complicava molt i el temps de desenvolupament era llarg degut a que sempre s'ha tractat de modularitzar-ho tot en la cerca de la simplicitat i naturalitat del codi pensant en el desenvolupament del futur. Tot i així, fer-ho d'aquesta manera val la pena, ja que algun cop ha passat que s'ha programat monolíticament i després, per fer altres tasques relacionades, s'ha hagut de realitzar més treball del degut per reordenar-ho tot.

- Tot i la dificultat en instal·lació d'*Android Studio* i els seus components, i l'enorme lentitud del seu emulador de dispositius, és una molt bona eina de treball que permet moltes accions útils pel desenvolupament i *debugging*.

5. CONCLUSIONS

- S'han adquirit molts coneixements sobre *Android*, programació en general i videojocs gràcies a *MOOC* (cursos online massius), documentacions, fòrums, tutories i, sobretot, al treball diari. És curiós com en un sol treball es noten els coneixements dispars de moltes assignatures realitzades durant la carrera.

- A nivell personal, estic molt content amb el treball realitzat i els resultats obtinguts. És el treball més gran que he realitzat i el que més m'agrada és que l'he fet jo des de l'inici de la idea de fer un videojoc o fer una aplicació per *Android*, fins l'actual aplicació *Zombis*. És un treball al qual vull donar-li continuïtat, ja que crec que si he fet aquesta carrera és per acabar fent coses que m'entusiasmin, i no per convertir-me en multimilionari... tot i que si van de la mà molt millor.

6. TREBALL FUTUR

Revisant el disseny del joc (tema 2.2.), es pot comprovar que hi ha moltes coses per fer perquè quedi el videojoc complet plantejat allà. Tot i això, cal tenir en compte que el disseny és molt ambiciós, ja que al mercat no sol haver videojocs tan complets per ús exclusiu de mòbils.

Apart de les parts del disseny que falten en la versió final del treball, hi ha temes més immediats o de mig termini que han quedat pendents i que, o s'estaven desenvolupant però per falta de temps no s'han acabat, o estaven a la llista de coses per a desenvolupar en les següents iteracions del *scrum*, com ara:

- Fer els següents nivells.
- Millorar els temps d'atac dels zombis per fer la supervivència més complexa i alentir la velocitat de l'heroi just després que l'ataquin i quan tingui poca vida.
- Vetar poder caminar per determinades zones del mapa, és a dir, posar obstacles, com ara cotxes, edificis, roques, arbres, bancs, etc.
- Que no hi hagi bales infinites, que hi hagin carregadors amb límit de bales i activar el botó per recarregar.
- Dissenyar un nou sistema per a fer els atacs cos a cos d'empènyer ja que no es pot fer la zona de lliscament.
- Crear zombis diferents, amb diferent *sprite*, velocitats de moviment i d'atac, dany d'atac, rang de visió, etc.
- Crear armes diferents, que es poguessin anar recollint pels nivells i, en un futur més llunyà, que es poguessin comprar.
- De la mateixa manera que amb les armes, poder trobar-se objectes que proporcionin vida a l'heroi.

6. TREBALL FUTUR

- Afegir més sons de trets de bales per cada arma perquè sonés un so aleatòriament cada cop, així com sons d'atac dels zombis, i sons per la seva mort.
- Crear un sistema per auto-apuntar per l'heroi, és a dir, que les bales vagin dirigides automàticament cap als zombis si l'usuari dispara molt a prop d'ells però realment no li arriben a donar. Només seria una petita ajuda.
- Que hi hagi un petit desviament de les bales si dispares molt seguit un arma ja que en la vida real les armes tenen un retrocés que fa que et vagis desviant de mica en mica.
- Tractar d'aconseguir dispositius mòbils on l'aplicació tingui errors degut a la seva arquitectura de software concreta i corregir-los.

7. BIBLIOGRAFIA

- https://www.acamica.com/clases/394/android-ataca/ – Curs d'*Android*.
- http://developer.android.com – Referències d'*Android*, documentació de l'*API*, guies bàsiques d'ús, etc.
- http://stackoverflow.com/ – Fòrum de dubtes concrets d'informàtica.
- http://www.javacodegeeks.com/tutorials/android-tutorials/android-game-tutorials/ – Petit tutorial sobre videojocs en *Android*.
- http://www.kilobolt.com/game-development-tutorial.html – Petit tutorial sobre videojocs i *Android*.
- I moltes altres webs i algun apunt de la carrera que m'han servit per resoldre dubtes molt puntuals.

ANNEX: GLOSSARI DE TERMES

- Array: Llista ordenada d'objectes.
- Bug: Error de software.
- Callback: *Event* de devolució de crida d'un mètode.
- Cyberpunk: Subgènere de ciència ficció on es presenta un futur distòpic amb alta tecnologia i baix nivell de vida.
- Debug: Acció de treure *bugs*.
- DLC: *Downloadable Content*, o contingut descarregable.
- Event: Acció exterior detectada per un software de forma asíncrona amb aquest.
- Fling: Llançament d'un dit per la pantalla.
- FPS: *Frames per second*, o imatges per segon.
- GUI: *Graphical User Interface*, o interfície gràfica d'usuari.
- IDE: *Integrated Development Environment*, o entorn de desenvolupament integrat.
- Joystick: Palanca de comandament.
- LIFO: *Last In, First Out*, o el primer que entra és el primer que surt, una *stack* o pila.
- Modding: Tècnica per a personalitzar nivells del joc (aplica a altres camps).
- Model 3D: Geometria de punts en 3 dimensions que conformen una figura en l'espai amb una textura que la recobreix, i un material.
- Polimorfisme: Capacitat d'un objecte en transformar-se en altres classes diferents heretades d'aquesta, és a dir, la instància pot adquirir vàries formes.
- Scrum: Model de desenvolupament àgil i incremental basat en iteracions curtes.
- SDK: *Software Development Kit*, o conjunt d'eines pel desenvolupament de software.
- SO: Sistema Operatiu (*OS*, en anglès, *Operating System*).
- Sprite: Imatge que representa algun objecte en un videojoc.

ANNEX: GLOSSARI DE TERMES

- Thread: Fil d'execució o sub-procés, la unitat de processament més petita que pot planificar un sistema operatiu.
- XML: *eXtensible Markup Language*, és un llenguatge utilitzat per emmagatzemar dades de forma llegible naturalment.