GalaxyDefenders: Desenvolupament d'un *side-scrolling shooter* per iOS

Memòria del projecte d'Enginyeria Tècnica en Informàtica de `Sistemes` realitzat per

Sergi Garcia Campoy
i dirigit per
Diego Javier Mostaccio
Escola d'Enginyeria

Sabadell, *setembre* de *2012*

El sotasignat, ***Diego Javier Mostaccio***, professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:
Que el treball al que correspon la present memòria

ha estat realitzat sota la seva direcció per

Sergi Garcia Campoy
I per a que consti firma la present.

Sabadell, ***setembre*** de ***2012***

Signat: ***Diego Javier Mostaccio***

FULL RESUM-PROJECTE DE FINAL DE CARRERA DE LA ESCOLA D'ENGINYERIA

Títol del projecte: GalaxyDefenders: desenvolupament d'un *side-scrolling shooter* per iOS	
Autor: Sergi Garcia Campoy	**Data:** Setembre 2012
Tutor: Diego Javier Mostaccio	
Titulació: Enginyeria Tècnica en Informàtica de Sistemes	
Paraules clau: Enginyeria Tècnica en Informàtica de Sistemes ● Català: videojoc, matamarcians, iOS, iPhone, Cocos2D ● Castellano: videojuego, matamarcianos, iOS, iPhone, Cocos2D ● English: videogame, side-scrolling shooter, iOS, iPhone, Cocos2D	

Resum del projecte:

● Català:
Aquest projecte neix del interés personal de l'autor per a la indústria dels videojocs.
El principal objectiu de l'aplicació és esdevenir un joc complet mitjançant l'ús de tècniques clàssicament empleades en el desenvolupament de jocs en dues dimensions de manera que la experiència i tècniques adquirides sigui el més reutilitzable possible.

● Castellano:
Este proyecto nace del interés personal del autor por la industria de los videojuegos.
El principal objetivo de esta aplicación es la de convertirse en un juego completo mediante el uso de técnicas clásicamente empleadas en el desarrollo de juegos de dos dimensiones de manera que la experiencia i técnicas adquiridas sean lo más reutilitzables posible.

● English:
This project stems from the author's personal interest in the videogame industry.
The main goal of this application is to become a full game using techniques traditionally used in the development of two-dimensional games so acquired experience and skills are as reusable as possible.

Índex de continguts

1. Introducció

1.1. Sobre la memòria

Aquest document és cabdal en la comprensió del projecte que es presenta doncs en ell s'hi expliquen els seus principals objectius i es detalla el procés que s'ha seguit per tal d'assolir-los.

1.2. Descripció del projecte

El projecte el·laborat consisteix en el desenvolupament d'un *side-scrolling shooter*, el que vulgarment coneixem com 'matamarcians', per a *iOS* que és el sistema operatiu dels dispositius mòbils d'Apple. (iPhone, iPad, iPod Touch).S'ha tractat que el resultat final del joc tingui el millor acabat possible com si realment anès a publicar-lo a la *AppStore* per comercialitzar-lo.

Al llarg d'aquest projecte es tracten les tècniques més típicament empleades en els videojocs de dues dimensions: animació d'*sprites*, detecció de col·lisions, generació de partícules, tècniques de *scroll* del Background, l'el·laboració d'un gestor de nivells independent del codi font del programa, etc.

El resultat que s'espera obtenir és una aplicació d'iPhone estable que al engegar-la ens permeti començar una partida. La partida a grans trets consisteix en:

* Controlem una nau situada inicialment al mig esquerra de la pantalla.

* Inclinant el telèfon endavant i endarrere podem desplaçar la nau amunt i avall.

* Prement la pantalla tàctil la nostre nau dispararà un làser en linea recta cap a la dreta.

* Haurem d'atravessar un cinturó d'asteroides i acabar amb hostis OVNIS que ens sortiran al nostre pas.

* Esporàdicament apareixeran *powerups* o bonificacions que el usuari podrà agafar per guanyar inmunitat durant uns segons i situar-se en la trajectòria de les naus enemigues.

1.3. Objectius genèrics

El principal objectiu d'aquest projecte, com ja he comentat abans, és experimentar el procés de la creació d'un videojoc, per iPhone en el cas que ens ocupa, desde zero. Aprenent així els principals mecanismes adoptats en la indústria dels videojocs.

Tanmateix s'ha tractat de fer un joc amb un gestor de nivells fàcilment configurable de forma independent al codi que ens permeti definir què succeix en el joc en tot moment.

1.4. Motivacions personals

La elecció d'aquesta temàtica és l'interés personal en la indústria dels videojocs per part del projectista i l'excel·lent estat en que es troba la indústria de les aplicacions, i concretament els jocs, per *Smartphones.*

Además del repte que em suposa endinsar-me en un entorn copletament desconegut però a la part atractiu per mi, doncs la meva experiència prèvia en el camp tant dels videojocs com de les 'aplicacions mòbils' és nul·la.

El fet de involucrar-se en el procés complet de la creació d'un joc de ben segur satisfarà una inquietud personal que he tingut desde que era un nen pràcticament ademés d'ampliar el meu ventall professional.

2. Estudi de viabilitat

2.1 Introducció

En aquest capítol es presenta l'estudi de viabilitat del projecte que ens ocupa. Això ens brindarà un coneixement sobre la planificació temporal, pressupostària i organitzativa del projecte.

2.1.1. Tipologia i paraules clau

Aquest és un projecte de **desenvolupament** de software.

Paraules clau: videojoc, matamarcians, iOS, iPhone, COCOS2D

2.1.2. Objectius del projecte

Els objectius que es pretenen assolir amb aquest projecte són els següents:

O1. Utilitzar la tècnica de ParallaxScrolling per donar sensació que el jugador avança en l'espai.

O2. Desenvolupar un sistema de control que utilitzi l'acceleròmetre del iPhone.

O3. Que el joc sigui "amigable" i senzill d'utilitzar, és a dir que el jugador trigui pocs minuts en assolir la mecànica del joc.

O4. Fer un joc variat que a mesura que s'avanci en els nivells que el composin es vagin afegint nous reptes i noves mecàniques.

O5. Complir els requeriments que Apple imposa a totes les aplicacions que es pretén publicar a la AppStore, tot i que no tingui intenció de comerciar amb aquesta aplicació en concret.

O6. Implementar un mètode que permeti amb facilitat el disseny de nous nivells o el canvi dels diferents paràmetres dels elements que intervenen en el joc sense haver de modificar el codi.

O7. Implementar un sistema de detecció de col·lisions.

O8. Implementar un mecanisme de Powerup o bonificació que el jugador podrá recollir per fer facilitar-li l'avanç.

O9. Crear més d'un tipus d'enemics que interactuin de diferent forma amb l'entorn o l'usuari.

O10.Afegir efectes especials per els principals events del joc: explosions, powerup, etc.

2.1.3. Classificació dels Objectius

	Crític	Prioritari	Secundari
O1	X		
O2	X		
O3	X		
O4		X	
O5			X
O6	X		
O7			X
O8			X
O9	X		
O10			X

2.1.4. Definicions, acrònims i abreviacions.

A continuació enumeraré les definicions de tots els acrònims, abreviacions i paraules tècniques o estrangeres que apareguin al llarg del document.

1. Matamarcians: gèneric del joc que serà programat al llard d'aquest projecte. És un clàssic dins la indústria del videojoc i consisteix en un tipus de jocs en que el jugador controla, normalment, una nau espacial i ha d'enfrontar-se a varis reptes i enemics.

2. Smartphone: dispositiu mòvil d'última generació, habitualment, dotat d'una pantalla tàctil, internet, etc.

3. iPhone 4: *smartphone* de la marca Apple. Aquesta és la plataforma per la que es programarà el joc.

4. iOS: Sistema operatiu que utilitzen els dispositius mòvils d'Apple: iPhone, iPod Touch i iPad.

7. (SDK) Xcode: l'Xcode és el SDK per el disseny d'aplicacions tant per *iOS* com per Mac OS (el sistema operatiu que utilitzen els ordinadors d'Apple), és a dir, és l'entorn de programació que ens proporciona les eines necessàries per escriure aplicacions per qualsevol dispositiu Apple.

8. Objective-C: Són els dos llenguatges de programació que, principalment, son empleats en el disseny de jocs per iOS. L'Objective-C és un llenguatge orientat a objecte que està construit com un superconjunt de C.

9. COCOS-2D: el COCOS-2D és un framework escrit en Python per crear jocs en 2 dimensions i presentacions gràfiques.

10. Side scrolling shooter: tipologia específica del joc. Es tracta de jocs en els que consisteix a disparar els múltiples enemics que es puguin presentar en el que s'avança de forma hortizontal fent un scroll del fons de pantalla.

11. Scroll: en la informàtica s'utilitza el terme scroll per definir tipus de desplaçament.

12. Sprite-sheet: és un arxiu d'imatges que conté els diferents fotogrames que composen un o més objectes dins del joc, i que visualitzant la porció

adient de la imatge en el moment precís es com s'aconsegueix les animacions de les il·lustracions en dos dimensions.

13. pList "Property List": Un fitxer de propietats son fitxers senzills escrits usualment en .XML que aporten una forma senzilla d'incorporar al codi la lectura de certs paràmetres de forma que en comptes d'editar el codi per fer determinades tasques sigui suficient editant el plist en qüestió.

14. IDE: L'Entorn Integrat de Desenvolupament és l'eina principal quan ens disposem a escriure el codi font de l'aplicació. Allà és on treballem la major part del temps.

15. Framework: Marc de treball. Son aquelles col·leccions de biblioteques que ens servim per dotar al nostre IDE d'una major funcionalitat. Com, en el nostre cas, el Cocos2D.

2.1.5. Parts interessades.

Al tractar-se d'un joc i ser un projecte d'iniciativa pròpia no es pot concretar cap part interessada a nivell d'unitats organitzatives que pugui haver implicades en el projecte. Tampoc és una aplicació destinada a clients com a tal, sino més aviat ens referirem tota l'estona a l'usuari que sempre serà la persona que es disposi a jugar el joc.

Dissenyador, tutor i l'audiència de la meva presentació seran realment les parts interessades del projecte.

2.1.6. Producte i documentació del projecte.

Es lliurarà una aplicació per iPhone que podrá ser simulada en un ordinador d'Apple i es podrà executar, mitjançant una sèrie de procediments, en el meu terminal iPhone 4.

(per tal que l'aplicació pogués ser instalada a qualsevol iPhone

caldria pagar la llicència de desarrollador d'Apple que son $99 i

además esperar l'aprovació de la companyia).

S'elaborarà una memòria del projecte.

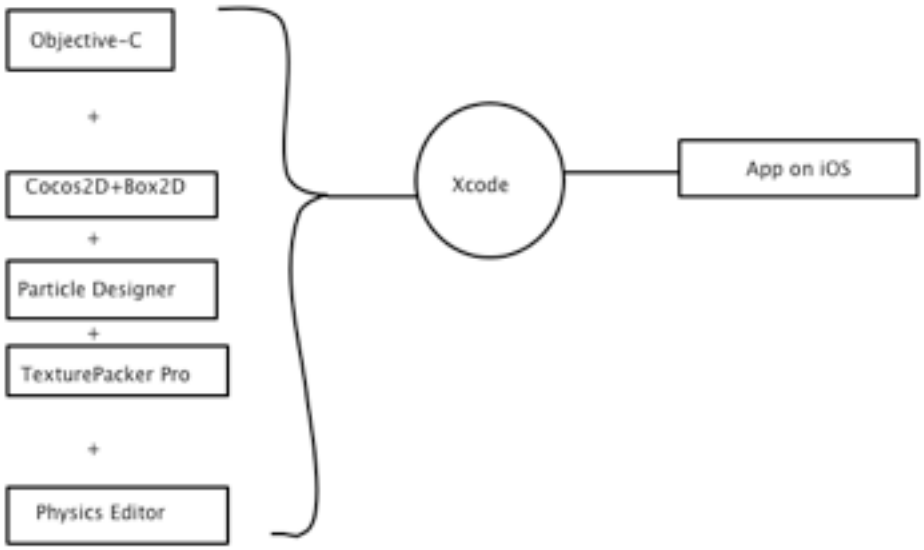
2.2. Estudi de la situació actual

Aquest projecte serà dissenyat i el·laborat desde zero de la manera que millor s'adapti als requeriments i objectius fixats.

2.2.1. Context

No hi ha cap context previ doncs, com he esmentat amb anterioritat, es tracta d'un projecte que parteix de zero.

2.2.2. Lògica del sistema.



2.2.3. Descripció física.

iPhone 4:

CPU: 1GHz Apple A4 chipset.

Gràfics: PowerVR SGX535GPU (integrat al chip A4)

Suporta OpenGL ES 2.0 i DirectX 10.1 amb *Shader Model 4.1*.

Memòria: 512 MB de memòria RAM.

Giroscopi de 3 eixos.

Acceleròmetre.

Pantalla tàctil *multi-touch* de 3,5 polzades i panel IPS, amb una superfície resistent a ratllades i amb una resolució de 960x640 píxels.

Contrast de 800x1. La pantalla anomenada *Retina Display* s'anomena

així degut a la alta densistat de píxels, propera a la de l'ull humà amb

326 pixels per polzada.

2.2.4. Usuaris i/o personal del sistema.

Usuari (jugador): és l'entitat a qui va dirigida l'aplicació. Aquesta aplicació ha d'ésser senzilla d'utilitzar i que qualsevol tipus d'usuari pugui estar-ne interessat.

Dissenyador i programador: aquest és el rol que prenc jo personalment ademés del de cap del projecte. La meva tasca serà la completa el·laboració del projecte i documentar tot el transcurs de la manera que se'ns exigeix.

Tutor: el tutor del projecte, en el meu cas el senyor Diego Mostaccio, adopta un rol de supervisió de que es vagin assolint els objectius marcats i que aquests, siguin de l'exigència que cal esperar per un projecte de final de carrera.

2.2.5. Diagnòstic del sistema.

Donat la naturalesa del projecte no té identificats problemes, inconvenients, deficiències, millores... el que si que voldria fer en aquest apartat és enfatitzar el comentat anteriorment quan parlava del rol d'"usuari", i és que la corba d'aprenentatge ha de ser assequible per tota mena d'usuaris, si que cap la possibilitat que s'implementin diferents nivells de dificultat o que aquesta sigui creixent durant el transcurs del joc, però d'entrada la mecànica del joc té que ser senzilla i assequible per tota mena de publics.

2.2.6. Normatives i legislació.

Normativa de projectes de final de carrera de l'EI.

Llei de propietat intelectual.

2.3. Requisits del sistema

En aquesta secció procediré a explicar els requisits del sistema que desenvoluparé, els classificaré i també en mencionarem les restriccons

2.3.1. Requisits funcionals.

1. S'han de diferenciar nivells on l'usuari s'enfronti a diferents tipus d'enemics que suposin un repte diferent a l'hora d'enfrontar-los.

2. Es podrà començar una nova partida.

3. L'avanç entre les diferents fases ha de ser continu i dinàmic.

4. Es diferenciaran events que determinaran la victòria o la derrota en la partida.

5. Un cop finalitzada la partida, de la manera que sigui, es donarà opció de reiniciar la partida.

6. Es podrà consultar les puntuacions obtingudes.

7. Es donarà l'opció de publicar puntuacions obtingudes a les principals xarxes socials o reptar algú per e-mail.

2.3.2. Requisits no funcionals.

1. Compliment de la normativa d'Apple per la publicació d'aplicacions al seu AppStore.

2. El joc ha de ser fàcilment ampliable o modificable.

3. Tolerància a errades i accions incorrectes.

2.3.3. Restriccions del sistema.

L'aplicació es podrà executar correctament en un iPhone 4 o superior.

(Perquè l'aplicació pugui ésser executada en qualsevol terminal caldria que adquirís la llicència de desarrollador d'Apple [$99], però per a poder fer una demostració durant la defensa del projecte he modificat el IDE Xcode i el meu terminal iPhone 4 per eludir aquesta restricció).

L'aplicació ha de ser desenvolupada en un ordinador de la marca Apple, doncs és la única plataforma que disposa de l'SDK necessari per programar aplicacions natives per qualsevol mena de dispositiu de la companyia.

El projecte es lliurarà abans del 19 de setembre del 2012.

2.3.4. Catalogació i priorització dels requisits.

Prioritats dels requisits funcionals:

	RF1	RF2	RF3	RF4	RF5	RF6	RF7
Essencial	X	X		X	X		
Condicional			X				
Opcional						X	X

Prioritats dels requisits no funcionals:

	RNF1	RNF2	RNF3
Essencial		X	X
Condicional			
Opcional	X		

Relació entre els objectius i els requisits:

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RNF 1	RNF 2	RNF 3
O1			X						X	
O2										X
O3	X		X						X	X
O4	X			X	X					
O5								X		
O6	X		X						X	X
O7									X	
O8	X					X	X			
O9	X					X	X		X	
O10										

2.4 Alternatives i selecció de la solució

A continuació s'exposaran les diferents alternatives disponibles per el disseny de l'aplicació amb les seves principals mancances i beneficis i finalment es justificarà la opció escollida.

2.4.1. Alternativa 1

Desenvolupament propi de les eines gràfiques, dels processos per moure sprites, de les físiques, etc. directament en el SDK: Xcode.

Aquesta opció seria molt costosa a nivell de temps doncs hauria d'invertir molt de temps en mecàniques que d'altres solucions que veurem a continuació porten ja implementades.

El cost del desenvolupament es gratuït sempre que es treballa amb l'Xcode, l´únic que és de pagament és la possibitat de poder publicar aplicacions a la AppStore i poder executar-les de forma legal fora del simulador que el SDK incorpora. Però el problema que el desenvolupament de les funcions que tractessin les físiques, sprites, partícules, etc seria quasi un projecte per si sol.

2.4.2. Alternativa 2

Ús d'un *framework* orientat a la multiplataforma per fer aplicacions compatibles amb els principals *smartphones* del mercat.

Actualment es disposa de frameworks molt bons escrits en HTML5 que además de multiplataforma estan bastant a la última.

El problema d'aquesta proposta és que pel joc que ens ocupa es fa ús de l'acceleròmetre i, habitualment, quan s'ha de tractar amb elements tan específics del telèfon l'ús de frameworks multiplataforma pot tornar-se excessivament complex.

2.4.3. Solució proposada.

La solució que he proposat finalment per desenvolupar el projecte és fer ús de Xcode com a IDE, amb el framework Cocos2D en la seva variant Box2D. Aquest "paquet" ademés de ser gratuït em proporciona multiples estructures de dades de gràfics, polígons, partícules, etc. que son empleades en la indústria dels videojocs amb molta freqüència.

Per no haver d'exercir de dissenyador i que la meva aplicació tingui un aspecte decent he adquirit, ademés del paquet ja esmentat, la llicència d'uns programes que enunciaré a continuació que m'han lliurat de complexos processos per tractar la part més artística del projecte.

Aquests programes son:

- Pixelmator
- TexturePacker Pro
- Physics Editor
- Particle Designer

El cost de les llicències d'aquests programes es desglossarà al capítol reservat per a tal efecte. Tampoc explicaré en què consisteix cadascún perquè hi he dedicat un apartat del capítol tercer "Anàlisi".

2.4.4. Conclusions

Beneficis

Com ja veure'm més endevant, tot i que el cost del projecte en quan a hores de treball dels diferents participants podria resultar elevat, el cost del projecte real que ha estat només el de l'adquisició de les llicències d'aquells programes no gratuïts dels que m'he servit per desenvolupar el joc.

Podran comprovar que el cost d'aquestes aplicacions no és gens elevat si som conscients de la complexitat què implicaria obtenir resultats semblants si m'hagués de crear les meves pròpies eines de disseny.

Inconvenients

La dificultat de fer que l'aplicació destaqui entre el nombrós catàleg de jocs i aplicacions de que disposa l'App Store de iOS.

Enfrontar una nova disciplina, el desenvolupament de videojocs, en un entorn completament desconegut per mi, el marc de treball de la programació per a *iOS.*

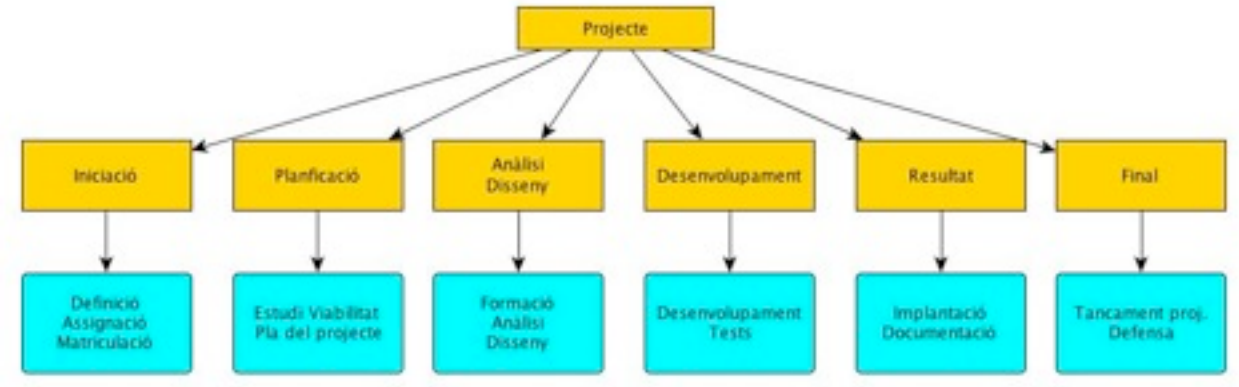
2.5. Pla del projecte

En aquesta segona meitat del segon capítol procediré a explicar el pla del projecte. És a dir les activitats que el composen, una previsió temporal del mateix, un pressupost, avaluació de riscos, etc.

2.5.1. Fases i activitats

Fases	Descripció
Iniciació	Fase d'iniciació. Inclou les activitats definició del projecte, assignació i matriculació.
Planificació	Inclou l'Estudi de Viabilitat i el Pla del projecte
Formació	Inclou una fase d'aprenentatge del llenguatge i les tècniques que s'emplearan.
Anàlisi	Anàlisi de requisits funcionals i no funcionals. Arquitectura del sistema.
Disseny	Inclou el disseny de la capa de dades, de control i l'interfície. Disseny dels tests.
Desenvolupament	Fase de desenvolupament de l'aplicació.
Test i proves	Fase de prova de l'aplicació. Inclou tests unitaris i d'Integració.
Implementació	L'aplicació s'instal·la en el seu entorn real.
Generació de documents	Fase de documentació del projecte. Inclou manuals i memòria del projecte.
Tancament del projecte	Fase de tancament. El director del projecte signa l'acceptació i tancament del projecte.
Defensa del projecte	Defensa del projecte davant la comissió.

2.5.2. Diagrama WBS



2.5.3. Milestones

Nom	Descripció	Data
Iniciació	Matriculació	~10/10/2011
Est. Viabilitat	Aprovació	25/12/2011
Pla del projecte	Aprovació	02/12/2011
Formació completada	Aprovació	25/07/2012
Anàlisi	Aprovació	1/08/2012
Disseny	Aprovació	12/08/2012
Tancament	Acceptació	19/09/2012
Defensa	Avaluació	?

Com podre'm comprovar degut a circunstàncies personals no s'ha acomplert la previsió inicial d'acabar el projecte al juny de 2011 i les parts de Anàlisi, disseny, tancament, redacció de la memòria i defensa s'han realitzat entre Agost i Setembre de 2012.

2.6. Recursos del projecte

2.6.1. Recursos

Recursos humans i materials:

Recursos humans	Valoració
Cap de projecte	50€/h
Analista	25€/h
Programador	12€/h

Recursos materials: preu (ús)

Pages (Apple): 14.99€ (documentació)

Keynote (Apple): 14.99€ (documentació i presentacions)

PixelMator: 15.99€ (edició d'imatges)

TexturePacker Pro + Physics Editor Bundle:33.90€ (Sprite sheets i físiques)

Particle Designer: 6.30€ (edició de partícules)

* Cal esmentar que en cas de voler publicar l'aplicació per a la seva comercialització s'hauria d'abonar la llicència de *developer* d'Apple de un any amb un cost de 80€. Aquest cost no serà computat ja que no s'ha adquirit.

Apple Mac Mini del 2011 (propietat del projectista): 650€

Apple iPhone 4 (propietat del projectista): 389€

2.6.2. Calendari dels recursos

Els recursos s'utilitzaran al llarg de tot el projecte exceptuant "Pages" i "Keynote" que seran empleats en l'el·laboració de la memòria i la presentació respectiva i exclusivament.

2.6.3. Calendari del projecte

Calendari del projecte: el projecte es desenvoluparà d'octubre de 2011 fins al juny del 2012. La dedicació en hores serà incremental assolint les 12 hores setmanals al segon semestre. El total d'hores dedicades al projecte serà de ~250 hores.

Data començament: 10 d'octubre de 2011.

Data finalització: <= 18 de setembre 2012.

Eines de planificació i control: Microsoft project 2007.

* En els annexos A i B trobarem el "Quadre de tasques del projecte" i el "Calendari Temporal" del mateix respectivament.

2.6.4. Dependències

Un cop arribat a la fase de formació i anàlisi totes les fases restants seguiran un model lineal i per tant cada fase començarà quan acabi l'anterior. Les fases d'anàlisi, disseny, desenvolupament i test seguirant un model iteratiu. I la fase de documentació es completarà a cotinuació d'aquestes però s'anirà el·laborant i preparant al llarg de tot el projecte.

2.7. Avaluació de riscos

2.7.1. Llista de riscos

R1. Planificació temporal optimista: Pla de projecte. No s'acaba en la data prevista, augmenten els recursos.

R2. Manca alguna tasca necessària: Pla de projecte. No es copleixen els objectius del projecte.

R3. Canvi o ampliació de requisits: estudi de viabilitat, anàlisi. Endarreriment en els desenvolupament i resultat.

R4. Eines de desenvolupament inadequades: desenvolupament. Endarreriment en la finalització del projecte, menys qualitat...

R5. No es fa correctament la fase de test: desenvolupament, implantació. Manca de qualitat, deficiències operatives, insatisfacció usuaris.

R6. Incompliment d'alguna norma, reglament o legislació: en qualsevol fase. No es compleixen els objectius, repercussions legals.

R7. Manca d'adopció de mesures de seguretat: estudi de viabilitat, anàlisi, desenvolupament. Pèrdua d'informació, incompliment legal.

R8. Abandonament del projecte abans de la finalització: en qualsevol fase. Frustració, suspens, pèrdues econòmiques.

2.7.2. Catalogació de riscos

	Probabilitat	Impacte
R1	Alta	Crític
R2	Alta	Crític
R3	Alta	Marginal
R4	Baixa	Crític
R5	Alta	Crític
R6	Mitjana	Crític
R7	Alta	Critic
R8	Baixa	Catastròfic

2.7.3. Pla de contingència

	Solució que cal adoptar
R1	Ajornar alguna funcionalitat, augmentar el temps de treball.
R2	Revisar Pla de Projecte, modificar la planificació.
R3	Ajornar funcionalitat, modificar planificació i pressupost.
R4	Preveure eines alternatives, millorar la formació del equip, millorar la qualitat.
R5	Dissenyar els test amb antel·lació, realitzar tests automàtics.
R6	Revisar les normes i legislació, consultar un expert, afrontar possibles repercussions penals.
R7	Revisar la seguretat en cada fase, aplicar polítiques de seguretat actives.
R8	No té solució, no ha d'ocórrer en cap cas.

2.8. Pressupost

2.8.1. Estimació cost de personal.

Cap de Projecte	83h	4150 €
Analista	372.9h	9323 €
Programador	527.4h	6329 €

El cost total en personal seria de **19.802€**. Pot semblar excessiu però cal tenir en compte que he estat 'generós' en la quantitat d'hores del analista i el programador degut a què com he comentat era un entorn nou per mi que ha requerit d'una fase de formació prorrogada.

Si es tornés a realitzar un projecte del mateix caire un cop acabat aquest, seria molt més capaç de estimar amb precissió les duracions de cada tasca i la fase de formació seria molt menys extensa.

2.8.2. Estimació cost dels recursos

(preu expressat en Euros)

Recurs	Tipus	Preu
Pages	Llicència software	14,99
Keynote	Llicència software	14,99
Pixelmator	Llicència software	15,99
TexturePacker+Physics Editor	Llicència software	33,90
Particle Designer	Llicència software	6,30
Mac Mini 2011	Hardware	650
iPhone 4	Hardware	389
Total:		**1125,17**

2.8.3. Resum i anàlisi cost benefici.

Cost aproximat	Quantitat
Personal	19.802
Recursos	1125,17
Total:	**20.927,17**

Com ja he comentat anteriorment, el cost pot semblar elevadíssim derivat de les hores previstes i els sous del personal. Però tan sols ho he fet de forma ficticia doncs realment, els 3 rols els assumiré jo mateix de franc, per tant el cost total de publicar l'aplicació a l'AppStore seria de poc més de 100 €.

Si suposem que comercialitzés l'aplicació un 70% del preu de l'aplicació serien els beneficis que correspondrien ja que el 30% restant, va a parar a Apple en concepte de proporcionar la plataforma de venta.

Tot i que no comercialitzaré la aplicació en la pràctica i alguns costos serien ficticis doncs el hardware ja el tenia en el moment de començar el projecte

així com també algunes de les llicències de software, realitzaré un anàlisi d'amortització de l'aplicació amb el cost estimat.

Si comercialitzés l'aplicació **0.79€** correspondrien **0.55€** de benefici. Per tant a partir de les **38.050** descàrregues de l'aplicació començaríem a obtenir beneficis.

2.9. Conclusions

S'han determinat les fases, activitats principals i punts de control del projecte.

S'han representat gràficament utilitzant WBS.

S'han valorat els recursos del projecte.

S'ha generat el calendari del projecte incloent el diagrama de Gantt.

S'ha avaluat els riscos del projecte i s'ha preparat un pla de contingència.

S'ha determinat el pressupost del projecte.

S'ha analitzat el cost del projecte en relació als beneficis esperats.

Donat l'alt índex de dèscarregues que hauríem d'obtenir per amortitzar els **20.927,17€** que ens costaria desenvolupar aquesta aplicació no seria mala idea buscar opcions de marketing comercial i demés promocions per obtenir beneficis adicionals.

A part queda demostrat que no seria barat contractar personal per realitzar la tasca i no tenim cap garantia de que la nostra aplicació sigui descarregada per lo que si algún dia faig alguna incursió en aquest mercat suposo que serà com ha sigut el projecte en realitat que he hagut de assumir jo tots els rols. En aquest cas donat el baix cost de materials, sobretot si aprofite'm les eines d'ús personal que tinguem, fa d'aquest model de negoci un model atractiu.

Doncs si tornés a fer un projecte similar per comercialitzar, el cost de recursos també me l'estalviaria i començaria a obtenir beneficis un cop amortitzats els irrisoris 80€ que costa la llicència *developer* d'Apple.

3. Anàlisi

Aquest capítol consta d'una introducció de les funcionalitats que composen el joc que he desenvolupat. El *side-scrolling shooter* que ens ocupa ha estat dissenyat per tal d'assolir una sèrie de tècniques que son empleades habitualment en la indústria del videojoc en el desenvolupament de jocs de dues dimensions.

El que es pretenia aconseguir, i més endevant valoraré si s'ha aconseguit, és el realitzar un joc el més complet possible de forma didàctica. Hi ha molts aspectes del joc que es podrien haver aconseguit de formes més senzilles o empleant altres eines, per tant en la elecció del mecanisme a afrontar un aspecte del joc a dissenyar s'ha prioritzat el fer ús de mètodes que puguin ésser reutilitzats en d'altres projectes.

Com s'ha esmentat en l'anàlisi de costos s'ha optat per utilitzar ademés del *IDE* que Apple disposa per a programar aplicacions per als seus productes: *Xcode* amb el framework de Cocos2D, gratuït, s'ha comprat la llicència de les aplicacions *Particle Designer, Texture Packer, Physics Editor* i *Pixelmator*.

3.1. Anàlisi de les eines utilitzades

Amb l'adquisició de les aplicacions abans esmentades s'ha contribuït notablement a reduïr de forma necessària la complexitat del projecte i no haver d'abordar directament temes, que ademés excessivament complexes, es desvien del objectiu principal del projecte.

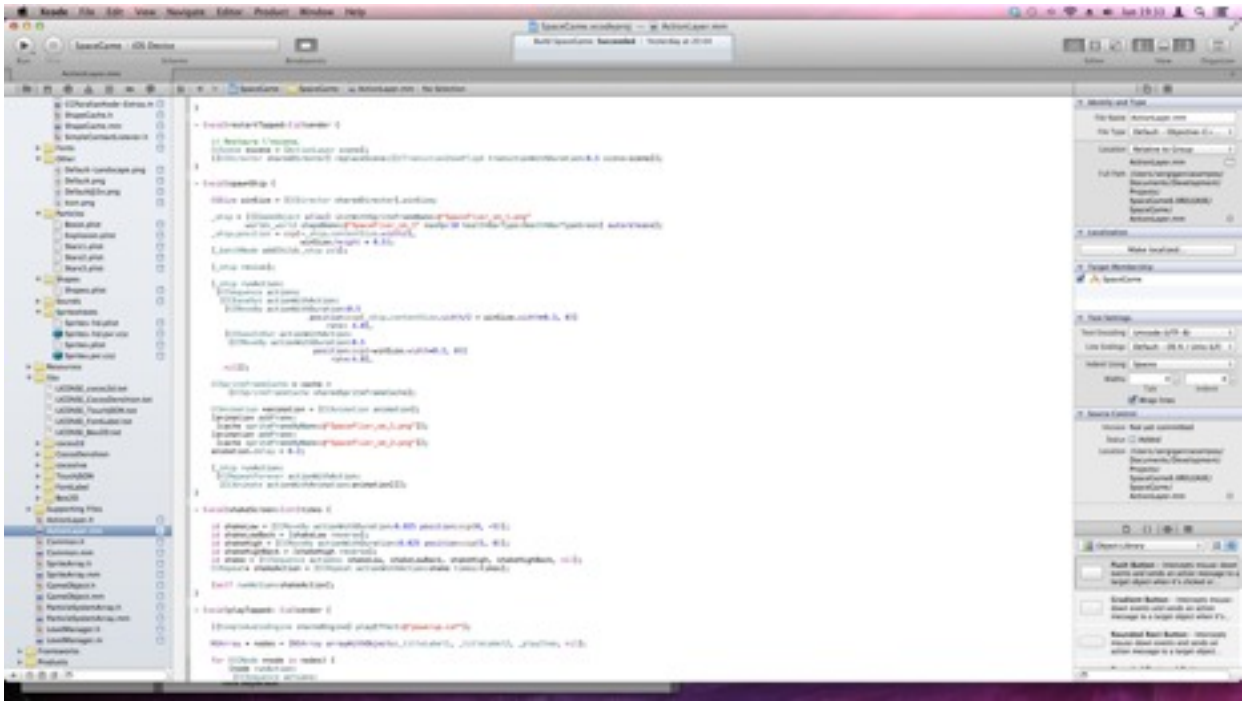
A continuació analitzarem aquestes eines i com ens hem servit d'elles a la hora d'el·laborar el projecte abans d'aprofundir més en detall en l'anàlisi de les funcionalitats que composen el joc.

Les eines comprades m'han permès centrar-me en la programació de l'aplicació i sense prescindir d'un bon aspecte visual no haver de desviar-me en excès del pretès.

3.1.1. Xcode

En la següent captura de pantalla podem veure'n l'interfície principal:

Com altres IDEs coneguts està composat per un disseny de panells que es pot personalitzar al gust.



Xcode és l'entorn integrat de desenvolupament (IDE) oficial d'Apple. Aquí no disposem de cap alternativa doncs Apple és molt estricta en aquest sentit i per poder programar per als seus dispositius s'ha d'emplear un ordinador Macintosh i el seu IDE.

Disposa d'una interfície agradable i de pràcticament tota funcionalitat que es pugui requerir a l'hora de desenvolupar aplicacions per dispositius Apple.

El paquet no té cap cost doncs es pot adquirir gratuïtament si es disposa d'un ordinador Macintosh i s'està registrat al programa *developer* d'Apple.

Además de l'IDE en sí incorpora el SDK (Kit de desenvolupament de Software).

Incidència:

Cal esmentar que per a la realització del projecte he hagut de modificar l'aplicació per tal que pugui aceptar certificats no firmats. Això és degut a que Apple exigeix pagar la llicència de *developer* (80€ l'any) per tal de poder testejar les nostres aplicacions fora del simulador, en el nostre terminal. Per estalviar-me aquest cos doncs he fet la modificació esmentada al Xcode i he hagut de desbloquejar també el meu iPhone.

3.1.2. Cocos2D i Box2D

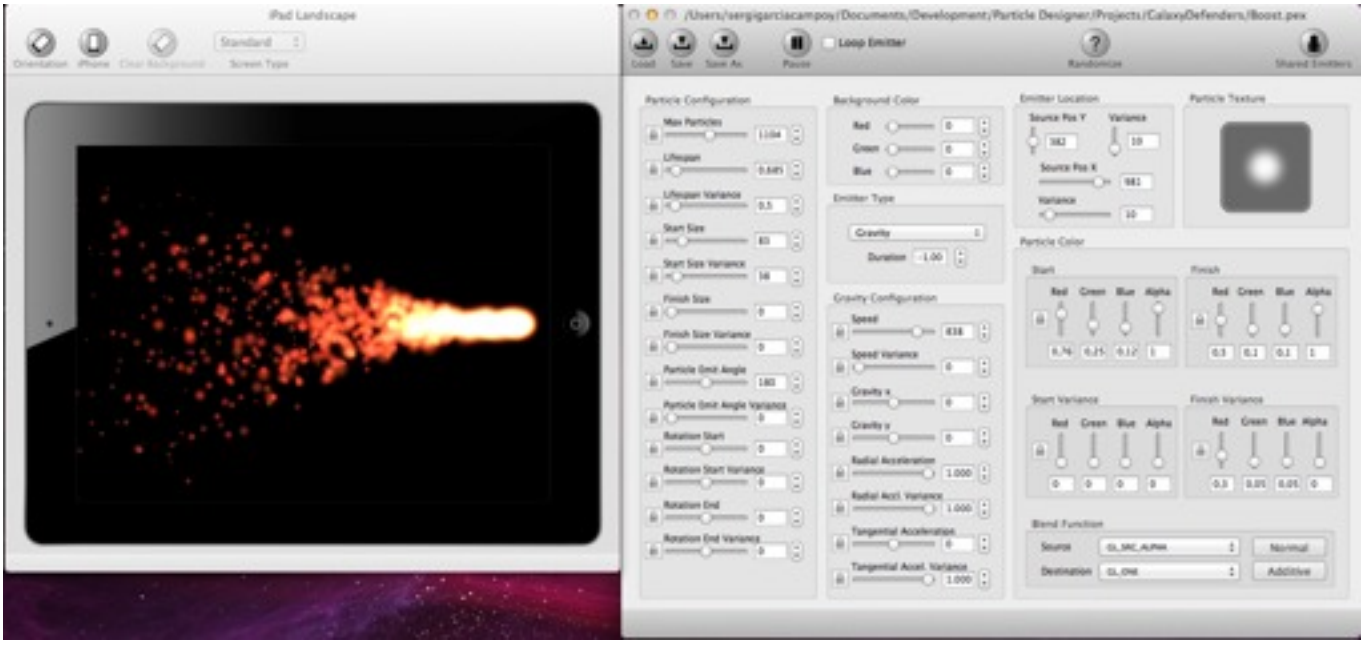
A continuació els parlaré del framework utilitzat: Cocos2D. I concretament de les 3 plantilles que ens ofereix he escollit la Box2D que ens proporciona una sèrie de funcionalitats que ens faciliten molt la feina a l'hora de desenvolupar l'aplicació, deslliurant-nos de la preocupació de lidiar a baix nivell amb el tractament de molts elements comunament empleats en l'elaboració de videojocs. Com poden ser: les físiques involucrades, tractament de *Sprites* i *partícules*.

Cocos2D està escrit en Python i incorpora un ampli recurs de funcions, com he esmentat anteriorment, habitualment empleades en la producció de videojocs. Possiblement el 80-85% de l'aplicació fa ús d'aquest framework per lo que és una part fonamental del projecte.

Box2D està escrit en C++ i és una variant de Cocos2D que incorpora una gran biblioteca de definicions de polígons i codi de detecció de col·lisions que emplearem.

Degut a estar en C++ tots els arxius font que guarden alguna relació amb ell tenen el format .mm típic de C++ i no el .m d'Objective-C.

3.1.3. Particle Designer



Aquesta aplicació ens ha permès incorporar de forma molt senzilla al nostre joc complexos gràfics de partícules que donen al joc uns efectes especials aconseguits.

El programa disposa de vàries plantilles amb efectes predefinits de les quals en podem modificar tota mena de paràmetres al nostre gust desde la interfície gràfica.

Un cop hem aconseguit l'efecte desitjat exporta tota la configuració en un fitxer *Property List* que porta la textura incrustada. Tan sols tenim que incorporar aquest fitxer al nostre codi en una estructura de dades que Cocos2D ens proporciona.

En la següent figura podem veure en acció el *Particle Designer* amb l'efecte que he creat per representar el benefici del *PowerUp*.

3.1.4. Pixelmator

Aquesta aplicació ha estat empleada tan sols per eliminar el fons, entre d'altres retocs, de les fotografíes que m'he baixat d'Internet com el cos de la

nau protagonista, asteroides, el *OVNI* que representa la invasió alienígena, *PowerUp,* els planetes i galaxia que he fet servir de fons.

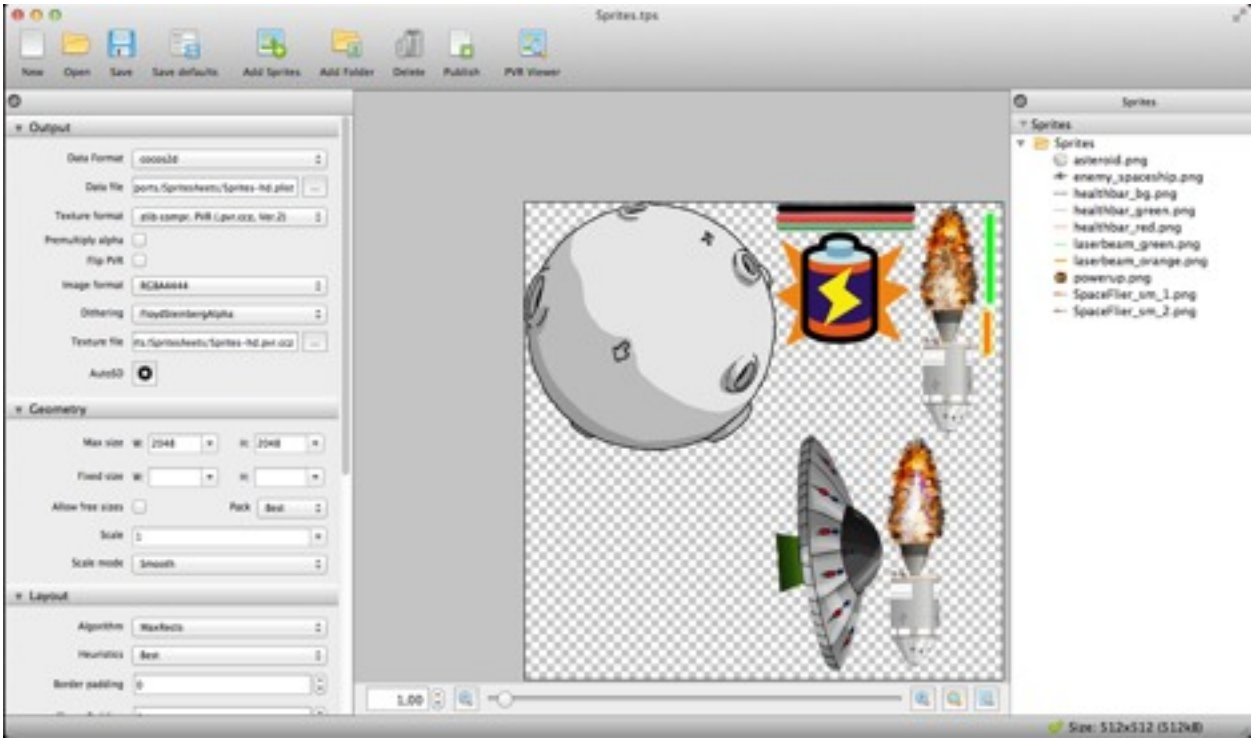
Además de crear elements molt senzills com una capa de boirina amb transparència, els "rajos làser" que dispara la nostre nau i les enemigues, i les barres d'indicador de salut.

3.1.5. TexturePacker Pro

Amb aquesta aplicació m'ha resultat francament senzill disposar d'un *Sprite sheet* amb el què treballar tan sols he hagut de posar tots els arxius d'imatge que anava a emplear per composar l'sprite sheet del meu joc en una carpeta, arrastrar-la al programa i establir una sèrie de paràmetres de format i compressió abans de la exportació.

He triat un format d'arxiu (d'extensió .pvr.ccz) que es composa d'un arxiu que conté el sprite sheet amb les imatges, el qual va molt bé per importar-lo al Cocos2D ja que iutilitza un format de compressió acceptable i es mostra amb un bon rendiment, i un fitxer *Property List* (.plist).

A continuació podem veure una captura de l'aplicació amb l'*Sprite sheet* que he generat per fer ús en el joc.



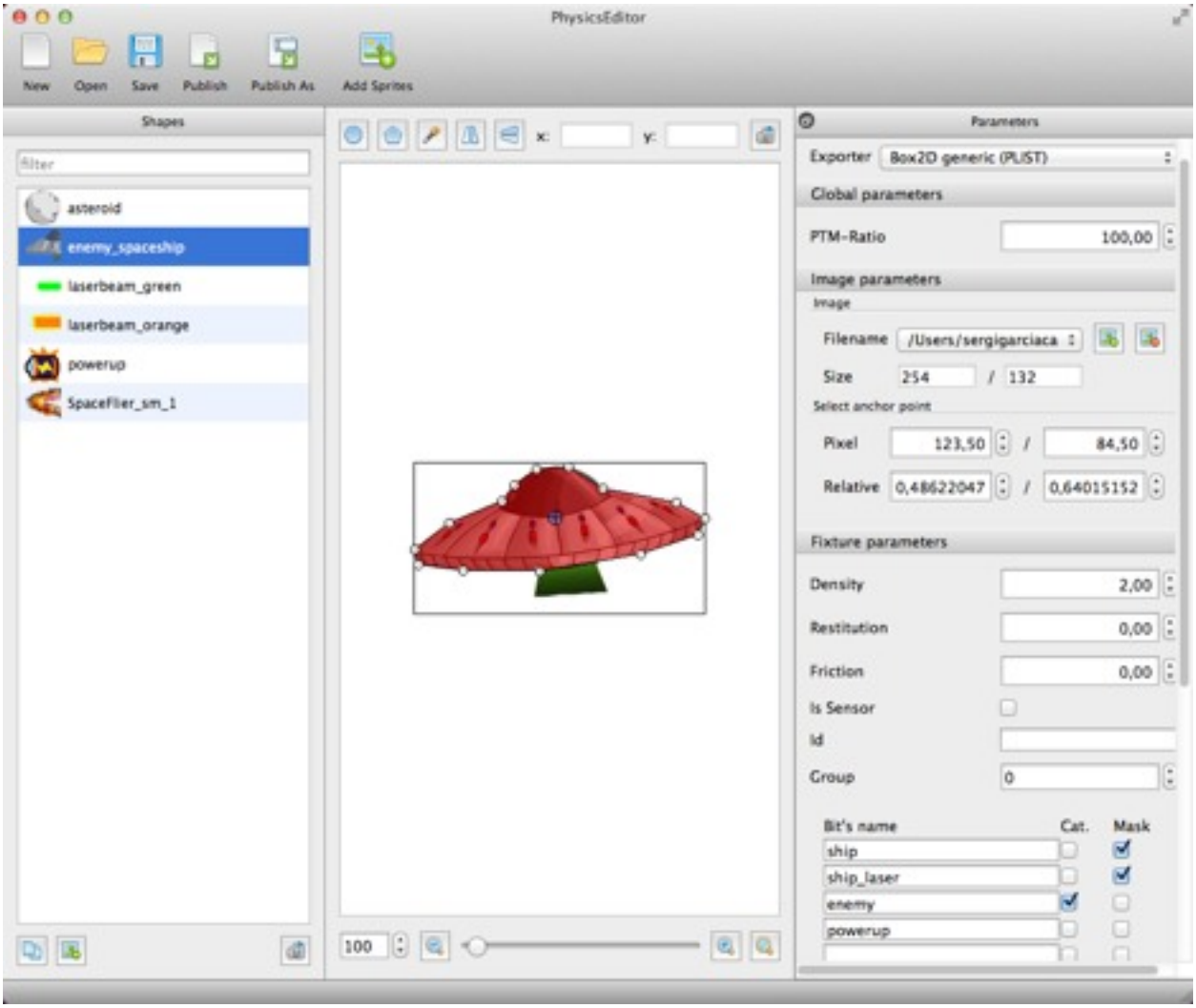
3.1.6. Physics Editor

L'ús d'aquesta aplicació ha estat essencial en la detecció de col·lisions implementada.

El que ens permet aquest programa es dibuixar un polígon amb exactament les dimensions dels nostres dissenys per utilitzar-ne les coordenades que el programa ens genera en un fitxer *plist*. Aquestes coordenades son transferides al *Box2D* que genera els objectes amb les proporcions extretes.

Además d'això ens brinda un sistema d'enmascarat que ens permet crear categories d'elements i les interaccions entre elles. De manera que no només n'extraiem unes coordenades sino una estructura lògica que definim al programa per decidir, per cada element, amb qui pot col·lisionar i amb qui no.

En la següent figura apreciem una captura de pantalla de l'apliació mostrant-nos el polígon que li he dibuixat a les naus enemigues i la interacció que tenen amb els altres elements del joc (en la part inferior del panell de la dreta).



3.2. Introducció de les funcionalitats del joc

En aquest apartat senzillament em disposo a introduïr, molt breument, les funcionalitats que composen aquest joc i que seran detallades en el següent capítol: "Disseny del joc".

Aquestes son, a grans trets, les principals funcionalitats que el nostre joc haurà d'acomplir:

- Al iniciar el programa es mostrarà una pantalla de benvinguda donant opció a començar la partida.

  Es fa ús d'un fitxer de fonts per mostrar el missatge de benvinguda, en el nostre cas concret serà el títol del joc (GalaxyDefenders) i l'autor, ademés del botó per iniciar la partida.

- L'aplicació es mostrarà de forma contínua sense transicions entre les diferents fases que contingui. Per donar efecte de dinamisme.

- El joc ha de generar *sprites* que representaran els elements que interactuen en el joc, aquests son: la nau protagonista, els lásers disparats per ella i les naus enemigues, les citades naus enemigues i asteroides; i un *PowerUp* o bonificació.

  - Recrear una senzilla animació per a la nau protagonista.

- Sistema de control de la nau governat pels moviments que captura l'acceleròmetre del terminal. En comptes de polsar cap botó per fer moure la nau al llarg de l'eix Y s'implementarà un mecanisme perquè el moviment vingui determinat per la inclinació que li apliquem al dispositiu.

- Implementació de *Parallax Scrolling*. Consisteix en un mecanisme que posa diferents elements de fons de pantalla i els fa desplaçar-se a diferents velocitats creant una falsa sensació de profunditat. Aquest és un recurs molt utilitzat en els jocs de dues dimensions desde els temps de les consoles de 8 i 16 bits.

- Ús de Box2D i el fitxer de coordenades generat pel *Physics Editor* per implementar una detecció de col·lisions precisa i definir les lògiques que controlen els paràmetres de salut de cada element, el event de rebre mal, etc.

- L'aplicació ha de tenir un mecanisme per determinar l'èxit o el fracàs de la partida. Després de produir-se l'event de victòria o derrota es deixarà de generar enemics i *PowerUps* i se li donarà al usuari la opció de tornar a la pantalla de títol.

- Es farà ús també del editor de partícules per associar un efecte d'explosió a cada col·lisió diferenciant les que originen la mort d'un dels elements implicats en la col·lisió o no.

- Amedés de generar simples asteroides que l'únic que fan es desplaçar-se de dreta a esquerra de la pantalla, incorporarem un enemic que suposi un repte, si més no, diferent. Es generaran onades de naus alienígenes que amedés de moures amb un patró determinat dispararan lásers tractant de ferir la nostre nau.

- Amedés d'enemics ocasionalment es generarà un *PowerUp* o bonificació, que es mourà de dreta a esquerra "esperant ésser capturada per la nostre nau". La bonificació consisteix en uns segons d'inmunitat i una empenta de la nau que avançarà fins a la meitat de la pantalla aproximadament (en l'eix de les X) per poder col·lisionar amb les naus enemigues.

Com s'ha vist en la captura d'exemple quan parlava sobre el *Particle Designer* amedés es generarà un efecte amb l'ús de partícules per simular l'empenta que rep la nau. I es modifica el zoom de l'aplicació per donar més amplitud de visió en el que dura la bonificació

- L’aplicació disposarà d’una funció per mostrar una barra sota la nau protagonista que representa la salut restant. La barra de vida estarà animada actualitzant-se progressivament a cada impacte. Per defecte, aquesta romandrà oculta a menys que rebem mal, moment en el que serà mostrada durant un parell de segons.

- Es dissenyarà un mecanisme de gestió de nivells mitjançant un fitxer *plist.* El que això ens permetrà és poder definir, fàcil e independentment del codi font del programa, la quantitat de nivells que composen l’aplicació, fases de transició entre nivells mostrant text i la composició de cada nivell. És a dir si s’han de generar asteroides, naus enemigues o ambdues coses i les variables que en governen la peroidicitat entre d’altres coses.

4. Disseny e implementació del joc

El capítol que enceto a continuació serà un repàs a el procés de disseny e implementació del joc. En ell veurem quins han estat els passos que he seguit per el·laborar el joc i quines tècniques i recursos s'han emprat en aquestes fases.

Donada la singularitat del joc he desestimat fer diagrames de classe o de casos d'ús que poc aportarien a la comprensió del procés d'el·laboració del joc així que he preparat una exposició argumental acompanyada de captures de pantalla, quan siguin precises, per tal que es vegi el que es pretenia explicar.

4.1. Creació dels primer elements

Primerament aclariré uns conceptes bàsics de com treballen les aplicacions en iOS.

S'ha de dir que treballarem en programació orientada a objecte així que obviament cada fixer de classe (.m, .mm, etc) tindrà el seu pertinent fitxer de capçalera (.h).

Tenim una classe *scene*, abstracta, de la qual pengen unas classes anomenades *layer* (s). Aquestes classes filles de *scene*, son vulgarment les diferents pantalles que mostrarà l'aplicació i podem definir transicions entre elles amb efectes que ens proporcionen les eines empleades.

En el cas del nostre joc en concret he cregut pertinent desenvolupar tota la acció en una mateixa *layer* per aconseguir un dels objectius marcats: què la partida sigui el més dinàmica i contínua precisa.

Per tant en el nostre projecte l'acció esdevindrà en la classe "ActionLayer.mm".

4.1.1. Pantalla de títol

Vull que al iniciar l'aplicació es mostri una pantalla amb el títol del joc, el nom de l'autor i la opció per començar una partida.

Per fer això faig ús de les funcions que ens proporciona Cocos2D per mostrar etiquetes (a partir d'ara *labels)* agafant el contingut d'un *NSString* que és l'estructura de dades que representa les *strings* a Objective-C.

Durant els inicis d'aquest capítol miraré d'explicar les estructures de dades utilitzades i com es relacionen entre elles. A mesura que avancem i les explicacions es tornin més tècniques i banals per al caire d'un document d'aquesta mena, explicaré de forma més descriptiva el rerefons de la qüestió acompanyant-ho amb captures de pantalla explicatives quan sigui precís.

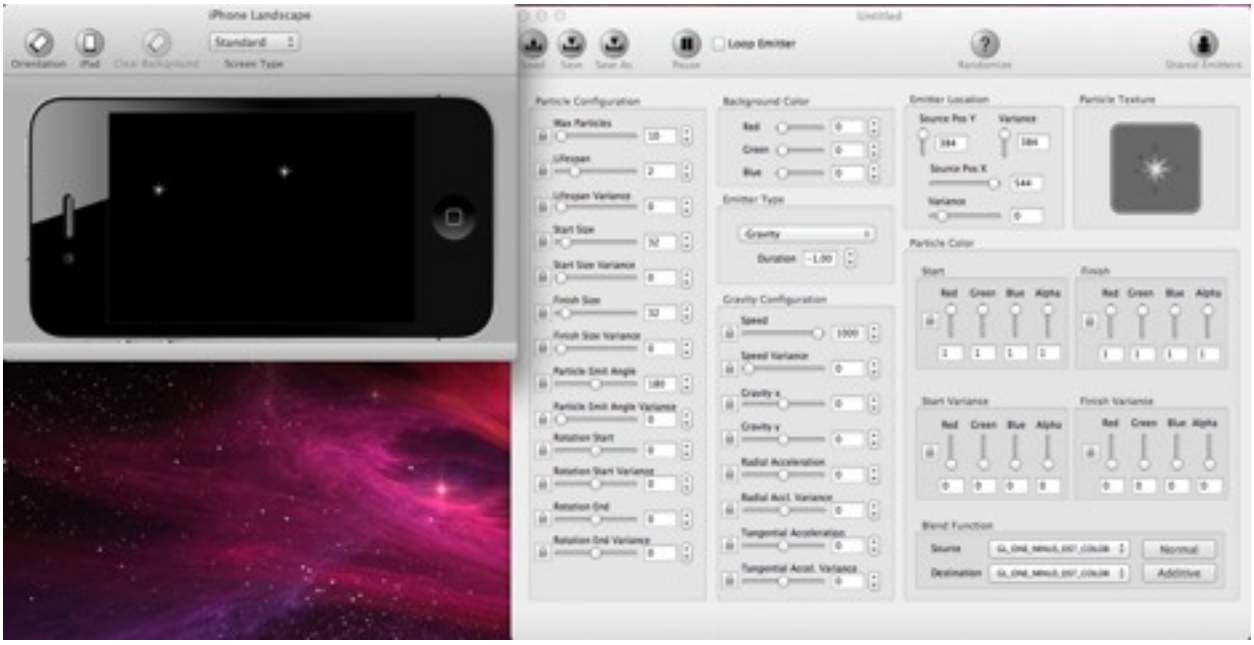
Aquí el que faig és declarar aquests objectes a "ActionLayer.h" i els implemento a "ActionLayer.mm", els situo de forma centrada a la pantalla.

Després d'implementar algunes de les funcionalitats que explicaré a continuació vaig afegir una tercera *label* que llegia "Jugar", vaig afegir un mètode per que les lletres apareguessin fem un zoom desde 0 al seu tamany original

i un altre per que al presionar sobre la etiqueta desapareguessin les etiquetes amb un efecte.

Tot això ho vaig implementar dins la funció - (void)setupTitle que es pot trobar, així com pràcticament totes les funcions que mencionarem al fitxer "ActionLayer.mm" , degut a la seva llargària no l'he inclòs als annexos, consultar al codi font.

4.1.2. Desplaçant estrelles horitzontalment



Ja que, com he comentat abans, aprofitarem la mateixa *Layer* tant per mostrar el títol com per mostrar l'acció del joc, vaig fer ús del *Particle Designer* per generar unes partícules que es veuen com estrelles desplaçant-se d'una banda a l'altre de la pantalla.

Vaig crear-la en 3 tamanys diferents per donar una sensació de que estem avançant per l'espai.

La flexibilitat que ens ofereix Objective-C aquí de poder emmagatzemar qualsevol mena de dada en una estructura d'Array genèrica anomenada *NSArray* junt amb l'ús de la funció de Cocos2D *CCParticleSystemQuad* que ens permet agafar les dades dels fitxers *plist* que hem importat a un *NSArray* i configurar sota quins paràmetres els tindrà que mostrar.

Hi havia mecanismes molt més senzills per desenvolupar això:

- Podríem haver creat un fons de pantalla (a partir d'ara *background)* amb moltes estrelles i anar desplaçant-nos a través d'ell, però aquest hauria de ser molt gran perque ens durés tota la partida o que poguèssim reproduir-

lo en bucle sense que es notès el tall, però això tindria un cost elevat en el disseny del mateix i per això la vaig descartar.

- Podríem crear un *sprite* per cada estrella mostrada d'un *Array* de unes poques estrelles diferents. Això ens estalviaria memòria d'emmagatzemar les textures però alhora faria que se'n ressentissin els fotogrames per segon de la nostre aplicació, doncs mantenir molts *sprites* en pantalla alhora es força costós.

- Podem usar sistemes de partícules els quals estan força optimitzats, es generen fàcilment amb l'aplicació *Particle Designer*, així que ha estat la opció escollida.

4.1.3. Afegir la nau protagonista

Per afegir els nostres arxius d'imatges com a *sprites* al joc fem ús de un mètode que hem declarat a "ActionLayer.mm" que s'anomena - (void)setupBatchNode en la qual senzillament instanciem un objecte del tipus *CCSpriteBatchNode* el qual usarem per desar el nostre spritesheet composat d'un arxiu *pvr.ccz* i un arxiu *plist.*

A continuació creem el mètode - (void)spawnShip el qual ens mostrarà el *sprite* que designem per la nau al centre del eix X a l'esquerra de l'eix Y. I cridem el mètode desde "*playTapped:*" per a que tal i com el jugador comenci la partida la nau aparegui.

4.1.4. Animant la nau protagonista

Per donar una sensació de que la nostre nau realment s'està propulsant a través del espai que hem creat, he disposat 2 *sprites* per a la nostre nau protagonista.

El que farem és afegir al mètode *spawnShip* una seqüència d'accions amb un objecte *CCAnimation* que és executat per un *CCAnimate* el qual l'únic que fa es canviar la imatge mostrada d'una a l'altre amb un retràs que he definit de 0.2 segons. Com aquesta acció es produeix tan sols una vegada l'he envolcallat amb un *CCRepeatForever* que precisament ens permet que això es repeteixi durant el transcurs de tota la partida.

4.2. Desplaçant la nau amb l'acceleròmetre

Com el joc no tindria cap gràcia si la nau no es pogués desplaçar he disposat un mètode que s'encarregui de rebre dades del acceleròmetre que aprofitarem per fer moure la nostre nau per l'eix Y. D'aquesta manera acomplim un dels altres objectius que ens varem fixar de dotar el joc d'un sistema de control acord amb les funcionalitats que disposa l'iPhone.

El mètode **accelerometer:didAccelerate** defineix una estructura de dades en la que emmagatzemar en temps real les dades que ens serveix l'acceleròmetre.

Com seria massa costós i visualment inviable actualitzar la posició de la nau exactament amb cada petita variació de les dades rebudes del acceleròmetre es defineix la variable d'instància "_shipPointsPerSecY" de la que ens servirem per guardar una referència de quan hauriem de moure'ns a cada fotograma (a partir d'ara *frame*).

Actualitzant la posició a cada frame s'aconsegueix que el desplaçament de la nau en l'eix Y es mostri de forma suau i contínua, tal i com s'esperava. Per a que la actualització de la posició de la nau es produeixi realment a cada *frame* hem de actualitzar la component Y de *_ship.position* sumant-li la multiplicació de *ShipPointsPerSecY* per una variable delta de temps (a partir d'ara *delta time* o *dt*).

Finalment fem que el nostre mètode *update* sigui cridat a cada *frame* i ja tenim la nau movent-se en relació a l'inclinació que apliquem al terminal.

4.3. Fons de pantalla animat amb Parallax Scrolling

Per donar sensació de profunditat he fet ús d'una tècnica que ve sent utilitzada en els videojocs de dues dimensions desde les èpoques de les consoles de 8 i 16 bits (allà per la dècada dels anys 90).

Aquesta tècnica s'anomena *Parallax Scrolling* i consisteix en composar els nostres fons de pantalla (a partir d'ara *background(s)*) de una col·lecció d'elements independents els quals desplaçarem de dreta a esquerra de la pantalla a diferent velocitat. És així com s'aconsegueix que el joc doni sensació de profunditat doncs els elements que es desplacin més depressa semblaran ser més propers al jugador que els elements que es desplacen més lent.

Per implementar això Cocos2D ens disposa *CCParallaxNode* que és l'estructura de dades on guardarem les imatges que utilitzarem per construir el nostre *background* i que aporta una sèrie de funcions que ens permetran mostrar aquests nodes amb els efectes desitjats per aconseguir aquest efecte.

He definit *setupBackground* i en tan sols 3 passes ja podre'm utilitzar aquesta estructura de dades:

Creem un *CCParallaxNode* i l'afegim a la nostre *layer.*

Agefim els elements que volem desplaçar al *CCParallaxNode* utiltizant *add:Child:parallaxRatio:positionOffset*.

Movem el *CCParallaxNode* per desplaçar el *background*. Ell desplaçarà cada node fill més o menys depressa depenent en el que haguem configurat a *parallaxRatio.*

El problema és que un cop els objectes desapareixen de la pantalla no tornen a aparèixer per defecte. Per donar el desitjat efecte de continuitat i que cada cop que els objectes desapareguin de la pantalla se'n restauri la posició he creat un mètode auxiliar "CCParallaxNodeExtras.m" (**annexe C**).

Ara ja puc fer un reinici de les posicions dels nodes fills del *CCParallaxNode* a *setupBackground*. He definit també *updateBackground* per tal de realitzar aquestes comprovacions a cada *frame*.

4.4. Implementant asteroides, làsers i detecció de col·lisions bàsica

Empleant les mateixes estructures de dades que quan afegia la nau protagonista al apartat 4.1.3. he creat el làser que dispara la nau i asteroides que es generen en un punt aleatori del eix Y a la dreta del eix X, es desplacen a velocitat també aleatòria de dreta a esquerra de la pantalla i poden ésser generats aleatòriament en 3 mides predefinides.

Vaig adonar-me que sovint quan hi havia molts elements en pantalla els frames per segon (a partir d'ara *fps*) se'n ressentien. Investigant he descobert que generar i destruir constantment *sprites* és una operació costosa en memòria per al iPhone que disposa només de 512Mb. Vaig descobrir un mecanisme que estalviava memòria i consisteix en l'addició d'una col·lecció d'elements en un *SpriteArray* i reutilitzar-los tan bon punt desapareguessin de la pantalla.

Aquesta és una estructura de dades que he utilitzat en tots els *sprites* del joc exceptuant la nau protagonista per raons obvies. Per tant l'he implementat en un altre fitxer "SpriteArray.mm" (**annexe D**).

Detecció de col·lisions bàsica

Inicialment vaig fer una detecció de col·lisions molt bàsica que consistia en utilitzar dues propietats dels *sprites* en Cocos2D. La propietat *boundingBox* i la propietat *visible*. Senzillament quan el *boundingBox*, que és la caixa que representa l'*sprite* coincidia amb el del làser establia la propietat *visible* d'ambdós elements a *NO*.

De seguida vaig detectar una mancança considerable d'aquest mecanisme, el *boundingbox* dels *sprite* no correspon amb la representació gràfica dels mateixos i, per tant, succeia que, de vegades, sense que visualment haguèssim apreciat com el làser impactava el asteroide la col·lisió es produís. Ho vaig solventar empleant la variant de *Cocos2D* anomenada *Box2D*. En el següent apartat explicaré en què consistia la detecció de col·lisions 'millorada'.

4.5. Detecció de col·lisions acurada

Donat que necessitava una detecció de col·lisions més precisa vaig investigar quins mecanismes em podien servir i un que es fa servir molt quan es necessiten físiques acurades en *Cocos2D* és la seva variant *Box2D* que ens aporta unes estructures de dades i mecanismes que ens permetra'n precisar amb detall la forma dels equivalents als *boundingBox* en *Box2D* els *body*.

A continuació veure'm com he afrontat e implementat aquest mecanisme en el meu projecte.

4.5.1. Ús de Box2D i Physics Editor

He triat el *Box2D* perquè disposa d'una gran biblioteca de definicions polinòmiques i de detecció de col·lisions que podem utilitzar.

El primer inconvenient que vaig trobar és que *Cocos2D* treballa amb punts i *Box2D* amb metres. Per tant vaig establir al fitxer "Common.h" un *PTM_ratio*(*PointToMeter*) que establís la relació de aproximadament 100 píxels - 1 metre.

Als mètodes *setupWorld i updateBox2D* he declarat les estructures de dades i mètodes necessaris per treballar amb *Box2D* i que actualitzi el seu estat cada *frame* respectivament els paràmetres del *Box2D*.

La segona dificultat que se'm presentava era la de definir els *body*s de *Box2D* amb la forma exacta de la representació gràfica dels elements de nau protagonista, nau enemiga, asteorides, làsers i el powerup.

Donada la complexitat de definir formalment de forma matemàtica els cossos que representen els *sprites* empleats, vaig decidir adquirir la llicència del *Physics Editor* que ademés de permetre'm fer aquesta definició de foma gràfica i exportar-la a un fitxer *plist*, em permetia establir categories de cossos i la interacció que aquests tenien entre ells.

Així doncs com els he mostrat en la captura de pantalla d'exemple del *Physics Editor* del capítol anterior, vaig tractar d'igual manera tots els arxius d'imatge esmentats abans i vaig definir les següents categories: *ship, ship_laser, enemy* i *powerup*. I vaig definir una màscara que determinés, per a cada categoria, els elements de quina altre categoria podien col·lisionar amb ella.

4.5.2. Adaptant el projecte per suportar Box2D

Un cop definides les geometries que volem emprar als cossos de *Box2D* associaré cada cos particular de *Box2D* al seu corresponent *sprite*. Llavors a cada *frame* mourem quan el *sprite* es desplaci situarem el cos de *Box2D* a la mateixa posició.

Per implementar aquesta funcionalitat, junt amb la que veurem en el següent apartat, vaig crear una subclasse de *CCSprite* anomenada *GameObject* (el fitxer "GameObject.mm" serà l'**annexe E**).

Obviament vaig haver de canviar de tipologia tots els objectes *CCSprite* per als que volia implementar la detecció de col·lisions per la de *GameObject*. També vaig haver d'adaptar la implementació del fitxer *SpriteArray* per tal de treballar amb els *GameObjects.*

Aleshores vaig utilitzar *ShapeCache* per guardar els paràmetres exportats amb el *PhysicsEditor* i vaig establir una sèrie de paràmetres que veure'm en el apartat següent.

4.5.3. Detectant les col·lisions

Si ens fixem en la implementació de *GameObject* podem veure com aquesta classe no només té mètodes per gestionar la lògica implícita dels cossos del *Box2D* sino que també defineix les mètodes: *setNodeInvisible, revive, dead, takeHit* i *destroy*.

Aquests mètodes van més enllà de la simple detecció de col·lisions i ens permeten descriure la lògica que ens servirà per determinar les conseqüències de les col·lisions.

Per fer ús de les categories que hem enmascarat amb el *Physics Editor* defineixo les constants *kCategoryShip*, *kCategoryShipLaser*, *kCategoryEnemy* i *kCategoryPowerup*. Això és perque el nom que hem definit per les categories en el *Physics Editor* no s'exporta al *plist* sino que es referencien com a *0x1, 0x2, 0x4* i *0x8* respectivament.

Com hem definit una quantitat de salut màxima *maxHp* per a cada element, hem assignat als asteroides una quantitat de vida, més gran per als asteroides de major tamany.

A continuació fem ús de la funció *SimpleContactListener* que ens proporciona *Box2D* per cridar dues funcions que hem de definir (*beginContact* i *endContact*) en les quals cridem l'event "rebre un impacte" *takeHit* per actualitzar els valors de salut dels objectes implicats, i si aquesta és veu reduïda a zero cridar els efectes pertinents i destruir l'objecte.

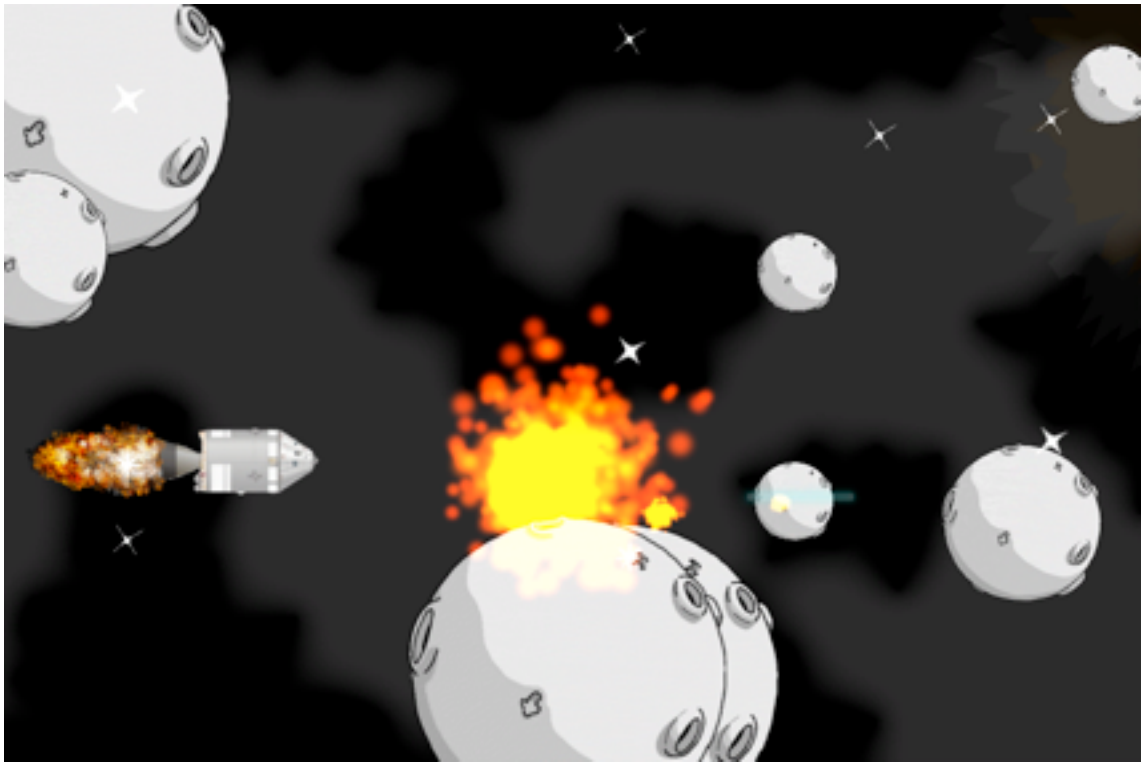
Tanmateix ens ha permés determinar l'event que determinarà la victòria de la missió o la derrota (la nau protagonista té la salut a zero). Tan si es guanya com si es perd, es mostrarà un missatge per informar al jugador i se li donarà la opció de reiniciar la partida. Que bàsicament consisteix en fer una transició que recarregui la nostre *Layer* de manera que tornem inmediatament a la pantalla de títol.

Finalment a *setupHealthBar* he definit un mètode que ens mostra i amaga una barra de vida durant dos segons quan la nau rep mal. La idea és de situar una barra de fons negre i a sobre una de verda que farem decrementar quan es produeix l'impacte i mostrarà una porció equivalent a la quantitat de vida restant de la nau.

4.6. Afegint efectes de partícules

En aquesta secció explicaré l'addició d'efectes especials creats amb el *Particle Designer* per als events de "impacte", "mort" i "powerup".

4.6.1. Explosions



En comptes de crear una explosió cada cop que es produeixi un impacte, ja que seria molt costós, farem ús de la mateixa tècnica empleada per mostrar els *sprites*. Així doncs hem creat la classe *ParticleSystemArray* i l'hem configurat de la mateixa manera que *Sprite* array però inicialitzant un altre tipus de dades.

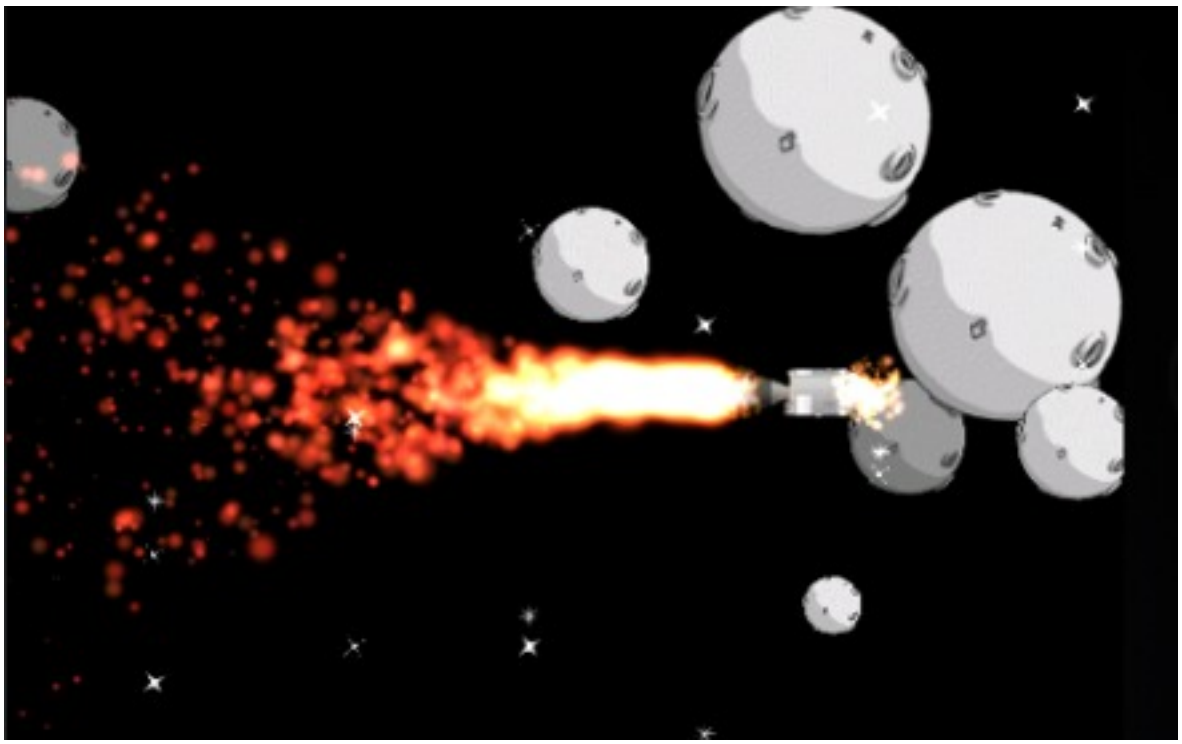
En aquest cas utilitzarem com ja he explicat en la secció 4.1.2. l'objecte *CCParticleSystemQuad* el qual ens permetrè utilitzar la explosió que he creat amb el *Particle Designer* i importat desde el seu corresponent *plist*.

Aleshores vaig associar la generació de la explosió al event que s'executa quan un enemic mor o quan la nostre nau mor. També vaig assignar aquest

efecte amb una escala més petita quan es produïa un impacte que no comportava la mort de cap element.

Per donar un efecte més espectacular cada cop que es produeix una mort, he definit el mètode *shakeScreen* que mou la pantalla 5 pixels amunt i 5 pixels avall per aconseguir l'efecte de sacsejada.

4.6.2. Powerup



He creat un efecte per quan la nau col·lisioni amb el powerup. L'objectiu és que l'efecte aquest es mostri enganxat a la nostre nau per tant vaig crear el mètode *updateBoostEffects* que senzillament iguala la posició de la particula que afegeixo a la posició de la nau. Al igual que en els anteriors mètodes d'update aquest serà cridat a cada frame pel mètode genèric d'*update* per tant la posició s'actualitzarà tan bon punt la de la nau es vegi afectada.

Ademés de crear aquest vistòs efecte volem que aquest extra de potència adquirit per la nau ens doni la sensació de que aquesta rep una empenta endavant, per tant, a *beginContact* he establert que la posició de la nau en

l'eix X al agafar el *powerup* s'incrementi fins a un 60% del tamany de la pantalla. Això la situarà enmig de la trajectòria de les naus enemigues.

Per a que aquesta bonificació ens suposi realment una ajuda farem que en el temps que dura l'efecte la nau protagonista sigui invencible, per tant pot aprofitar-se per col·lisionar i fer desaparèixer tants enemics com es pugui.

4.7. Gestor de nivells

Un dels principals objectius que em vaig marcar per aquest joc és fer-lo modificable sense haver de incidir directament en el codi font.

Per aquest motiu he afegit un mecanisme de gestió de nivells que em permetrà carregar alguns paràmetres desde un fitxer de propietats que em defineixin el comportament de l'aplicació durant una fase determinada.

En aquest fitxer extern definirem concretament una sèrie de paràmetres booleans que determinaran si s'han de generar asteroides, *powerups* o nau enemigues, la peridicitat i velocitat de desplaçament amb que generem els asteroides, la freqüència amb la que apareixen els *powerups.*

Així com preteníem podrem controlar en tot moment el comportament general del joc ajustant tan sols aquest fitxer *"Levels.plist"* (**annexe F**).

Per tal de poder extraure'n les dades i que siguin les dades proporcionades per *levels.plist* les que controlin l'aplicació he hagut de fer les següents modificacions:

* Creació d'una classe auxiliar "LevelsManager" (el fitxer d'implementació de LevelsManager serà l'**annexe G**).

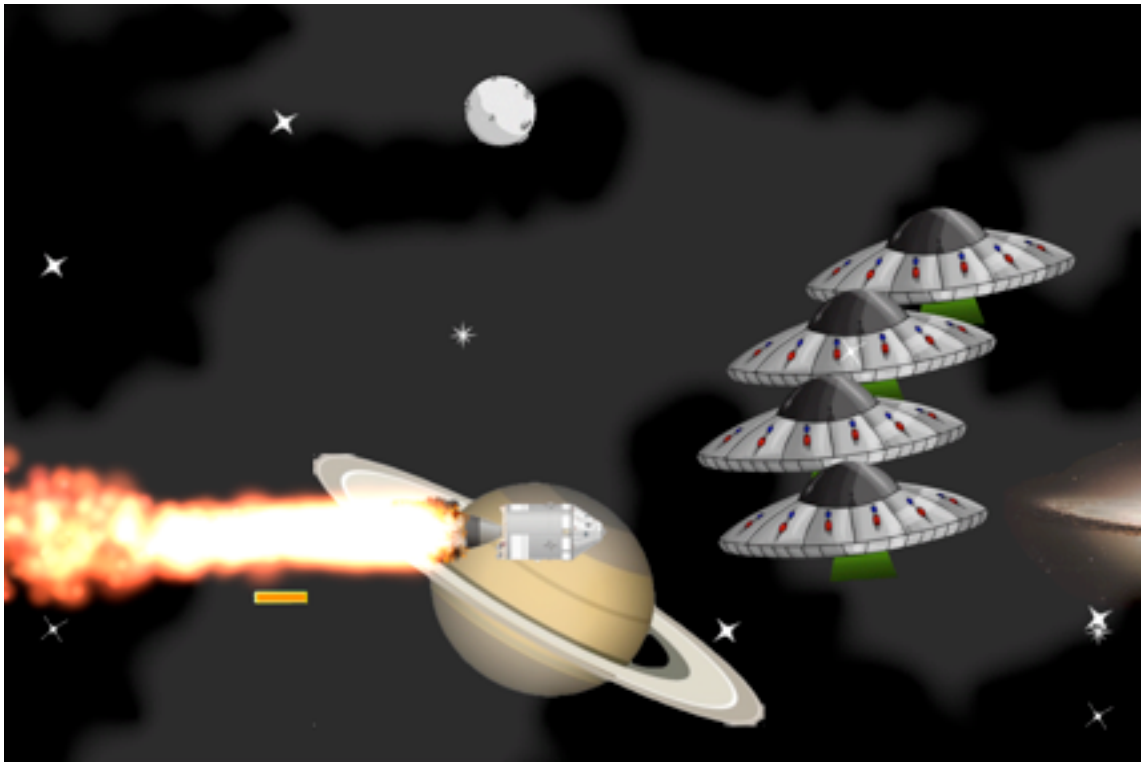
* Canviar els paràmetres establerts a *spawnAsteroids* per agafar els valors proporcionats per *LevelManager*.

* Creació dels mètodes *updateLevel* i *newStageStarted* per portal el control del estat de la fase i encetar-ne una de nova.

Finalment he creat el mètode *doLevelIntro* que em permetrà definir en el fitxer *Levels.plist* una etiqueta *LText* que podré mostrar en la fase que determinem. Això ho faré servir per mostrar interfases en la que es presenti

l’enumeració i títol de la pantalla i així podré definir fases diferenciades assolint un dels objectius fixats.

4.8. Trajectòria de les naus enemigues

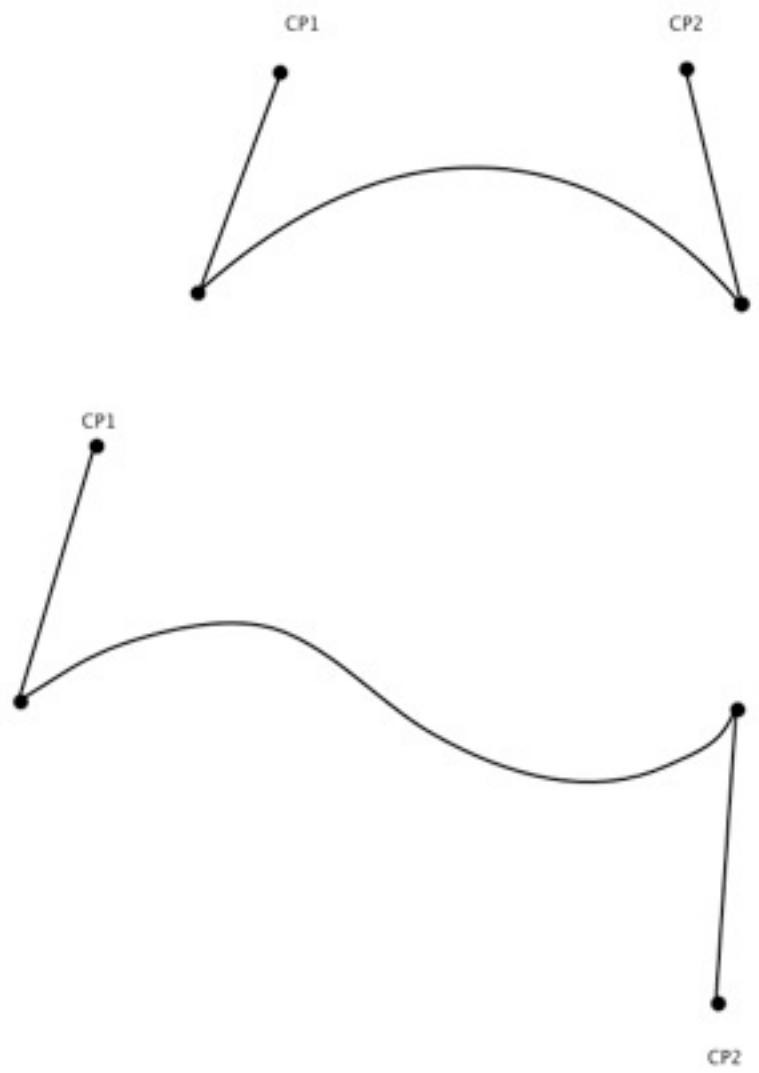


Finalment per donar-li una mica més de varietat i assolir un dels objectius que em vaig proposar, com és el de crear diferents tipus d'enemics que suposi un repte diferent a l'hora d'enfrontar-los he creat unes naus enemigues que seguiran un patró de moviment molt diferent dels asteroides i ademés ens dispararan els seus làsers.

Com ja he repassat el mecanisme per crearme un *Array* de *GameObjects* em centraré en explicar en com s'ha realitzat el patró de moviment de les naus enemigues.

Per a determinar patrons de moviment podem fer ús d'una funció de *Cocos2D* que s'anomena *ccBezierConfig.* Com veurem a *updateAlienSwarm* cada onada d'alienígenes serà d'un nombre aleatori de 1 a 20 integrants. I per definir la trajectòria per la que es mouran hem definit tan sols un punt de començament, un de final i dos punts de control.

En la següent figura podem veure en què consisteix aquesta tècnica:



Conceptualment és com si els punts de control estiressin de la corba fent que es torni convexa en relació a aquella banda.

5. Proves realitzades

Per la singularitat de l'aplicació per tal de poder avançar s'ha anat provant individualment cada nova funcionalitat tal i com era implementada. Podent així localitzar els errors i anar guardant versions vàlides del projecte fins un cert punt.

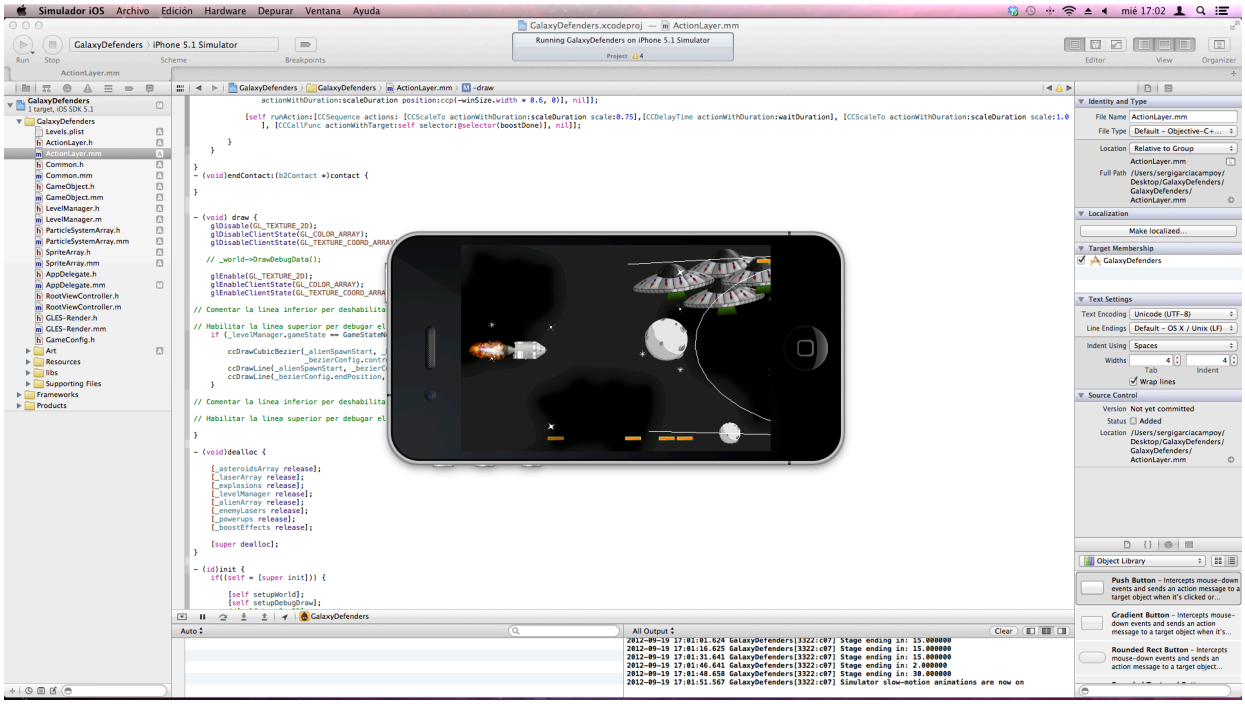
També s'han realitzat proves de rendiment de l'aplicació medint-ne els *fps*. Ja que qualsevol decrement per sota de 24*fps* la vista humana no ho apreciaria com a un moviment continu sino com a la successió de fotogrames que és en realitat. He de dir que el resultat és satisfactori i en tot moment es manté estable a 60*fps* exceptuant els moments que es mostra l'efecte del *powerup* que baixa fins a 30*fps* però es manté estable en aquella xifra sense sufrir més baixades.

Finalment per comprovar la correcta generació dels cossos de *Box2D* i que la trajectòria de les naus enemigues era la desitjada, s'han inclòs al codi font un parell de mètodes per dibuixar representacions dels cossos *Box2D* i de la trajectòria dibuixada pel *"Bezier Path"*.

En les següents dues captures es pot apreciar el que vull dir:

Prova dels cossos de *Box2D:*

Prova de les trajectòries de les naus enemigues:



6. Conclusions i millores

La realització d'aquest projecte considero que ha estat satisfactòria tant en quant a resultats obtinguts com en satisfacció personal.

S'han assolit els principals i més crítics objectius que es van fixar al inici d'aquest camí i, el més important per mi, he après mecanismes de treball que sovint es prenen quan es produeix un videojoc.

El més positiu que extrec de l'el·laboració d'aquest projecte és el fet que arrel d'això em veig capacitat per afrontar reptes de similar envergadura i progressivament anar incrementant el meu coneixement en aquest camp, doncs amb l'execució d'aquest projecte he assolit uns fonaments de la programació de videojocs que puc utilitzar en multitud de jocs de diferent mena.

Quan vaig escollir aquest projecte lliurement era això el què pretenia. Aprofitar l'ocasió de fer el meu projecte final de carrera per adquirir una nova especialització en el camp de la informàtica, i més enllà dels resultats, crec que això s'ha aconseguit doncs la inqüietud que tinc actualment per aquest tema, estic segur que em portarà a seguir indagant pel meu compte en les meves estones lliures i, qui sap, si algun dia se'm presentarà la oportunitat de treballar en aquesta indústria que tan m'agrada i la conseqüent sensació de realització quan algú treballa en lo que realment el motiva.

El resultat final del projecte ha estat una aplicació estable que fa es comporta en tot moment com l'esperat.

L'usuari pot començar una partida al engegar-la, i amb un senzillíssim mecanisme de control utilitzant l'acceleròmetre i la pantalla tàctil del iPhone, en pocs segons, pot assolir la mecànica del joc.

Com introduïa al començament d'aquesta valoració els principals objectius s'han assolit, l'únic que no he implementat és un sistema de puntuacions que fomenti la competitivitat, per tant que es pugui publicar els resultats en les xarxes socials o reptar a algun amic.

El motiu ha estat un error meu en la estimació de la complexitat i durada del projecte que em va fer preveure que podria realitzar també aquesta funcionalitat i què finalment per circunstàncies personals unida, a la citada, major complexitat trobada del que m'esperava, m'han portat a no implementar aquesta funcionalitat.

Dit això aprofito per nombrar les millores que se m'ocorren per seguir fent del meu joc un competidor digne de a la *AppStore*:

* Afegir un sistema de puntuacions.

* Afegir un sistema d'obtenció de crèdits per objectius.

* Afegir una botiga virtual on poder comprar beneficis per la nostre nau com podrien ser noves armes, escuts, salut, una millor resposta de la nau i un llarg etcètera.

* Afegir diferents tipus de *Powerup* i que, aquests, segueixin patrons de moviment més complexos per incrementar el repte de recopil·lar-los.

* Afegir la citada conectivitat amb les principals xarxes socials i que puguis publicar puntuacions i reptar amics a superar-te. El que podria ser una forma de promocionar el joc alhora.

* Afegir un botó per pausar la partida. I un menú de pausa que et permeti accedir a la botiga i les diferents opcions del joc.

* Afegir fases d'enemics finals de patrons d'atac i moviment més el·laborats.

La llista és interminable, aquest projecte tan sols és el punt de partida.

Tan sols concluir esmentant que no m'ha estat possible seguir la planificació temporal prevista per circumstàncies personals, com he deixat entreveure anteriorment. Havia planificat el projecte per ésser entregat al Juny perquè tenia un segon semestre amb poques assignatures i havia planificat dedicar-m'hi.

La qüestió és que vaig trobar feina i porto desde el Febrer treballant per això he hagut de posposar les fases d'Anàli, disseny e implementació, proves i l'elaboració d'aquest document a les setmanes de vacances que he disposat entre meitats d'Agost i meitats de Setembre.

En canvi, si que s'han acomplert aproximadament les duracions previstes, havent resultat més llarga del prevista la fase de formació, i més curta la d'implementació.

7. Bibliografia

Lloc web de Cocos2D per iPhone:

http://www.cocos2d-iphone.org/wiki/doku.php/

Lloc web de Box2D:

http://box2d.org/

Lloc web de *developer* d'Apple:

https://developer.apple.com/

Lloc web del TexturePacker Pro:

http://www.codeandweb.com/texturepacker

Lloc web del Physics Editor:

http://www.codeandweb.com/physicseditor

Lloc web del Particle Designer:

http://particledesigner.71squared.com/

8. Annexos

Annex A: Quadre de tasques del projete:

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Projecte: Desenvolupament d'un Space Shooter per iOS	74,87 días	lun 14/11/11	mar 28/02/12		
Inici del projecte: assignació i matriculació del projecte	2 horas	lun 14/11/11	lun 14/11/11		Cap de Projecte;Director de Projecte[10%]
Planificació	4,25 días	lun 14/11/11	vie 18/11/11		
Estudi de viabilitat	12 horas	lun 14/11/11	mar 15/11/11		Cap de Projecte
Aprovació Estudi Viabilitat (milestone)	1 hora	mar 15/11/11	mar 15/11/11	4	Cap de Projecte;Director de Projecte[10%]
Pla del projecte	20 horas	mar 15/11/11	vie 18/11/11	5	Cap de Projecte
Aprovació Pla del Projecte (milestone)	1 hora	vie 18/11/11	vie 18/11/11	6	Cap de Projecte;Director de Projecte[10%]
Formació	38,13 días	vie 18/11/11	jue 12/01/12		
Formació en ISDK (Xcode)	156 horas	vie 18/11/11	jue 15/12/11	7	Analista,Programador
Formació en Framework 2D (COCOS2D for iPhone)	156 horas	jue 15/12/11	jue 12/01/12	9	Analista,Programador
Formació completada (milestone)	1 hora	jue 12/01/12	jue 12/01/12	10;9	Cap de Projecte;Director de Projecte[10%]
Anàlisi de l'aplicació	3,38 días	jue 12/01/12	mar 17/01/12		
Anàlisi de requisits (casses d'ús)	16 horas	jue 12/01/12	lun 16/01/12	11	Analista
Anàlisi de la seguretat i legalitat	6 horas	lun 16/01/12	mar 17/01/12	13	Analista
Documentació de l'anàlisi	4 horas	mar 17/01/12	mar 17/01/12	14	Analista
Aprovació de l'anàlisi (milestone)	1 hora	mar 17/01/12	mar 17/01/12	15	Cap de Projecte;Analista[50%];Director de Projecte[10%]
Disseny de l'aplicació	3,25 días	mar 17/01/12	vie 20/01/12		
Disseny de la interfície de menú	2 horas	mar 17/01/12	mar 17/01/12	16	Analista[50%];Programador[25%]
Disseny estructural de joc/aplicació	16 horas	mar 17/01/12	jue 19/01/12	16	Analista[50%];Programador[25%]
Disseny de les proves de test	5 horas	jue 19/01/12	vie 20/01/12	18;19	Analista[50%];Programador[40%]
Documentació del disseny	4 horas	vie 20/01/12	vie 20/01/12	20	Analista
Aprovació del disseny (milestone)	1 hora	vie 20/01/12	vie 20/01/12	21	Cap de Projecte;Analista[50%];Director de Projecte[10%]
Desenvolupament de l'aplicació	15,25 días	lun 23/01/12	lun 13/02/12		
Preparació entorn de desenvolupament	2 horas	lun 23/01/12	lun 23/01/12	22	Programador
Desenvolupament de l'interfície de menú	10 horas	lun 23/01/12	mar 24/01/12	24	Programador
Desenvolupament de la lògica del joc	120 horas	lun 23/01/12	lun 13/02/12	24	Programador
Desenvolupament dels diferents nivells	40 horas	lun 23/01/12	lun 30/01/12	24	Programador
Test i proves	4,37 días	lun 13/02/12	vie 17/02/12		
Proves unitàries	12 horas	lun 13/02/12	mar 14/02/12	25;26;27	Programador
Proves d'integració	12 horas	mar 14/02/12	jue 16/02/12	29	Programador
Proves d'estrès	6 horas	jue 16/02/12	jue 16/02/12	30	Programador
Documentació de desenvolupament de test	4 horas	vie 17/02/12	vie 17/02/12	31	Programador
Aprovació del desenvolupament de proves (milestone)	8,94 horas	vie 17/02/12	vie 17/02/12	32	Cap de Projecte;Programador[50%];Director de Projecte[10%]
Implantació	1,88 días	vie 17/02/12	mar 21/02/12		
Instal·lació	3 horas	vie 17/02/12	vie 17/02/12	33	Analista[75%];Programador[25%]
Proves reals	12 horas	vie 17/02/12	mar 21/02/12	35	Analista[75%];Programador[25%]
Generació de documents (memòria del projecte)	36 horas	mar 21/02/12	lun 27/02/12	36	Cap de Projecte
Tancament del projecte	1 hora	lun 27/02/12	mar 28/02/12	37	Cap de Projecte;Director de Projecte[10%]
Defensa del projecte	6 horas	mar 28/02/12	mar 28/02/12	38	Cap de Projecte

Annex B: Calendari del projecte:

Annex C: ParallaxNode-extras

```
//
//  CCParallaxNode-Extras.m
//  SpaceGame
//
//  Created by Sergi Garcia Campoy on 11/09/12.
//
//

#import "CCParallaxNode-Extras.h"

@implementation CCParallaxNode(Extras)
@class CGPointObject;

-(void) incrementOffset:(CGPoint)offset forChild:(CCNode*)node
{
    for( unsigned int i=0;i < parallaxArray_->num;i++) {
            CGPointObject *point = parallaxArray_->arr[i];
            if( [[point child] isEqual:node] ) {
                  [point setOffset:ccpAdd([point offset], offset)];
                  break;
            }
    }
}

@end
```

Annex D: SpriteArray

```
//
//  SpriteArray.m
//  GalaxyDefenders
//
//  Created by Sergi Garcia Campoy on 12/09/12.
//
//

#import "SpriteArray.h"

@implementation SpriteArray
@synthesize array = _array;

- (id)initWithCapacity:(int)capacity spriteFrameName:(NSString *)spri-
teFrameName
             batchNode:(CCSpriteBatchNode *)batchNode world:(b2World
*)world
             shapeName:(NSString *)shapeName maxHp:(int)maxHp
healthBarType:(HealthBarType)healthBarType {

    if ((self = [super init])) {

        _array = [[CCArray alloc] initWithCapacity:capacity];

        for(int i = 0; i < capacity; ++i) {
            GameObject *sprite = [[[GameObject alloc]
initWithSpriteFrameName:spriteFrameName
                                            world:world shapeName:shapeName
maxHp:maxHp healthBarType:healthBarType] autorelease];
            sprite.visible = NO;
            [batchNode addChild:sprite];
            [_array addObject:sprite];
        }

    }
    return self;

}

- (id)nextSprite {
    id retval = [_array objectAtIndex:_nextItem];
    _nextItem++;
    if (_nextItem >= _array.count) _nextItem = 0;
    return retval;
}
```

```
- (void)dealloc {
    [_array release];
    _array = nil;
    [super dealloc];
}

@end
```

Annex E: GameObject

```
//
//  GameObject.m
//  GalaxyDefenders
//
//  Created by Sergi Garcia Campoy on 13/09/12.
//
//

#import "GameObject.h"
#import "ShapeCache.h"

@implementation GameObject
@synthesize maxHp = _maxHp;

- (void)setupHealthBar {

    if (_healthBarType == HealthBarTypeNone) return;

    _healthBarBg = [CCSprite spriteWithSpriteFrameName:
                    @"healthbar_bg.png"];
    _healthBarBg.position = ccpAdd(self.position,
                                   ccp(self.contentSize.width/2,

-_healthBarBg.contentSize.height));
    [self addChild:_healthBarBg];

    NSString *progressSpriteName;
    if (_healthBarType == HealthBarTypeGreen) {
        progressSpriteName = @"healthbar_green.png";
    } else {
        progressSpriteName = @"healthbar_red.png";
    }
    _healthBarProgressFrame =
    [[[CCSpriteFrameCache sharedSpriteFrameCache]
      spriteFrameByName:progressSpriteName] retain];
    _healthBarProgress =
    _healthBarProgress =
```

```
    [CCSprite spriteWithSpriteFrameName:progressSpriteName];
    _healthBarProgress.position =
ccp(_healthBarProgress.contentSize.width/2,
_healthBarProgress.contentSize.height/2);
    _fullWidth = _healthBarProgress.textureRect.size.width;
    [_healthBarBg addChild:_healthBarProgress];

}

- (void)update:(ccTime)dt {

    if (_healthBarType == HealthBarTypeNone) return;

    float POINTS_PER_SEC = 50;

    float percentage = _hp / _maxHp;
    percentage = MIN(percentage, 1.0);
    percentage = MAX(percentage, 0);
    float desiredWidth = _fullWidth *percentage;

    if (desiredWidth < _displayedWidth) {
        _displayedWidth = MAX(desiredWidth,
                              _displayedWidth - POINTS_PER_SEC*dt);
    } else {
        _displayedWidth = MIN(desiredWidth,
                              _displayedWidth + POINTS_PER_SEC*dt);
    }

    CGRect oldTextureRect = _healthBarProgress.textureRect;
    CGRect newTextureRect = CGRectMake(
                                       oldTextureRect.origin.x,
oldTextureRect.origin.y,
                                       _displayedWidth,
oldTextureRect.size.height);

    CGRect rectInPixels =
    CC_RECT_POINTS_TO_PIXELS( newTextureRect );
    [_healthBarProgress setTextureRectInPixels:rectInPixels

rotated:_healthBarProgressFrame.rotated
                                  untrimmedSize:
     _healthBarProgressFrame.originalSizeInPixels];

    _healthBarProgress.position = ccp(_displayedWidth/2,

_healthBarProgress.contentSize.height/2);
    if (desiredWidth != _displayedWidth) {
        _healthBarBg.visible = TRUE;
        [_healthBarBg stopAllActions];
```

```
        [_healthBarBg runAction:
         [CCSequence actions:
          [CCFadeTo actionWithDuration:0.25 opacity:255],
          [CCDelayTime actionWithDuration:2.0],
          [CCFadeTo actionWithDuration:0.25 opacity:0],
          [CCCallFunc actionWithTarget:self
selector:@selector(fadeOutDone)], nil]];
        [_healthBarProgress stopAllActions];
        [_healthBarProgress runAction:
         [CCSequence actions:
          [CCFadeTo actionWithDuration:0.25 opacity:255],
          [CCDelayTime actionWithDuration:2.0],
          [CCFadeTo actionWithDuration:0.25 opacity:0], nil]];
    }

}

- (void)fadeOutDone {
    _healthBarBg.visible = FALSE;
}

- (id)initWithSpriteFrameName:(NSString *)spriteFrameName
world:(b2World *)world shapeName:(NSString *)shapeName
maxHp:(float)maxHp healthBarType:(HealthBarType)healthBarType {

    if ((self = [super initWithSpriteFrameName:spriteFrameName])) {
        _hp = maxHp;
        _maxHp = maxHp;
        _world = world;
        _shapeName = [shapeName retain];
    //    _healthBarType = healthBarType;
    //    [self setupHealthBar];
    //    [self scheduleUpdate];
    }
    _healthBarType = healthBarType;
    [self setupHealthBar];
    [self scheduleUpdate];
    return self;
}

- (void) destroyBody {
    if (_body != NULL) {
        _world->DestroyBody(_body);
        _body = NULL;
    }
}

- (void) createBody {
```

```
    [self destroyBody];

    b2BodyDef bodyDef;
    bodyDef.type = b2_dynamicBody;
    bodyDef.position.Set(self.position.x/PTM_RATIO,
                         self.position.y/PTM_RATIO);
    bodyDef.userData = self;
    _body = _world->CreateBody(&bodyDef);
    [[ShapeCache sharedShapeCache]
     addFixturesToBody:_body
     forShapeName:_shapeName
     scale:self.scale];
    [self setAnchorPoint:
     [[ShapeCache sharedShapeCache] anchorPointForShape:_shapeName]];

}

- (void)setNodeInvisible:(CCNode *)sender {
    sender.position = CGPointZero;
    sender.visible = NO;
    [self destroyBody];
}

- (void)revive {
    _hp = _maxHp;
    [self stopAllActions];
    self.visible = YES;
    self.opacity = 255;
    [self createBody];
    _displayedWidth = _fullWidth;
    _healthBarBg.visible = NO;
}

- (BOOL)dead {
    return _hp == 0;
}

- (void)takeHit {
    if (_hp > 0) {
        _hp--;
    }
    if (_hp == 0) {
        [self destroy];
    }
}

- (void)destroy {

    _hp = 0;
```

```
    [self stopAllActions];
    [self runAction:
     [CCSequence actions:
      [CCFadeOut actionWithDuration:0.1],
      [CCCallFuncN actionWithTarget:self
                           selector:@selector(setNodeInvisible:)],
      nil]];

}

- (void)dealloc {
        //  [_healthBarProgressFrame release];
        //  _healthBarProgressFrame = nil;
        [_shapeName release];
        _shapeName = nil;
        [super dealloc];
    }



@end
```

Annex F: Levels.plist

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">

<dict>

        <key>Levels</key>

        <array>

                <array>

                        <dict>

                                <key>Duration</key>

                                <integer>2</integer>

                                <key>SpawnLevelIntro</key>
```

```
        <true/>
        <key>LText</key>
        <string>El Cinturo d&amp;apos;Asteroides</string>
</dict>
<dict>
        <key>SpawnAsteroids</key>
        <true/>
        <key>Duration</key>
        <integer>15</integer>
        <key>AMoveDurationHigh</key>
        <integer>8</integer>
        <key>AMoveDurationLow</key>
        <integer>2</integer>
        <key>ASpawnSecsHigh</key>
        <integer>1</integer>
        <key>ASpawnSecsLow</key>
        <real>0.2</real>
</dict>
<dict>
        <key>SpawnAsteroids</key>
        <true/>
        <key>SpawnPowerups</key>
        <true/>
        <key>PSpawnSecs</key>
        <integer>15</integer>
```

```
        <key>Duration</key>
        <integer>15</integer>
        <key>AMoveDurationHigh</key>
        <integer>3</integer>
        <key>AMoveDurationLow</key>
        <integer>2</integer>
        <key>ASpawnSecsHigh</key>
        <real>0.5</real>
        <key>ASpawnSecsLow</key>
        <real>0.3</real>
    </dict>
    <dict>
        <key>SpawnAsteroids</key>
        <true/>
        <key>SpawnPowerups</key>
        <true/>
        <key>PSpawnSecs</key>
        <integer>20</integer>
        <key>Duration</key>
        <integer>15</integer>
        <key>AMoveDurationHigh</key>
        <integer>7</integer>
        <key>AMoveDurationLow</key>
        <integer>3</integer>
        <key>ASpawnSecsHigh</key>
```

```
                <real>0.4</real>
                <key>ASpawnSecsLow</key>
                <real>0.2</real>
        </dict>
</array>
<array>
        <dict>
                <key>SpawnLevelIntro</key>
                <true/>
                <key>Duration</key>
                <integer>2</integer>
                <key>LText</key>
                <string>INVASIO ENEMIGA !</string>
        </dict>
        <dict>
                <key>SpawnAlienSwarm</key>
                <true/>
                <key>Duration</key>
                <integer>30</integer>
                <key>SpawnAsteroids</key>
                <true/>
                <key>SpawnPowerups</key>
                <true/>
                <key>PSpawnSecs</key>
                <integer>20</integer>
```

<key>AMoveDurationHigh</key>

<integer>10</integer>

<key>AMoveDurationLow</key>

<integer>2</integer>

<key>ASpawnSecsHigh</key>

<integer>2</integer>

<key>ASpawnSecsLow</key>

<integer>1</integer>

</dict>

<dict>

<key>SpawnAlienSwarm</key>

<true/>

<key>Duration</key>

<integer>30</integer>

<key>SpawnAsteroids</key>

<true/>

<key>SpawPowerups</key>

<true/>

<key>PSpawnSecs</key>

<string>11</string>

<key>AMoveDurationHigh</key>

<integer>6</integer>

<key>AMoveDurationLow</key>

<integer>2</integer>

<key>ASpawnSecsHigh</key>

```
                        <real>0.7</real>
                        <key>ASpawnSecsLow</key>
                        <real>0.3</real>
                    </dict>
                </array>
            </array>
</dict>
</plist>
```

Annex G: LevelManager

```
//
//  LevelManager.m
//  GalaxyDefenders
//
//  Created by Sergi Garcia Campoy on 15/09/12.
//
//

#import "LevelManager.h"
#import <QuartzCore/QuartzCore.h>

@implementation LevelManager
@synthesize gameState = _gameState;
@synthesize curLevelIdx = _curLevelIdx;

- (id)init {
    if ((self = [super init])) {

        NSString *levelDefsFile =
        [[NSBundle mainBundle]
         pathForResource:@"Levels" ofType:@"plist"];
        _data =
        [[NSDictionary
          dictionaryWithContentsOfFile:levelDefsFile]
         retain];
        NSAssert(_data != nil,
                 @"Couldn't open Levels file");
```

```
        _levels = (NSArray *)
        [_data objectForKey:@"Levels"];
        NSAssert(_levels != nil,
                 @"Couldn't find Levels entry");

        _curLevelIdx = -1;
        _curStageIdx = -1;
        _gameState = GameStateTitle;

    }
    return self;
}

- (BOOL)hasProp:(NSString *)prop {
    NSString * retval =  (NSString *)
    [_curStage objectForKey:prop];
    return retval != nil;
}

- (NSString *)stringForProp:(NSString *)prop {
    NSString * retval =  (NSString *)
    [_curStage objectForKey:prop];
    NSAssert(retval != nil,
             @"Couldn't find prop %@", prop);
    return retval;
}

- (float)floatForProp:(NSString *)prop {
    NSNumber * retval = (NSNumber *)
    [_curStage objectForKey:prop];
    NSAssert(retval != nil,
             @"Couldn't find prop %@", prop);
    return retval.floatValue;
}

- (BOOL)boolForProp:(NSString *)prop {
    NSNumber * retval =  (NSNumber *)
    [_curStage objectForKey:prop];
    if (!retval) return FALSE;
    return [retval boolValue];
}

- (void)nextLevel {
    _curLevelIdx++;
    if (_curLevelIdx >= _levels.count) {
        _gameState = GameStateDone;
        return;
    }
    _curStages = (NSArray *)
```

```
    [_levels objectAtIndex:_curLevelIdx];
    [self nextStage];
}

- (void)nextStage {
    _curStageIdx++;
    if (_curStageIdx >= _curStages.count) {
        _curStageIdx = -1;
        [self nextLevel];
        return;
    }

    _gameState = GameStateNormal;
    _curStage = [_curStages objectAtIndex:_curStageIdx];

    _stageDuration = [self floatForProp:@"Duration"];
    _stageStart = CACurrentMediaTime();

    NSLog(@"Stage ending in: %f", _stageDuration);

}

- (BOOL)update {
    if (_gameState == GameStateTitle ||
        _gameState == GameStateDone) return FALSE;
    if (_stageDuration == -1) return FALSE;

    double curTime = CACurrentMediaTime();
    if (curTime > _stageStart + _stageDuration) {
        [self nextStage];
        return TRUE;
    }

    return FALSE;
}

- (void)dealloc {
    [_data release];
    [super dealloc];
}

@end
```