Disseny i desenvolupament d’un videojoc.
“Ew, the small alien”

Sergi Rubiralta Fontanet

Màster Universitari en Enginyeria Informàtica

Desenvolupament d'Aplicacions sobre Dispositius Mòbils

Jordi Ceballos Villach

Jordi Almirall López

Robert Clarisó Viladrosa

09/2016 - 01/2017

Agraïments:

Al meu avi, t'he perdut en aquest camí, però sempre estaràs al meu costat. No t'oblidaré mai!

A la Mireia, la meva parella, gràcies per la teva paciència, per escoltar-me, per estar sempre al meu costat i animar-me quan més ho necessito.

Als meus professors, tutors, companys, amics, familiars, etc. pels consells que m'heu donat.

Al Jordi Ceballos i al Jordi Almirall, per l'orientació, suport i confiança que m'heu donat durant totes les fases del projecte.

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Títol del treball:	*Disseny i desenvolupament d'un videojoc.* *"Ew the small alien"*
Nom de l'autor:	*Sergi Rubiralta Fontanet*
Nom del consultor/a:	*Jordi Ceballos Villach*
Nom del PRA:	*Jordi Almirall López,* Robert Clarisó Viladrosa
Data de lliurament (mm/aaaa):	*01/2017*
Titulació o programa:	Màster Universitari en Enginyeria Informàtica
Àrea del Treball Final:	Desenvolupament d'Aplicacions sobre Dispositius Mòbils
Idioma del treball:	*Català*
Paraules clau	*Videojoc, Android, Alien.*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

L'evolució constant dels dispositius mòbils està fent que cada vegada estiguin al nostre abast aparells més potents que ens permeten realitzar moltes tasques en qualsevol moment i lloc. Una d'elles és poder jugar a videojocs, ja sigui de forma local o per Internet, amb gent de tot el món.

Pel seu costat, la indústria dels videojocs, està en constant creixement. Les seves vendes s'aproximen als nivells del cinema o la música; fins i tot es creen esdeveniments on els jugadors s'enfronten entre ells i es retransmet com si d'un partit de futbol es tractés.

Aquest treball proposa els passos realitzats en el disseny i implementació d'un videojoc desenvolupat en Unity i compatible amb dispositius mòbils i tauletes Android, amb sistema operatiu 2.3 o superior.

El joc se centra en l'aparició d'un petit extraterrestre, l'Ew, prop de la terra amb la seva nau espatllada. L'Ew no té més remei que saltar de la seva nau per salvar la vida; però el descens cap a la terra no serà gens fàcil. Meteorits, satèl·lits terrestres, avions o coets són alguns dels perills que es trobarà en el seu camí i que haurà d'esquivar si vol sobreviure al perillós descens. Però això no és tot, l'Ew haurà d'obrir el paracaigudes en el moment just, si no ho fa, les

seves possibilitats d’arribar a la terra seran nul·les.

Abstract (in English, 250 words or less):

The constant evolution of mobile devices is making more powerful devices that allow us to do more work anytime, anywhere. One is play video games, either local or online, with people from all over the world.

For its part, the video game industry is constantly growing. Its sales levels approaching cinema and music; even create events where players face each other and is broadcasted as if it were a football match.

This paper proposes steps taken in the design and implementation of a video game developed in Unity for Android phones and tablets with OS 2.3 or higher.

The game starts with the appearance of a small alien called “Ew”, close to the earth with his damaged ship. Ew has no choice and have to jump from his ship to save his life; but the descent to the earth will not be easy. Meteors, satellites, planes or rockets are some of the dangers that will be in the fall and must avoid them if he wants to survive this dangerous descent. But that's not all, Ew will have to open the parachute at the right time if he want to survive, doing it at wrong time will diminish hopes or reaching the earth.

Índex

1.- Introducció. 10
1.1.- Motivació i descripció del problema. 10
1.2.- Objectius del treball. 11
1.3.- Enfocament i metodologia. 11
1.4.- Àmbit d'aplicació. 12
1.5.- Planificació del projecte. 13
1.6.- Maquinari i programari utilitzat. 15
1.7.- Sumari de materials obtinguts. 15
1.8.- Organització de la memòria. 16
2.- Història i disseny del joc. 17
2.1.- Definició de la història del joc. 17
2.2.- Disseny del Logotip. 17
2.3.- Disseny del fons de pantalla. 17
2.4.- Disseny del personatge. 19
2.5.- Disseny dels obstacles. 19
2.5.1.- Meteorits. 19
2.5.2.- Satèl·lits. 19
2.5.3.- Coets. 20
2.5.4.- Avions. 20
2.6.- Altres dissenys. 20
2.6.1.- Paracaigudes. 20
2.6.2.- Monedes. 21
2.6.3.- Núvols. 21
2.6.4.- Nau espacial. 21
2.7.- Disseny o elecció de música, tipografia i botons. 22
2.7.1.- Música. 22
2.7.2.- Tipografia. 22
2.7.3.- Botons. 22
2.8.- Sketches i prototip. 23
2.8.1. - Pantalla principal. 23
2.8.2.- Menú d'opcions o configuracions. 23
2.8.3.- Pantalla de crèdits. 24
2.8.4.- Pantalla del joc. 24
2.8.5.- Menú pausa. 25

2.8.6.- Prototip horitzontal d'alta fidelitat 25
3.- Implementació dels menús..... 27
3.1.- Menú principal..... 27
3.2.- Menú de configuracions..... 31
3.3.- Crèdits. 34
4- Implementació del joc..... 35
4.1.- Moviment del personatge i control de la càmera..... 35
4.1.1.- Moviment del personatge. 35
4.1.2.- Control de la càmera. 37
4.2.- Creació, control i configuració d'objectes enemics. 38
4.2.1.- Creació d'enemics (Enemy Spawner)..... 39
4.2.2.- Control i moviment d'enemics..... 42
4.2.3.- Control de monedes. 45
4.2.4.- Control dels núvols. 47
4.2.5.- Destroy Objects. 47
4.3.- Zona d'obertura del paracaigudes. 48
4.4.- Sistema i control de col·lisions..... 50
4.5.- Menús, texts i botons en el joc..... 51
4.6.- Game Manager..... 53
5.- Anàlisi, detecció i correcció d'errors..... 57
5.1. El personatge surt de la pantalla..... 57
5.2.- Reducció box collider enemics..... 58
6.- Funcionament de l'aplicació..... 59
7.- Valoració econòmica..... 61
8.- Conclusions i línies de futur. 63
9.- Glossari..... 65
10.- Bibliografia. 66
11.- Annex..... 67
11.1 – Compressió imatges en Unity. 67
11.2 – Creació fitxer binari (APK) en Unity..... 68
11.3 – Reducció pes APK en Unity. 69
11.4 – Selecció del nivell mínim de compilació d'Android. 70

Llista de figures

Fig. I Gràfica de versions d'API/Quota de mercat.
Fig. II Diagrama de l'Estructura de Distribució del Treball (EDT).
Fig. III Logotip de l'aplicació i escut de Arcadium.
Fig. IV Disseny del fons de menú.
Fig. V Disseny del fons del joc.
Fig. VI Esborrany i versió final de l'Ew.
Fig. VII Disseny Meteorits.
Fig. VIII Disseny satèl·lits.
Fig. IX Disseny coets.
Fig. X Disseny avions.
Fig. XI Disseny del Paracaigudes.
Fig. XII Disseny moneda en moviment.
Fig. XIII Disseny núvols.
Fig. XIV Nau espacial.
Fig. XV Mostra tipografia "Spicy Rice".
Fig. XVI Disseny Botons de “Game Art 2D”.
Fig. XVII Sketch pantalla principal.
Fig. XVIII Sketch menú opcions.
Fig. XIX Sketch crèdits.
Fig. XX Sketch pantalla del joc.
Fig. XXI Sketch menu pausa.
Fig. XXII Prototip d'alta fidelitat.
Fig. XXIII Script MenuManager.
Fig. XXIV Configuració acció botó btn_configuration.
Fig. XXV Script PathHolderScript.
Fig. XXVI Mostra funcionament PathHolder.
Fig. XXVII Script MoveOnEdithorPath.
Fig. XXVIII Menú Principal moviment.
Fig. XXIX Script PlaySmoke.
Fig. XXX Mostra funcionament fum.
Fig. XXXI Music Manager configuration.
Fig. XXXII Script MusicManager
Fig. XXXIII Configuració slider volum.
Fig. XXXIV sensitivityManager.
Fig. XXXV Menú Configuration Final.
Fig. XXXVI Menú crèdits final.
Fig. XXXVII Horitzontal Ew Movement Script.
Fig. XXXVIII Limit Bounds Ew.
Fig. XXXIX Script velocitat de caiguda prémer botó eject.
Fig. XL Error límit inferior càmera.
Fig. XLI Final moviment límit inferior càmera.
Fig. XLII Script CameraController.
Fig. XLIII Enemy Spawner i zones d'objectes.
Fig. XLIV Script EnemySpawner.
Fig. XLV Script EnemySpawner part 2.

Fig. XLVI Script enemySpawner part 3.
Fig. XLVII Configuració EnemySpawner.
Fig. XLVIII Objectes changeDirection.
Fig. XLIX Script enemyScript.
Fig. L Script RandomEnemyImage.
Fig. LI Exemple configuració coet enemic.
Fig. LII Script CoinController.
Fig. LIII Script FloatingText.
Fig. LIV Exemple FloatingText.
Fig. LV Script EnemyDestroyerScript.
Fig. LVI Objecte DestroyObjects.
Fig. LVII Script ParachuteZone.
Fig. LVIII Zona d'obertura del paracaigudes.
Fig. LIX Script ParachuteScript.
Fig. LX Tag Collisions.
Fig. LXI Script ewCollisionController.
Fig. LXII Script ButtonGamemanager part 1.
Fig. LXIII Script ButtonGamemanager part 2.
Fig. LXIV Script ButtonGamemanager part 3.
Fig. LXV Joc i menú pausa.
Fig. LXVI Game Over i Game finished.
Fig. LXVII Script GameManager part 1.
Fig. LXVIII Script GameManager part 2.
Fig. LXIX Script GameManager part 3.
Fig. LXX Configuració nivells.
Fig. LXXI Script CameraBoundsSize.
Fig. LXXII Mostra error límits moviment.
Fig. LXXIII Millora script EnemyScript.
Fig. LXXIV Mostra reducció BoxCollider enemic.
Fig. LXXV Funcionament de l'aplicació 1.
Fig. LXXVI Funcionament de l'aplicació 2.
Fig. LXXVII Funcionament de l'aplicació 3.
Fig. LXXVIII Funcionament de l'aplicació 4.
Fig. LXXIX Llista d'imatges importades.
Fig. LXXX Menú configuració imatge.
Fig. LXXXI Configuració compressió imatge.
Fig. LXXXII Resultat final compressió imatge.
Fig. LXXXIII Menú File Unity.
Fig. LXXXIV Build apk file Unity.
Fig. LXXXV Menú PlayerSettings Unity.
Fig. LXXXVI Menú OtherSettings Unity.
Fig. LXXXVII Menú versió Android mínima.

1.- Introducció.

1.1.- Motivació i descripció del problema.

L'objecte d'aquest treball és el disseny i implementació, amb Unity, d'un videojoc centrat en dispositius mòbils i tauletes amb sistema operatiu Android a partir de la versió 2.3 o superior.

Des de ben petit m'han fascinat els videojocs. Sempre em preguntava com ho podien fer perquè, amb un simple comandament, una consola o un televisor es veiés el personatge moure's, saltar, aconseguir punts i matar als enemics per poder salvar a la princesa. Amb els anys vaig anar veient que aquest sector sempre m'atrauria i ni que fos com a hobby, m'agradaria crear-ne algun. És per això que vaig iniciar-me al món de la informàtica i ara m'ha sortit l'oportunitat d'aplicar els coneixements apresos durant aquests anys en el meu primer disseny i implementació d'un videojoc.

La idea em va sorgir aquest mateix estiu, quan estava pensant si fer un joc o una aplicació com a treball final de màster. Si feia un joc, volia que fos una cosa nova, que hagués pensat jo i que pogués portar a terme tot i no tenir coneixements en disseny, música, efectes, ni en desenvolupament de videojocs. Sense pensar-ho em vaig decidir a utilitzar Unity com a motor de desenvolupament; ja que és una eina que cada dia està sent més utilitzada pels desenvolupadors, amb la qual ja tenia una mica d'experiència degut a una col·laboració amb un videojoc d'un amic, i ja portava temps volent treballar-hi de forma més exhaustiva, també sabia que em podria aportar nous coneixements i obrir portes en el sector.

En aquell moment ja tenia el com, però em faltava el què. Vaig pensar que el joc podria consistir en un petit extraterrestre que apareixia prop de la terra amb la seva nau traient fum. L'alien hauria de saltar de la nau si volia sobreviure i iniciar un descens cap a la terra. En aquest descens, s'aniria trobant objectes que hauria d'esquivar si no es volia fer mal o perdria la partida. Aquests objectes podrien anar variant en funció de la distància del terra en que es trobés; per exemple al principi de tot trobaríem meteorits, satèl·lits...i a mesura que ens anéssim apropant més i més al sòl terrestre ja començarien a aparèixer avions, coets... A més, durant el descens, podríem trobar monedes que ens donarien punts per ajudar-nos a pujar de nivell.

Per suposat, el joc o la partida havia de tenir un final, ja que no podíem estar en descens per sempre. Per això vaig pensar que l'usuari tindria un botó que obriria un paracaigudes i podria acabar el descens de forma segura. Això, però, comportava que l'usuari pogués obrir el paracaigudes en qualsevol moment i fer un descens més lent. La solució podria consistir en què l'usuari hagués d'activar el paracaigudes entre dos punts concrets, d'aquesta forma si ho feia massa aviat, o massa tard, el funcionament no fos l'esperat i perdessin la partida. En canvi, si ho feien en el moment adequat, el descens finalitzaria de forma segura i el jugador acabaria la partida amb èxit.

La única cosa que em faltava era comprovar si el joc ja existia o n'hi havia algun de semblant ja publicat. Realitzant algunes cerques a la botiga de Android, no vaig trobar cap joc que s'assemblés, ni en contingut ni en funcionament. Per tant, estaríem parlant d'un joc nou, el qual no es basa en cap altre.

Finalment vaig iniciar el procés de matriculació i de realització de la proposta, exposant part dels continguts plantejats anteriorment. Per tant, en el present document trobarem tots els passos que seguirem al llarg del projecte, dividits en apartats estructurats en funció de les necessitats de cada moment i la justificació de cadascuna de les decisions que s'han pres i com ha estat implementat cada apartat del joc "Ew, the small alien".

1.2.- Objectius del treball.

En aquest apartat trobarem els objectius principals i altres, que poden quedar fora de l'abast del projecte.

- Objectius principals:
    - Pantalla principal amb menú.
    - Pantalla de configuracions.
    - Pantalla del joc on els objectes es generin de forma aleatòria.
    - Creació d'un sistema de nivells, on segons el nivell que es tingui apareixeran més o menys objectes, s'adquirirà més o menys velocitat de caiguda i on el marge d'obertura del paracaigudes serà més gran o més petit en funció del nivell.
    - Disseny d'objectes que apareixeran per la pantalla (satèl·lits, coets, etc.).
    - Disseny i integració de música i efectes de so.
    - Compatibilitat amb mòbils i tauletes.
    - Compatibilitat amb Android 2.3 o superior.
    - Versió funcional del joc.

- Objectius que es poden quedar fora de l'abast del projecte:
    - Incloure diferents dissenys de pantalla.
    - Incloure pantalla d'estadístiques de punts.

1.3.- Enfocament i metodologia.

Un cop analitzades les necessitats del desenvolupament, es va arribar a la conclusió que s'hauria de desenvolupar un producte nou, ja que no podíem aprofitar el codi de cap joc per a la implementació.

Aquest projecte, doncs, es dividirà en quatre fases diferents.

- **Primera fase: Realització dels dissenys**. Es tracta de crear o buscar totes les imatges, sons, tipografies, música, etc. que s'utilitzaran per a realitzar tot el sistema del videojoc. A més a més, es dissenyaran, en

forma d'esborrany, les diferents pantalles, menús, etc. que contindrà el joc. D'aquesta manera tindrem una referència en què basar-nos i intentar complir estàndards de qualitat pel que fa la interfície gràfica. En cas d'aquell contingut que no estigui creat per nosaltres, se n'utilitzarà de caràcter públic i que tingui llicències obertes.

- **Segona fase: Creació de menús i submenús**. Un cop es tinguin tots els recursos disponibles, ja podrem importar-los en el Unity per tal de poder començar a implementar els dissenys realitzats en l'apartat anterior. En aquest punt ens assegurarem que les totes les funcionalitats dels diferents menús siguin correctes i se segueixin els estàndards dissenyats en tota l'aplicació.

- **Tercera fase: Creació del joc**. Amb les funcionalitats principals dels menús ja aplicades, podrem començar a realitzar la pantalla del joc, pròpiament dit. En aquest apartat durem a terme tot el que sigui necessari per a què el joc funcioni; per exemple afegir el personatge i els objectes, crear el sistema de col·lisions, implementar els controls del personatge i dels diferents objectes, etc. Al finalitzar aquesta fase, hauríem de tenir el joc amb totes les característiques que haguem definit tot i que encara podran existir petits errors de programació que seran corregits.

- **Quarta fase: Anàlisi, detecció i correcció d'errors**. En l'última fase de la creació del videojoc, analitzarem totes les funcionalitats que hem incorporat per a comprovar que no existeixin caràcters anòmals. Aquesta comprovació es realitzarà, no només amb el propi assistent de Unity, sinó que també en dispositius mòbils, tauletes i en màquines virtuals per assegurar la compatibilitat amb qualsevol tipus de dispositiu. En cas de detectar algun error, el corregirem per tal de finalitzar el projecte amb una versió estable del joc.

Al dissenyar l'estratègia hem tingut en compte, principalment, quins són els requisits de cada fase, per poder passar a la següent. Per exemple, no podem dissenyar el menú sense haver triat o dissenyat els botons que utilitzarem o la tipografia, o no podem tirar endavant el joc si no tenim en funcionament parts bàsiques com els menús.

1.4.- Àmbit d'aplicació.

Aquest projecte pretén arribar a tota la població en general, que tingui accés a dispositius mòbils, però sobretot que sigui apte per a la població infantil, realitzant una aplicació atractiva pels infants, sense contingut bèl·lic i que permeti millorar l'habilitat del jugador amb l'entreteniment que proporcionen els jocs de partides ràpides.

El grau d'afectació i integració en el mercat dependrà de la plataforma i nivell d'aplicació seleccionat. En el nostre cas es tracta d'una aplicació per a dispositius Android. Actualment, aquest sistema operatiu, té la major part de la quota de mercat mundial de dispositius mòbils.

Respecte al nivell de versió de distribució, podem veure en la següent imatge que actualment la major part dels dispositius utilitzen versions 2.3 o 4 de l'API d'Android. Nosaltres hem seleccionat la versió 2.3.1 com a plataforma mínima de compilació; per tant, com a mínim, arribarà al 97% dels dispositius que es troben en el mercat.

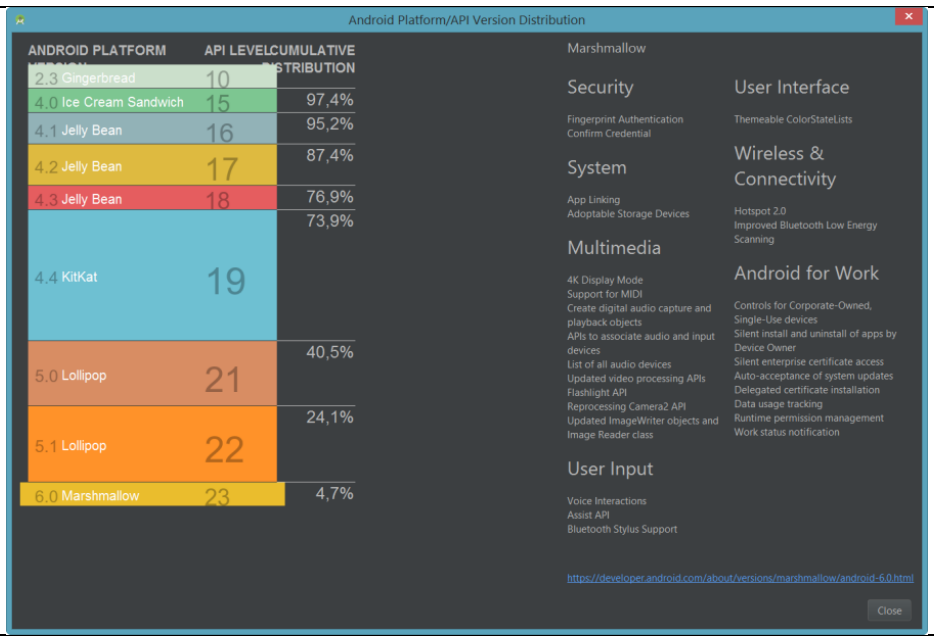


Fig. I Gràfica de versions d'API/Quota de mercat.

Gracies a això, podríem arribar a un nombre molt elevat d'usuaris, garantint la inclusió de qualsevol persona independentment de l'edat que tingui.

1.5.- Planificació del projecte.

Tot seguit veurem la planificació del projecte. La nostra idea serà que en la mesura del que sigui possible, de dilluns a dijous, dedicar-hi d'una a tres hores. Els divendres, quatre o més hores i els caps de setmana quatre hores sempre que es pugui i deixant algun dia de descans.

Primer de tot observarem el diagrama d'estructura de treball i a continuació, la definició de tasques:

ID	Tasca	Duració	Inici	Finalització	Set.	Oct.	Nov.	Des.	Gen.
0	Definició del projecte	11 dies	21/09/2016	05/10/2016	X	X			
	PAC 1		05/10/2016	05/10/2016		X			
1	Direcció del projecte								
1.1	Seguiment i control		21/09/2016	11/01/2017	X	X	X	X	X
1.2	Coordinació		21/09/2016	11/01/2017	X	X	X	X	X
2	Disseny								
2.1	Definició de la història del joc	1 dia	06/10/2016	06/10/2016		X			
2.2	Disseny del personatge	4 dies	07/10/2016	10/10/2016		X			
2.3	Disseny dels obstacles	8 dies	10/10/2016	18/10/2016		X			
2.4	Disseny o elecció d'altres materials	2 dies	19/10/2016	20/10/2016		X			
2.5	Esborranys dels menús i pantalles	2 dies	21/10/2016	22/10/2016		X			
	PAC 2		02/11/2016	02/11/2016			X		
3	Implementació dels menús								
3.1	Menú principal	5 dies	23/10/2016	27/10/2016		X			
3.2	Menú de configuracions	3 dies	28/10/2016	30/10/2016		X			
3.3	Altres menús i configuracions	2 dies	31/10/2016	01/11/2016		X	X		
4	Implementació del joc					X	X	X	
	Implementació del joc	29 dies	02/11/2016	11/12/2016			X	X	
	PAC3		14/12/2016	14/12/2016				X	
5	Anàlisi detecció i correcció d'errors								
	Anàlisi detecció i correcció d'errors	15 dies	12/12/2016	30/12/2017				X	
6	Documentació								
6.1	Documentació	9 dies	08/12/2016	20/12/2016				X	
6.2	Memòria	23 dies	08/12/2016	09/01/2017				X	X
	PAC 4		11/01/2017	11/01/2017					X

Fig. II Diagrama de l'Estructura de Distribució del Treball (EDT).

Tasca 1: Direcció del projecte.

- Descripció de la tasca: Analitzar i controlar l'evolució del projecte.
- Objectius de la tasca: Tenir un control exhaustiu de què s'ha fet, què falta fer i si es compleixen o no els terminis plantejats.

Tasca 2: Disseny.

- Descripció de la tasca: Dissenyar o buscar els diferents continguts, objectes o materials pel joc.
- Objectius de la tasca: Tenir el material necessari per a poder dur a terme la resta d'apartats.

Tasca 3: Implementació dels menús.

- Descripció de la tasca: Realitzar les diferents interfícies gràfiques pel joc.
- Objectius de la tasca: Programar les diferents funcionalitats que formen l'estructura del menú del joc.

Tasca 4: Implementació del joc.

- Descripció de la tasca: Dur a terme tota la implementació referent al joc, amb Unity.
- Objectius de la tasca: Obtenir una versió funcional del joc.

Tasca 5: Anàlisi, detecció i correcció d'errors.

- Descripció de la tasca: Provar de forma exhaustiva el joc, en diferents dispositius, per assegurar el seu bon funcionament.
- Objectius de la tasca: Obtenir una versió estable del joc, sense errors i compatible amb el major nombre de dispositius possible.

Tasca 6: Documentació.

- Descripció de la tasca: Crear la documentació, tant la memòria, com els comentaris en el codi del joc.
- Objectius de la tasca: Crear el document final pel TFM.

1.6.- Maquinari i programari utilitzat.

Com a requisits per a la implementació del sistema necessitarem:

Maquinari:

- Ordinador personal.
- Dispositius mòbils (BQ Aquaris 5, BQ Aquaris E4.5)
- Tauleta (BQ Edison).

Programari:

- Windows 8.1 o Windows 10.
- Unity 5.5. com a motor de desenvolupament.
- TFS. On guardarem el codi font, per tal que es facin backups i control de versions.
- Eclipse Neon amb ADT Plugin 23.0.7 o Android Studio.
- Màquines virtuals Android.
- Java per a la creació de llibreries externes de Google o Android.
- C# i JavaScript per a la realització de scripts del joc.

Altres:

- Conta de Google.
- Conta de Google Play Developer Console, per poder accedir a serveis de desenvolupador.

1.7.- Sumari de materials obtinguts.

El resultat final estarà format pels següents productes:

Memòria TFM	Es tracta del present document on es realitza la descripció del treball, els dissenys creats, les fites a realitzar, reflexions i conclusions dels resultats obtinguts.
EW_Alien.APK	Aplicació funcional compilada i preparada per a la seva instal·lació.
Presentació	Document amb la presentació del treball.

1.8.- Organització de la memòria.

Capítol 2: Història i Disseny del joc.

En el segon capítol es farà una descripció de la història del joc i s'introduiran els diferents dissenys creats per a la realització d'aquest, com també les fonts i objectes seleccionats per a la creació dels menús en els següents capítols.

Capítol 3: Implementació dels menús.

En el tercer capítol trobarem informació de la implementació dels menús que contindrà el joc i les diferents opcions que hi trobarem en cadascun d'ells.

Capítol 4: Implementació del joc.

En el quart capítol, trobarem tota la informació referent a la implementació del joc pròpiament dit. Concretament trobarem des de la justificació, els requisits del sistema, fins a la creació de les diferents parts que conformaran el joc.

Capítol 5: Anàlisi, detecció i correcció d'errors

En el cinquè capítol trobarem els anàlisis realitzats per a la detecció d'errors i les correccions que s'hauran dut a terme per a solucionar-los.

Capítol 6: Valoració econòmica.

En el sisè capítol, veurem el cost econòmic que podria tenir la creació d'aquest videojoc; des del disseny fins a la posada en marxa.

Capítol 7: Conclusions i línies de futur.

Finalment, en el setè capítol, es realitzarà una reflexió crítica de tots els aspectes referents a aquest projecte, a la seva implementació, al seu funcionament, futures línies de treball i l'experiència extreta en tot el procés.

2.- Història i disseny del joc.

Per tal de poder entendre i dur a terme de forma correcta el joc, primer de tot hem de definir la història d'aquest i els diferents objectes que s'han dissenyat per a poder-lo crear, ja que sense aquests no es podrien realitzar els demés apartats.

2.1.- Definició de la història del joc.

L'Ew és un petit extraterrestre que viatja per tot l'espai intentant descobrir nous planetes. Quan s'està aproximant a la terra, la seva nau s'espatlla i comença a treure fum. Només li queden dues opcions, estrellar-se contra la terra o saltar de la seva nau tot iniciant un descens a gran velocitat el qual no estarà absent de perills.

Meteorits, satèl·lits, avions i coets són els perills que es trobarà en aquest arriscat descens, que haurà d'esquivar si vol tenir alguna possibilitat de salvar-se. Però per a fer-ho també haurà de disminuir la velocitat; per sort abans de saltar de la nau ha agafat un paracaigudes que li permetrà disminuir la seva velocitat i aterrar a la terra. El paracaigudes, però, s'ha d'obrir en el moment indicat, sinó l'Ew perdrà el control del descens incrementant, de forma considerable, la seva velocitat de caiguda tenint un final tràgic.

Durant el descens, l'Ew també es trobarà amb altres objectes com monedes, que li permetran aconseguir punts, i núvols, que li dificultaran la visibilitat d'objectes enemics que poden provocar la pèrdua de la partida.

2.2.- Disseny del Logotip.

El primer disseny que vam realitzar va ser el logotip que tindria l'aplicació i que esta format per l'escut o les monedes del planeta de l'Ew.

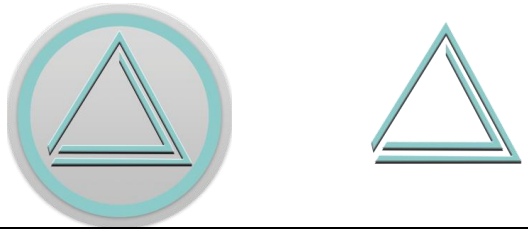


Fig. III Logotip de l'aplicació i escut de Arcadium.

Com podem veure, es tracta d'un disseny simple i minimalista que intenta representar el planeta d'on prové l'Ew, Arcadium. La forma de l'escut intenta representar la A del nom del planeta, tot tancant la part inferior del triangle i afegint dues parts del triangle per representar la lletra.

2.3.- Disseny del fons de pantalla.

El següent pas va ser realitzar el disseny dels fons de pantalla, tant del menú principal com de la pantalla del joc. En el cas del primer ens vam centrar en representar la immensitat de l'espai, mostrant galàxies i estrelles, on es representa l'origen del nostre extraterrestre i en la part inferior, les muntanyes amb els núvols a la llunyania de forma difuminada i amb ombres, intentant representar el nostre petit planeta on l'Ew vol aterrar.

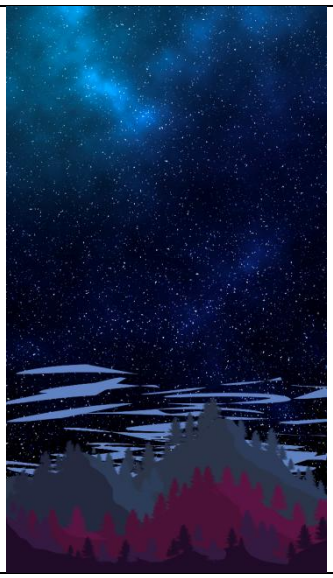


Fig. IV Disseny del fons de menú.

Un cop vam tenir el primer disseny, vam aprofitar la tècnica realitzada per a la creació de l'espai, per a dur a terme el que seria el fons que representaria la caiguda des de l'espai de l'Ew cap a la terra. Com veurem, la següent imatge és molt llarga, això és perquè hem de representar la caiguda des de l'espai fins al terra. En propers apartats veurem com la càmera anirà recorrent la imatge de dalt a baix.

Fig. V Disseny del fons del joc.

2.4.- Disseny del personatge.

Un cop ja teníem dissenyats els fons de pantalla, era hora de crear el nostre petit extraterrestre. Aquest havia de tenir un disseny simpàtic, amb colors clars i que fos atractiu tant pels nens com pels adults; també havia de tenir incorporat l'escut del seu planeta. El primer pas va ser fer un esborrany de com hauria de ser. Tot seguit, vam intentar perfilar-lo perquè tingués unes línies més clares i li vam posar color per a fer-lo més atractiu. Els resultats els podem veure en la següent figura.

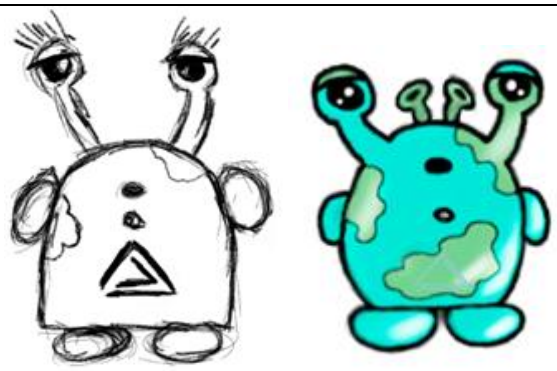


Fig. VI Esborrany i versió final de l'Ew.

Com podem veure, la millora és significativa, millorant els traçats, aplicant colors clars, fent ombres i reflexes en el seu cos.

2.5.- Disseny dels obstacles.

El següent pas va ser realitzar el disseny dels diferents objectes enemics que hauríem d'intentar esquivar en el descens. Aquests estan col·locats en l'ordre en què ens els anirem trobant en el joc.

2.5.1.- Meteorits.

Els primers que ens trobarem són els meteorits, ja que són els que estan més allunyats de la terra. Com veiem són els hem realitzat de diverses mides, formes i colors per tenir varietat.

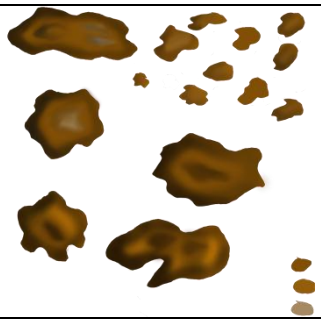


Fig. VII Disseny Meteorits.

2.5.2.- Satèl·lits.

Els següents objectes que ens trobarem són els satèl·lits. Aquests estaran situats en el segon apartat d'objectes. En la següent figura veurem que tenim dos dissenys diferents.

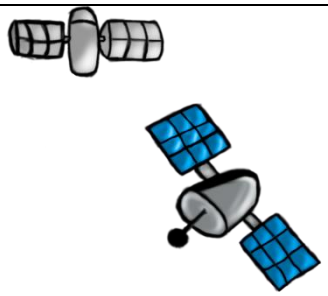


Fig. VIII Disseny satèl·lits.

2.5.3.- Coets.

Els coets serien els tercers en trobar-nos. Aquests, a més, se situarien en una zona on començarien a aparèixer núvols i es podrien amagar darrere i generar-nos problemes. Els dissenys creats són els següents.

Fig. IX Disseny coets.

2.5.4.- Avions.

Els últims objectes que ens podrien fer perdre la partida són els avions, que també es trobaran en la zona de núvols, a més de ser més grans que la resta i dificultar més el moviment de l'Ew.

Fig. X Disseny avions.

2.6.- Altres dissenys.

Tot seguit mostrem altres dissenys d'objectes necessaris pel joc.

2.6.1.- Paracaigudes.

El paracaigudes ens permetrà disminuir, de forma considerable, la velocitat de caiguda, a més s'haurà d'integrar amb l'Ew. El disseny realitzat és el següent.

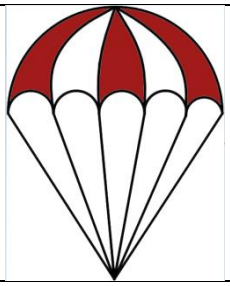


Fig. XI Disseny del Paracaigudes.

2.6.2.- Monedes.

Les monedes ens permetran aconseguir punts en el nostre descens. En aquest cas hem decidit crear un disseny dinàmic i que tingués moviment. Per això hem fet una seqüència d'imatges. A més, cadascuna d'elles conté l'escut de Arcadium. En la següent figura podem veure la seqüència creada.

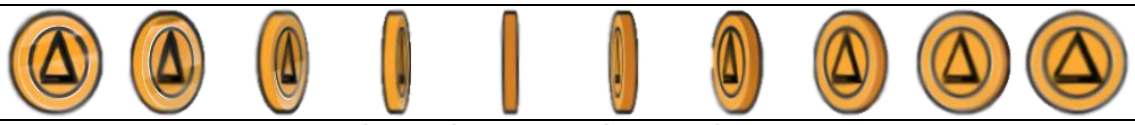


Fig. XII Disseny moneda en moviment.

Un cop importat en el Unity, crearem una animació per simular la rotació de la moneda. Així aconseguirem un sistema més dinàmic i atractiu.

2.6.3.- Núvols.

Els núvols permetran ocultar altres objectes, com coets o avions. Tot i que no tindran una afectació directa amb l'Ew, ens poden fer perdre una partida si darrere tenen amagat un objecte.

Fig. XIII Disseny núvols.

2.6.4.- Nau espacial.

El següent disseny que vam realitzar va ser la nau de l'Ew i que servirà per crear una animació en el menú principal del joc.

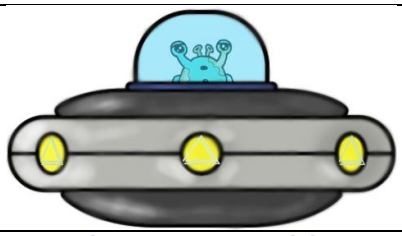


Fig. XIV Nau espacial.

2.7.- Disseny o elecció de música, tipografia i botons.

En aquest apartat trobarem altres objectes que necessitarem per a la part de menús del joc.

2.7.1.- Música.

Els jocs solen tenir efectes de so, o com a mínim música en els menús. En el nostre cas hem creat una petita seqüència pel menú principal per a fer-lo més dinàmic i entretingut. La podem escoltar reproduint el següent fitxer adjunt.

ew_bandasonoraV0.1.mp3

2.7.2.- Tipografia.

La tipografia és una de les coses més llargues de dissenyar. En el nostre cas quedava fora de l'abast del projecte, però volíem utilitzar una tipografia diferent a les habituals; és per això que en vam utilitzar una que fos pública i d'ús amb dispositius mòbils sense incomplir cap sistema de llicències. Concretament vam utilitzar la següent.

Ew, the small alien

Fig. XV Mostra tipografia "Spicy Rice".

L'origen d'aquesta tipografia la podem trobar en la següent referència:

Font Squirrel, Spicy Rice Typography -
https://www.fontsquirrel.com/fonts/spicy-rice [31 d'octubre del 2016].

2.7.3.- Botons.

En el cas del dissenys dels botons, hem recorregut a la mateixa solució que en el cas anterior. En el joc utilitzarem els botons de color verd ja que són els que donen una imatge més clara en el disseny de la interfície.

Fig. XVI Disseny Botons de "Game Art 2D".

L'origen d'aquests dissenys el podem trobar en la següent referència:

Game Art 2D - http://www.gameart2d.com/gui.html - [1 de Novembre del 2016].

2.8.- Sketches i prototip.

En aquest apartat trobarem els Sketches, són una sèrie de dibuixos fets a mà alçada que ens permeten representar com seria la interfície, per a poder ser analitzats i corregits en el mateix moment. Aquests dissenys podran variar des d'aquest punt fins a la versió final, però ens donaran una guia de com seran les diferents pantalles i menús que conformaran el joc.

2.8.1. - Pantalla principal.

En aquesta primera imatge, podem trobar l'esquema de com seria el menú principal. Al mig de la pantalla trobaríem el títol del joc. En la part superior dreta, trobarem un botó per apagar o engegar la música. En la part del mig inferior trobarem tres botons, el primer iniciarà el joc, el segon ens portarà a la pantalla d'opcions i l'últim ens mostrarà els crèdits del joc.

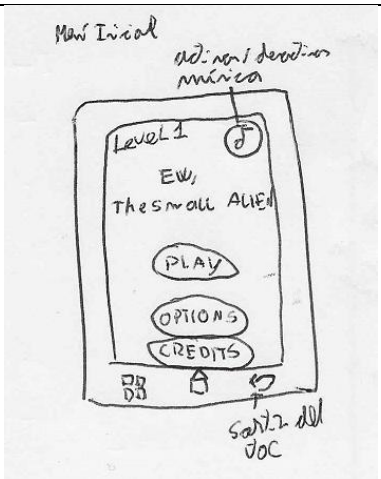


Fig. XVII Sketch pantalla principal.

2.8.2.- Menú d'opcions o configuracions.

El segon esquema que trobem és el del menú de configuracions. En aquest podrem configurar el volum de la música i també la sensibilitat dels moviments del joc. En la part superior tindrem un botó per desactivar la música i un altre que ens retornarà al menú principal.

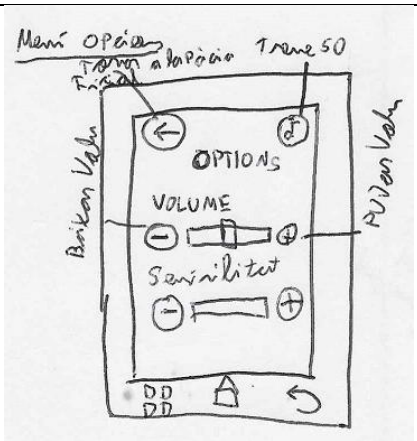


Fig. XVIII Sketch menú opcions.

2.8.3.- Pantalla de crèdits.

La següent pantalla que trobarem serà la pantalla de crèdits, on hi haurà la informació de totes les persones que han col·laborat en el joc. Com amb les altres pantalles, es podrà desactivar i activar la música i tornar al menú principal.

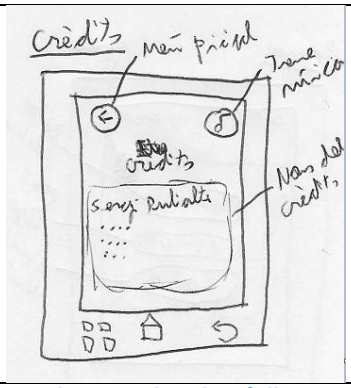


Fig. XIX Sketch crèdits.

2.8.4.- Pantalla del joc.

El següent sketch ens mostra com seria la pantalla un cop ens trobéssim en el joc. Aquí distingirem tres parts:

- La primera és la part superior, on trobarem informació dels punts que portem acumulats en la partida actual, d'aquesta forma tindrem informació de l'evolució de la partida.
- En la part central trobarem el personatge i els obstacles que haurem d'esquivar. Per a fer-ho girarem en l'eix Y (vertical) el telèfon, d'aquesta forma l'Ew es desplaçarà cap un costat o cap a l'altre de la pantalla.
- Finalment, en la part inferior, trobarem un botó que ens permetrà obrir el paracaigudes. Com ja vam explicar, aquest botó s'haurà d'obrir en el moment concret, sinó es perdrà la partida.

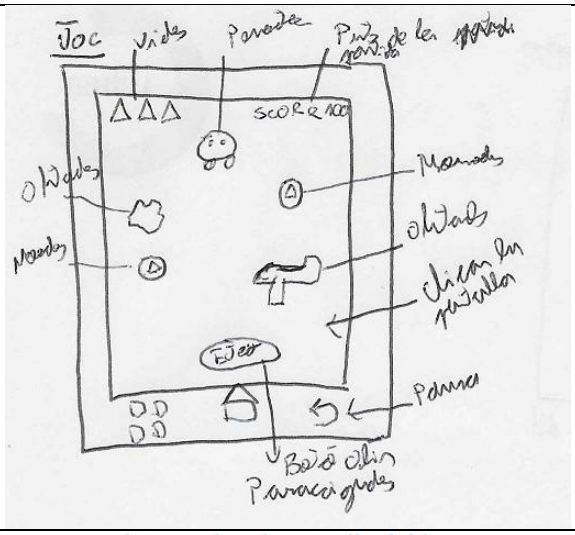


Fig. XX Sketch pantalla del joc.

2.8.5.- Menú pausa.

Durant el transcurs del joc, l'usuari pot voler aturar la partida, ja sigui per sortir del joc o per modificar alguna de les opcions de configuració. Al prémer el botó de pausa, ens apareixerà aquest menú sobreposat al joc i aquest s'aturarà en el moment en què ens trobem.

En aquest menú trobarem tres botons, un de continuar, que ens permetrà seguir jugant. El següent és el que ens permetrà reiniciar la partida i finalment un últim botó que ens portarà al menú principal del joc.

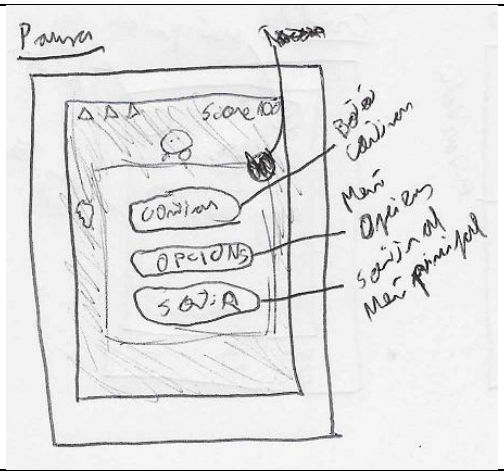
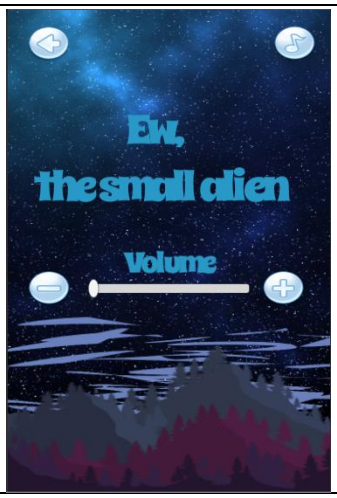


Fig. XXI Sketch menu pausa.

Tots aquests Scketches ens permeten representar, de forma bàsica i provisional, les diferents pantalles i accions que podran realitzar els usuaris en l'aplicació. Destacar que aquest és un disseny provisional i que com en tot procés de DCU, pot rebre canvis al transcurs del projecte i incloure noves funcionalitats.

2.8.6.- Prototip horitzontal d'alta fidelitat

En aquest apartat, hem dissenyat un prototip horitzontal d'alta fidelitat que inclou les principals funcions i pantalles que podrà veure i utilitzar l'usuari en el joc. Concretament, hem creat la pantalla principal, la d'opcions, la pantalla de crèdits i la del joc. Això ens permet tenir una visió de com podrà ser el joc, i fer-nos una idea de les principals parts i estètiques utilitzades i que ens permetrà millorar en el pas d'implementació si s'escau.



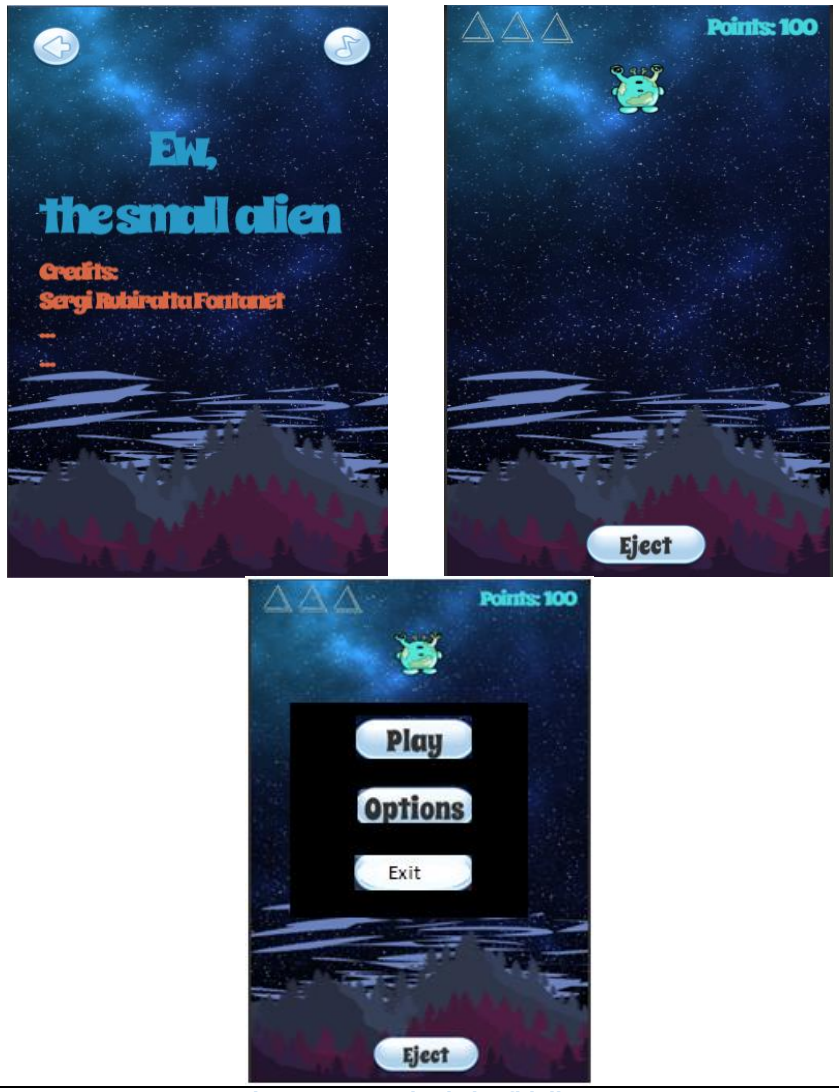


Fig. XXII Prototip d'alta fidelitat.

3.- Implementació dels menús.

En aquest capítol trobarem tota la informació referent a la implementació dels menús del joc, els quals estaran representats amb Scenes de Unity, que ens permeten emmagatzemar i configurar objectes de forma independent. D'aquesta forma podem dividir el joc en petites peces, o escenaris.

3.1.- Menú principal.

La primera que trobem és la SceneMainMenu del menú principal. Aquesta conté totes les opcions de navegació que ens permetrà realitzar l'aplicació, a més d'un botó que ens permet activar o desactivar la música dels menús i que trobarem en la resta de pantalles exceptuant la del joc que no té música.

Per a poder carregar qualsevol scene en funció de la opció del menú que ens seleccionin, hem creat un script anomenat MenuManager que té una funció que rep com a paràmetre el nom de la scene, la carrega i l'obre automàticament. En la següent figura podem veure el codi del MenuManager.

```
//script to load new scene
public class MenuManager : MonoBehaviour {

    //This function recibes the scene name to load
    public void ChangeScene(string sceneName)
    {
        //load the new scene
        SceneManager.LoadScene(sceneName, LoadSceneMode.Single);
    }

}
```

Fig. XXIII Script MenuManager.

Aquesta funció però, ha de ser cridada en el moment que ens premin un botó, per exemple el de Options, amb el paràmetre correcte. Per això Unity ens facilita la feina i quan afegim un botó en el nostre escenari, ens crea un mètode anomenat OnClick() al què li hem d'indicar quines accions fer quan es premi el botó. En la següent figura es pot veure un exemple on es configura el botó d'opcions i es carrega la SceneConfiguration que hem posat a la càmera, aquesta acció es repetirà per tots els botons dels menús.

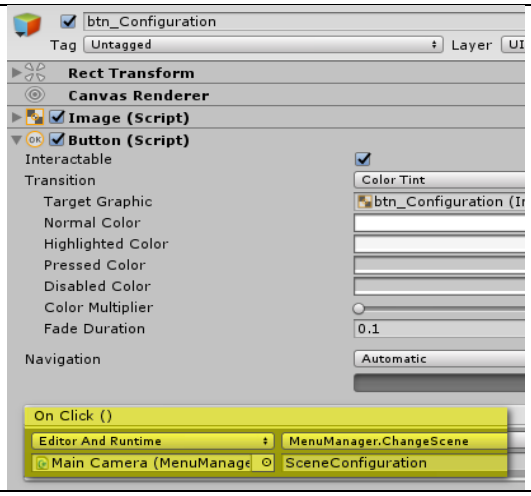


Fig. XXIV Configuració acció botó btn_configuration.

Per evitar tenir un menú estàtic i avorrit, es va decidir crear un sistema que mogués, seguint un patró definit, el personatge amb la seva nau traient fum, fins que arribés al terra, on representa que s'estavellarà i començarà a sortir fum de darrera les muntanyes.

Primer de tot s'havia de crear el camí que hauria de seguir la nau, per això es va fer un script anomenat PathHolderScript. Aquest ens generaria una sèrie de punts en la pantalla i ens permetria moure'ls de tal forma que creéssim una seqüència ordenada i que serviria com a indicador del camí.

Per a visualitzar de forma més còmode el camí i els punts a seguir, vam utilitzar una funció de Unity anomenada Guizmos que permet pintar en l'editor objectes o línies representatives, d'aquesta forma els punts els vam pintar amb un cercle i vam generar una línia entre els diferents punts. En la següent figura podem veure el codi que realitza aquesta acció.

```
public class PathHolderScript : MonoBehaviour {

    //variable with the color of the lines and points visibles in the editor
    public Color rayColor = Color.white;
    //list of points that object will follow
    public List<Transform> path_objs = new List<Transform>();
    //array with all the transform points generated
    Transform[] theArray;

    //this function will create a invisible line (visible on the editor) to make a object follow
    void OnDrawGizmos()
    {
        //set the colour
        Gizmos.color = rayColor;
        //get all object or points created to the array
        theArray = GetComponentsInChildren<Transform>();
        path_objs.Clear();
        //load all transforms on the list
        foreach (Transform path_obj in theArray)
        {
            if (path_obj != this.transform)
            {
                path_objs.Add(path_obj);
            }
        }
        //here will draw a line betwen points
        for (int i = 0; i < path_objs.Count; i++)
        {
            Vector3 position = path_objs[i].position;
            if (i > 0)
            {
                //Printem la linia entre el punt anterior i el actual
                Gizmos.DrawLine(path_objs[i - 1].position, position);
                //Printem en el editor un cercle indicant el punt
                Gizmos.DrawWireSphere(position, 0.1f);
            }
        }
    }
```

Fig. XXV Script PathHolderScript.

En la següent figura podem veure com en la pestanya Scene trobem el camí generat a partir dels diferents punts i en la del joc no es veuen. Finalment, en l'inspector podem veure com hem configurat el script perquè ens permeti canviar el color de la línia generada i els vint-i-quatre punts que conformen el recorregut que realitzarà la nau.

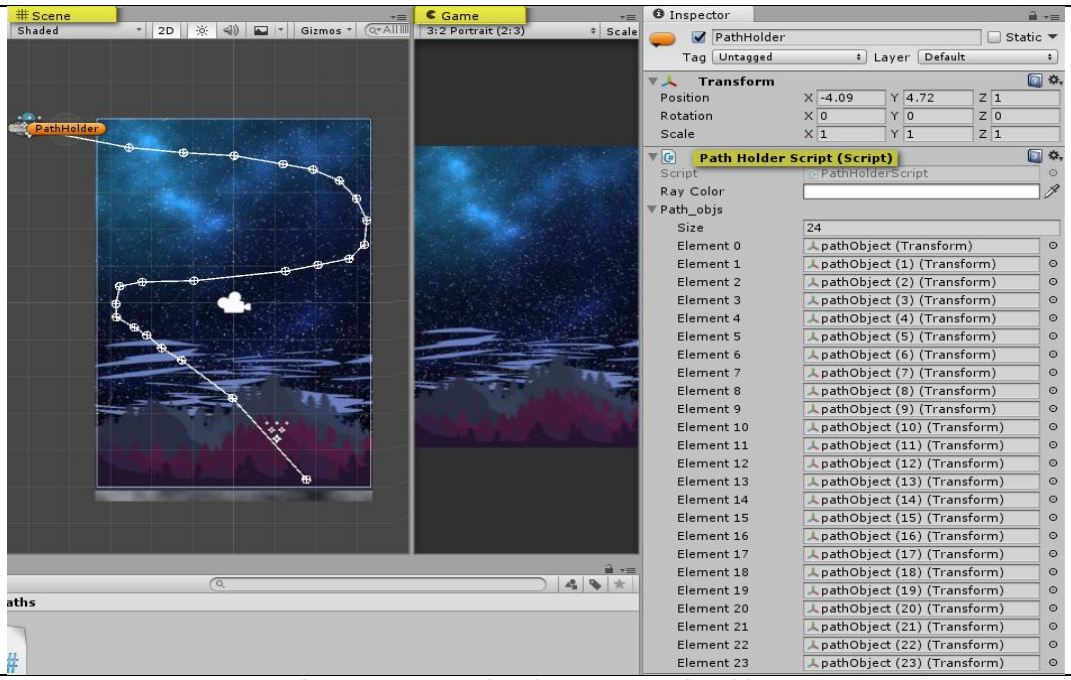


Fig. XXVI Mostra funcionament PathHolder.

El següent pas serà el d'afegir moviment a l'objecte de la nau perquè segueixi el camí generat anteriorment. Per això es va crear un altre script anomenat MoveOnEdithorPath, que rep el contenidor amb el camí creat anteriorment. Amb aquest contenidor, podem saber en quin punt ens trobem i quin és el següent punt a arribar, calcular la distància entre els dos i modificar la direcció de la nau cap al següent punt. Finalment, un cop hem recorregut tots els punts, activarem el fum darrera les muntanyes per simular que la nau s'ha estrellat. En la següent figura podem veure el codi del script.

```
//this script will move an object from all points created in the pathholder object
public class MoveOnEdithorPath : MonoBehaviour {

    //object with all the paths
    public PathHolderScript pathToFollow;
    //the current point we are
    public int currentWayPointId = 0;
    //the speed of the movement
    public float speed;
    private float reachDistance = 1.0f;
    //a vector3 to store the object position and help to move in the world
    Vector3 current_position;

    // Update is called once per frame
    void Update () {
        //if we are not in the last point path
        if(currentWayPointId < pathToFollow.path_objs.Count) {
            //calculate the distance from the game object to the next point
            float distance = Vector3.Distance(pathToFollow.path_objs[currentWayPointId].position,
                transform.position);
            //set the direction, and the speed to the next point to move the game object
            transform.position = Vector3.MoveTowards(transform.position,
                pathToFollow.path_objs[currentWayPointId].position, Time.deltaTime * speed);
            //if distance to the next point is smallen or equal 1 it means we are in the next point
            if(distance<= reachDistance)
            {
                //then we have to go to the next point
                currentWayPointId++;
            }
        }else{
            //if we arribe at the last point, we can start the smoke from begind mountains
            GameObject.Find("backgroundSmoke").GetComponent<playSmoke>().startSmoke();
        }
    }
}
```

Fig. XXVII Script MoveOnEdithorPath.

Tot seguit podem veure una figura de com queda el menú principal amb la nau en moviment i treient fum tal com vam indicar ja, que la nau està espatllada.

Fig. XXVIII Menú Principal moviment.

L'últim pas és el de crear el fum, tant en la nau com darrera les muntanyes. Per això s'utilitza un sistema: Particle System, que genera automàticament partícules i que permet molts valors de configuració on es poden canviar la mida, color, direcció, etc. de les partícules creades. Però aquest sistema també s'ha de controlar, per això es va crear un nou script anomenat playSmoke i que ens permet activar el fum a partir de la funció que vam crear anomenada startSmoke().

```
//function to control the smoke of main menu spaceShip
public class playSmoke : MonoBehaviour {

    //variable to control if smoke has to be played or not
    bool playSmokeVar;

    void Start () {
        //in main scene , background smoke hasn not to be played
        playSmokeVar = false;
        GetComponent<ParticleSystem>().Stop();
        //if we are in a not main scene, backround smoke has to be played
        if(!SceneManager.GetActiveScene().name.Equals("SceneMainMenu"))
        {
            GetComponent<ParticleSystem>().Play();
            playSmokeVar = true;
        }

    }

    void Update()
    {
        //check if smoke has to be played
        if (!playSmokeVar)
        {
            GetComponent<ParticleSystem>().Play();
        }
    }

    /*This function lets us change the state of playsmokevar to start the smoke when the
     *spaceship collision to the flor
     */
    public void startSmoke()
    {
        playSmokeVar = true;
    }
}
```

Fig. XXIX Script PlaySmoke.

Finalment, en la seguent figura podem veure com es reprodueix el fum darrere les muntanyes, tant en el propi joc, com si movem la càmera en l'editor. Indicar que aquest fum apareixerà en totes les pantalles exceptuant la del joc.

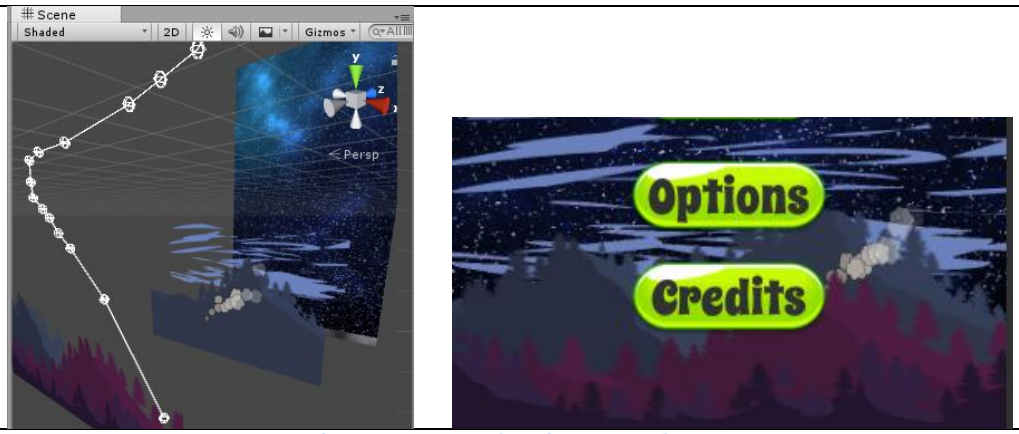


Fig. XXX Mostra funcionament fum.

3.2.- Menú de configuracions.

El següent menú o escena que hem creat és el d'opcions o SceneConfiguration. Aquest menú és el que ens permetrà modificar, per un costat, el volum de la música i per l'altre la sensibilitat del moviment de l'Ew.

Primer de tot ens centrarem en el control de la música. Per a poder tenir música en els diferents menús havíem de crear un objecte que la controlés i tingués la propietat d'afegir-la. Per això es va crear MusicManager en què vam afegir el component de Audio Source i vam configurar-lo perquè ens afegís el nostre clip d'àudio creat i que podem reproduir en el punt 2.7.1 del present document.

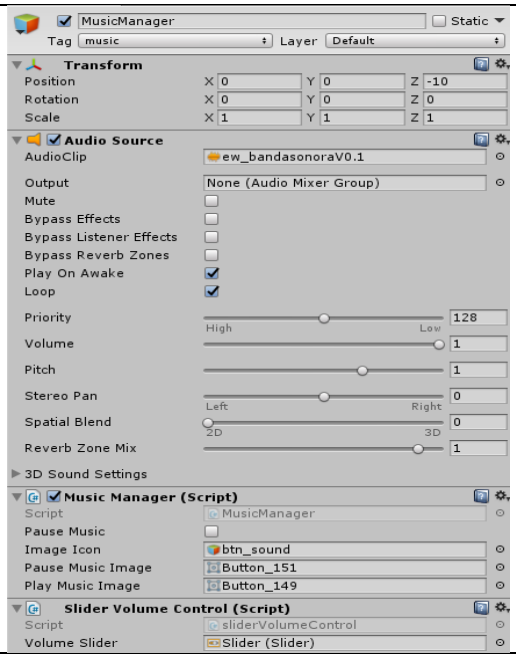


Fig. XXXI Music Manager configuration.

Amb això aconseguíem que la música s'escoltés de forma constant, però necessitàvem poder parar-la, canviar la icona de reproducció, i que quan canviéssim de menú, la música continués reproduint-se des del mateix punt. Per això vam crear un nou script anomenat MusicManager, que té una funció que para o torna a reproduir la música anomenada muteSong i es cridarà des

del botó de la part superior dreta dels menús. La resta de codi s'encarrega d'emmagatzemar el segon exacte en què ens trobem per a poder continuar reproduint-se quan la nova pantalla s'hagi carregat i modificar la icona. En la següent figura podrem veure el codi implementat.

```
//script to control music and volume
public class MusicManager : MonoBehaviour {

    //all variable and objects to controll audio
    public bool pauseMusic = false;
    public GameObject imageIcon;
    public Sprite pauseMusicImage;
    public Sprite playMusicImage;
    new AudioSource audio;

    void Awake()
    {
        //get the actual audio source
        audio = GetComponent<AudioSource>();
        //we set the las seconds where the song was playing
        audio.time = PlayerPrefs.GetFloat("audioTime");
        //we set the stored volume from stored variables
        audio.volume = PlayerPrefs.GetFloat("sliderVolume",1);
        //check if the music was stopped or not
        if (PlayerPrefs.GetInt("pauseMusic") == 1)
        {
            //if was stopped, we don't have to play it in this scene
            pauseMusic = true;
            //change the image
            imageIcon.GetComponent<Image>().sprite = pauseMusicImage;
        }
    }
```

```
// Update is called once per frame
void Update () {
    //here we check if music cas to be paused
    AudioListener.pause = pauseMusic;
    //store at what time of the song we are to continue on next scene
    PlayerPrefs.SetFloat("audioTime", audio.time);
}

//function to enable o disable music
public void muteSong()
{
    //if called, we change the music state
    pauseMusic = !pauseMusic;
    if (pauseMusic)
    {
        //We have to pause music
        //change the mute icon image
        imageIcon.GetComponent<Image>().sprite = pauseMusicImage;
        //store music state
        PlayerPrefs.SetInt("pauseMusic", 1);
    }
    else
    //else we have to start music
    {
        //change the mute icon image
        imageIcon.GetComponent<Image>().sprite = playMusicImage;
        //store music state
        PlayerPrefs.SetInt("pauseMusic", 0);
    }
}
```

Fig. XXXII Script MusicManager

Amb això ja podíem parar, reproduir o continuar la música; però també es volia poder modificar el volum d'aquesta. Per a fer-ho, vam afegir un slider que ens ho permetés. A més, per mantenir el volum entre les diferents pantalles, vam crear un segon script anomenat sliderVolumeControl que s'encarrega d'emmagatzemar el volum actual, així aconseguíem mantenir el volum entre les diferents pantalles segons la configuració de l'usuari. En la següent figura podem veure la configuració del slider del volum.

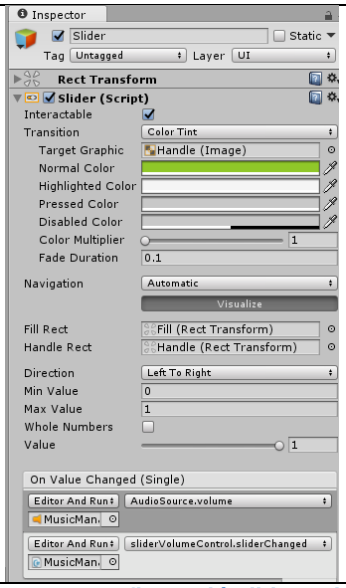


Fig. XXXIII Configuració slider volum.

La segona opció que volíem permetre modificar era la sensibilitat del moviment horitzontal de l'Ew, ja que no tots els telèfons tenen la mateixa o els usuaris volen poder moure el personatge més ràpid o més lent en funció de les seves habilitats. Per això vam afegir un segon slider, que funcionaria com el plantejat anteriorment, si augmentem el valor o la barra cap a la dreta augmentarem la velocitat de moviment, i si ho realitzem cap a l'esquerra, la disminuirem.

Aquest slider, però, per si sol no ens permetia visualitzar quin era el comportament actual en funció del valor seleccionat. És per això que vam decidir afegir el personatge en la part inferior del menú per a poder representar quin era el comportament a partir del canvi realitzat. D'aquesta forma l'usuari pot provar, en temps real i sense iniciar una partida, quin és el comportament en funció del valor seleccionat. Aquí, només ens faltava poder emmagatzemar aquest valor per a ser carregat en la pantalla del joc; per això es va crear el script sensitivityManager i que podem veure a continuació.

```
public class sensitivityManager : MonoBehaviour {

    //Sensitivity Slider GameObject
    public Slider slider;

    void Awake()
    {
        //Gets saved sensitivity value if exists, if not, set default value to 1
        slider.value = PlayerPrefs.GetFloat("sliderSensitivity", 3);
        //set this value on the Ew, to test horitzontal movement speed
        GameObject.Find("Ew").GetComponent<characterMovementController>().ewMovementSpeedHoritzontal = slider.value;
    }

    //Funtion to store volume slider value if changed and sets to Ew to test horitzontal movement speed
    public void sliderChanged()
    {
        PlayerPrefs.SetFloat("sliderSensitivity", slider.value);
        GameObject.Find("Ew").GetComponent<characterMovementController>().ewMovementSpeedHoritzontal = slider.value;
    }
}
```

Fig. XXXIV sensitivityManager.

Aquest script només serveix per emmagatzemar i afegir el valor del slider a l'Ew a partir de la funció ewMovementSpeedHoritzontal per ser provat en aquesta pantalla. La classe characterMovementController és l'encarregada de generar el moviment del personatge i que veurem en l'apartat 4 del present document.

El resultat final obtingut d'aquest menú és el de les següents figures on podem veure mostres amb diferents posicions de l'Ew i com la imatge de la música i els sliders varien.

Fig. XXXV Menú Configuration Final.

3.3.- Crèdits.

L'última pantalla de menús que hem creat és la dels crèdits. Aquesta simplement conté la meva informació i la informació del màster de la UOC, a banda d'altres objectes introduïts en els punts anteriors com el MusicManager i el fum darrera les muntanyes. Tot seguit podem veure el resultat obtingut.

Fig. XXXVI Menú crèdits final.

4- Implementació del joc.

Un cop vam tenir creats tots els menús, ens vam centrar en la implementació del joc pròpiament dit. Per tant, en aquest apartat trobarem tot el que conforma la scene del joc, des del moviment del personatge o objectes enemics, al sistema de puntuacions i nivells.

4.1.- Moviment del personatge i control de la càmera.

4.1.1.- Moviment del personatge.

El primer pas va ser crear el moviment de l'Ew, tant l'horitzontal com el de caiguda per l'espai. Per això es va crear el script characterMovementController. En aquest trobem dues variables que contenen el valor de la velocitat horitzontal i vertical. Recordem que el valor de moviment horitzontal ja l'hem introduït en el punt 3.2, on hem vist que el podíem modificar des del menú d'opcions o configuració. En canvi, el valor de caiguda vindrà donat pel nivell en què ens trobem, per exemple, en nivells més baixos la velocitat de caiguda serà menor i en nivells més alts serà superior.

Tot seguit necessitàvem poder modificar la direcció en què ens movíem en funció de com ens giraven el dispositiu. Aquí vam realitzar dues implementacions diferents per tal de poder provar el joc tant en dispositius Android com amb l'editor de Unity. Aquest últim permet especificar trossos de codi aplicables només per un dispositiu concret i sense que afecti a l'altre. Així doncs, el següent codi detecta el moviment. Si ens trobem en l'editor, s'aplica la direcció creada amb les fletxes del teclat de l'ordinador, en canvi, si utilitzem el telèfon s'aplica el valor de l'acceleròmetre, concretament el moviment detectat per l'eix X d'aquest. En la següent figura podem veure el tros de codi que realitza el moviment.

```
//called every frame
void Update () {
    //Here we can have multiple imputs to calculate the direction of the movement
    //if we play in the unity editor
    #if UNITY_EDITOR
        //get the direction and the speed of the X movement from keyboard
        ewPosition.x += Input.GetAxis("Horizontal") * ewMovementSpeedHoritzontal * Time.deltaTime;
    #endif
    //if we play in the unity editor
    #if UNITY_ANDROID
        //get the direction and the speed of the X movement from the accelerometer
        ewPosition.x += Input.acceleration.x * ewMovementSpeedHoritzontal * Time.deltaTime;
    #endif
    //calculate the clamp from the possibles values
    ewPosition.x = Mathf.Clamp(ewPosition.x, _minBounds.x + ewSize.x*2, _maxBounds.x - ewSize.x * 2);
    //set the falling speed based on time to complete a frame
    ewPosition.y += -ewMovementSpeedVertical * Time.deltaTime;
    //calculate the clamp from the possibles values
    ewPosition.y = Mathf.Clamp(ewPosition.y, _minBounds.y-10 + ewSize.y * 2, _maxBounds.y - ewSize.y * 2);
    //finaly set the values to Ew to let it move
    transform.position = ewPosition;
}
```

Fig. XXXVII Horitzontal Ew Movement Script.

Com podem veure en la imatge anterior, amb el codi "`Input.acceleration.x`" li multipliquem la velocitat configurada de moviment i el temps delta, que és el temps que triga a completar l'últim frame. Ara ja tenim la direcció amb la velocitat que pren l'Ew en aquest frame en l'eix X. Tot seguit calculem el mateix

per l'eix Y però sense tenir afectació pel giroscopi. Això ho realitzem així per si ens varien la velocitat de caiguda, per exemple quan s'obri el paracaigudes. Tot aquest codi, si ens hi fixem, es troba dins de la funció de update, aquesta s'executa en cada frame del joc i cal realitzar-ho així perquè volem assegurar-nos que el moviment s'apliqui el màxim de ràpid possible i aquest és cada frame.

Aquest moviment, però, havia d'estar limitat d'alguna forma, ja que si no ho fèiem, podríem moure el personatge, de forma indefinida cap als costats o la caiguda no tindria fi. Per aquest motiu vam crear un objecte anomenat CameraBounds i li vam afegir un Box Collider 2D per limitar la zona d'afectació del moviment. Aquest collider té la mateixa mida, tant horitzontal com vertical, que la imatge de fons que vam generar en el disseny del punt 2.3, d'aquesta forma podríem conèixer si l'Ew es trobava dins de la zona de la imatge i podríem evitar moviment cap a l'exterior.

Per a fer-ho, en la imatge anterior, hem vist que tant per l'eix X com per l'eix Y utilitzem una funció anomenada "`Mathf.Clamp`". Aquesta serveix per assegurar que un valor mai superarà un mínim o un màxim. Per tant, la utilitzem com a control per mantenir-nos dins del CameraBounds, amb l'ajuda de les posicions relatives del personatge i dels valors límits. Això, però, ens va generar un problema; la posició relativa de l'Ew que teníem era la del centre de l'objecte, per tant, sempre ens podia sortir mig cos del personatge pels costats de la pantalla. Per evitar-ho vam restar al mínim i sumar al màxim, la mida de l'Ew; d'aquesta forma ens asseguràvem que mai sortiríem de la pantalla. En la següent figura podem veure com el personatge es troba en els límits i no traspassa la barrera o box collider, que és la línia de color verd de la part esquerra de la imatge.

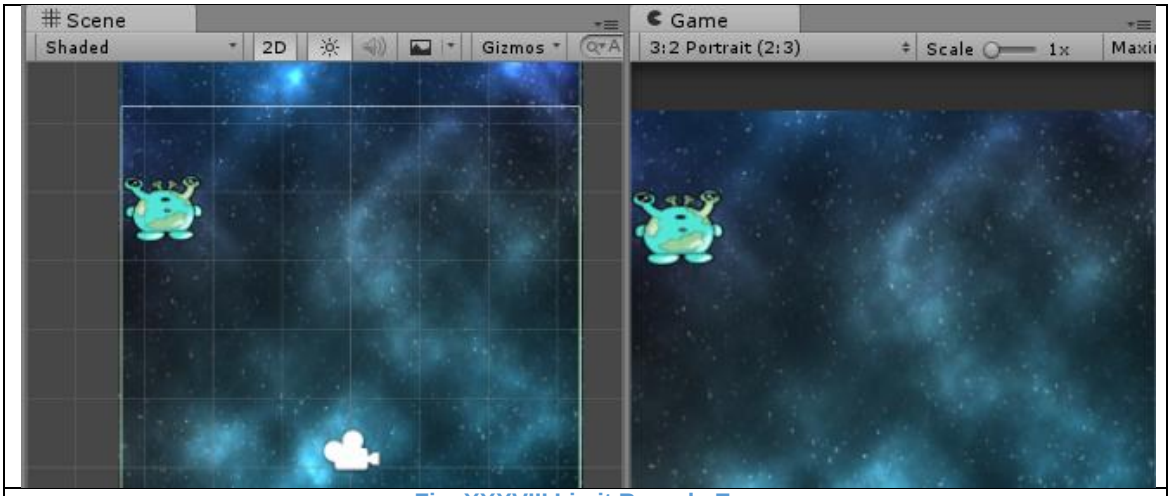


Fig. XXXVIII Limit Bounds Ew.

Com a fase final d'aquest script, vam afegir una última funció que s'encarregaria de modificar la velocitat de caiguda en cas que ens premessin el botó de eject. Si el botó es prem en la zona correcta, disminuirem la velocitat de caiguda a la meitat i indicarem que s'ha premut el botó en la zona correcta.

En canvi, si s'activa en una zona incorrecta, augmentarem la velocitat per 10 i finalment desactivarem el botó.

```
//this function is called if the eject button is clicked and will or not open the parachute
public void openParachute()
{
    //get the value to know if we are in the parachute zone
    correctParachuteZoneState = GameObject.Find("parachuteZone").GetComponent<parachuteZone>().correctParachuteZone;
    //if Ew is in the correct zone
    if (correctParachuteZoneState)
    {
        //reduce the velocity at the half
        ewMovementSpeedVertical = ewMovementSpeedVertical / 2;
        //the button has set in the correct zone
        buttonPatachuteClickedOnCorrectZone = true;
    }
    //if ew is not in the correct zone
    else
    {
        //increase the falling speed
        ewMovementSpeedVertical += 10;
    }
    //disable button eject
    GameObject.Find("btn_eject").SetActive(false);
}
```

Fig. XXXIX Script velocitat de caiguda prémer botó eject.

4.1.2.- Control de la càmera.

En aquest punt ja teníem implementat el moviment del personatge, però la càmera estava estàtica i no seguia el personatge en el seu descens. Per això es va crear el script CameraController. En aquest vam especificar la posició relativa en què volíem que es trobés l'Ew respecte la part superior de la càmera, concretament el offset Y, i li vam passar l'objecte del personatge per a poder extreure en quina posició es trobava de l'eix Y i d'aquesta forma, en la funció Update, anar modificant la posició de la càmera en cada frame.

A l'hora de fer proves, però, també va sorgir un problema; la càmera no es parava fins que l'Ew arribava a la part inferior del CameraBounds que havíem creat anteriorment i sortíem de la zona del fons de pantalla, tal com podem veure en la següent figura.

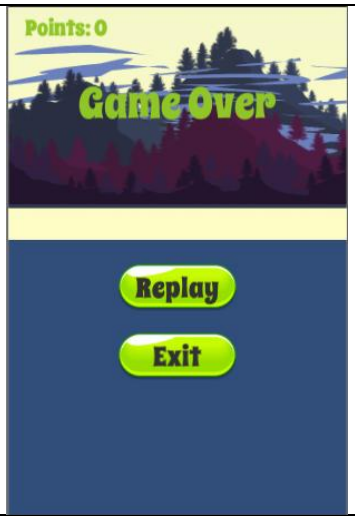


Fig. XL Error límit inferior càmera.

Per a solucionar-ho, vam aplicar un simple if en cada frame, on mirem en quin punt del món es troba la part inferior de la càmera, si aquesta és major que el

límit, vol dir que encara estem dins del fons de la pantalla i que per tant hem de moure la càmera. En canvi, si la posició de la part inferior de la càmera és menor que la zona inferior límit, ja no hem de moure més la càmera perquè hem arribat al límit. Tot seguit podem veure com queda després d'aplicar la solució i arribem al final de la pantalla.

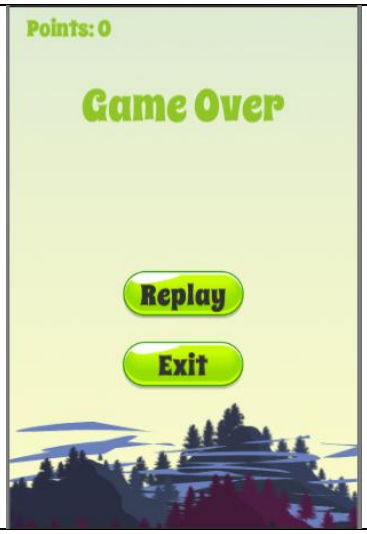


Fig. XLI Final moviment límit inferior càmera.

I finalment el codi que implementa tot el seguiment de la càmera.

```
//This script will control the camera to follow the character
public class CameraController : MonoBehaviour {

    //get the player (Ew) gameObject
    public Transform Player;
    Camera cam;
    //get the bounds zone to limit Ew movement
    public BoxCollider2D Bounds;
    //Two variables to get the minimum and maximum zone to move the player
    private Vector3
        _min,
        _max;

    //offset to set distance from the Ew and the top of the camera
    public float yOffset = -4.2f;
    //float with the ew size to correct ajust the distance
    float ewYsize = 0;

    void Start()
    {
        //get the minumum and maximum bounds size
        _min = Bounds.bounds.min;
        _max = Bounds.bounds.max;
        //get the current camera
        cam = GetComponent<Camera>();
        //get the Ew Y size
        ewYsize = Player.transform.localScale.y * 2;
    }

    //for every frame
    void Update()
    {
        //Next code will make the camera follow the character (EW) until it arribes at the final
        //FindObjectOfType the screen or the bonunds size Y
        if (!((Player.position.y - ewYsize - 0.1) <= Bounds.bounds.min.y))
        {
            //get the current Ew X position
            var x = transform.position.x;
            //calculate the camera half size
            var cameraHalfWidht = cam.orthographicSize * ((float)Screen.width / Screen.height);
            //ajust the x value to avoid exit from the camera
            x = Mathf.Clamp(x, _min.x + cameraHalfWidht, _max.x - cameraHalfWidht);
            //finaly, creates a new vector3 with the player position and the offset on the y axis
            //to avoid camera go out the screen
            transform.position = new Vector3(x, Player.position.y + yOffset, transform.position.z);
        }
    }
```

Fig. XLII Script CameraController.

4.2.- Creació, control i configuració d'objectes enemics.

En el següent apartat trobarem tot el que fa referència al control i generació d'enemics de forma aleatòria al llarg de la pantalla.

4.2.1.- Creació d'enemics (Enemy Spawner).

L'EnemySpawner és un objecte que tal com indica el seu nom, serveix per a crear o fer aparèixer enemics. En el nostre cas l'hem representat amb un rectangle a la part inferior de la càmera i l'hem posat com a fill d'aquesta perquè hereti el moviment creat en l'apartat anterior i puguem recórrer tot l'espai del joc.

A part d'aquest objecte, necessitàvem delimitar les zones on podien aparèixer cadascun dels enemics, tal com vam introduir amb anterioritat, primer han d'aparèixer els meteorits, després els satèl·lits, etc. Per això vam crear una sèrie d'objectes que contindrien un BoxCollider2D i que ens delimitarien les diferents zones. Gràcies a aquest col·lisionador i al de l'EnemySpawner podríem conèixer quan entrem i quan sortim de cadascuna de les zones i així poder seleccionar quin tipus d'objecte enemic hauríem de crear. Aquests objectes seran invisibles per l'usuari per tal que no molestin ni dificultin la jugabilitat. En la següent figura podem veure les diferents zones creades juntament amb el EnemySpawner; les hem identificat amb un icona per a facilitar la interpretació i validar el bon funcionament a l'hora de realitzar proves.

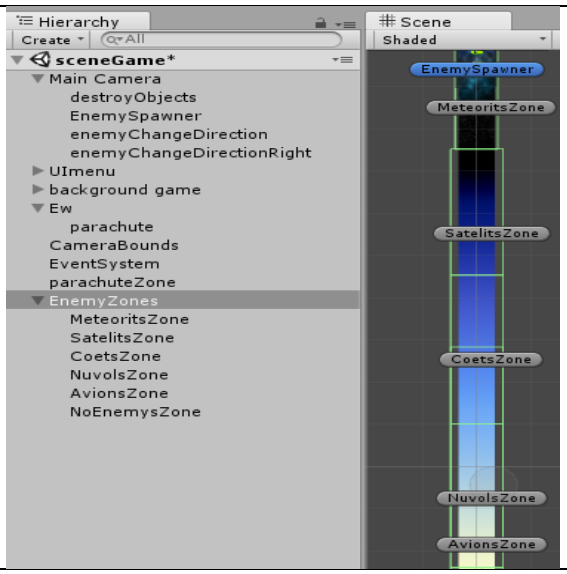


Fig. XLIII Enemy Spawner i zones d'objectes.

El següent pas era crear el script que, en funció de la zona i la dificultat del nivell, creés els enemics. Per això, vam fer el script enemySpawner que trobarem associat a l'objecte que rep el mateix nom. A més de crear els enemics, vam decidir aprofitar-lo i també crear les monedes i els núvols amb aquest mateix script, d'aquesta forma ens estalviaríem duplicitat d'objectes, codi i per tant, recursos.

Tot i que el codi ja es troba comentat, intentarem explicar les parts més rellevants del script. El primer que necessitàvem era una llista, dinàmica, amb els diferents enemics que podien aparèixer. Es va crear dinàmica perquè d'aquesta forma es podia estendre amb nous objectes sense modificar el codi. Tot seguit necessitàvem variables per contenir el nombre d'enemics a crear en cada zona, les velocitats mínimes i màximes que podrien tenir, i el mateix pel

nombre de monedes i de núvols. També necessitàvem tenir els límits on podien aparèixer els enemics, ja que no s'han de poder crear enemics fora de la pantalla, per això, si ho recordem utilitzàvem el CameraBounds, que ens donava els límits de la imatge de fons.

També s'havia de validar que el joc no hagués acabat per tal d'evitar la creació descontrolada d'enemics. Per a fer-ho, en la funció Update, que com ja hem indicat s'executa per cada frame, validem l'estat de les variables de finalització del joc de l'objecte GameManager, que com veurem, ens serveixen per controlar l'estat del joc entre d'altres coses. Si el joc no ha finalitzat, es criden tres funcions, la de creació d'enemics, la de creació de monedes i finalment si ens trobem a la zona de núvols, la de creació d'aquests. La següent figura ens mostra el codi de la funció Update anterior.

```
// Use this for initialization
void Start () {
    //get the bounds minumum and maximum zone
    _minBounds = Bounds.bounds.min;
    _maxBounds = Bounds.bounds.max;

    //set timer for enemies
    timerEnemics = delaySpawnTimer;
    //set random timer value for coins
    timerCoins = Random.Range(1, 10);
    //set random timer value for coins
    timetClouds = Random.Range(1, 10);
}

// Update is called once per frame
void Update () {
    //first we chack if game has finished for some reason, if not
    if (!GameObject.Find("Main Camera").GetComponent<gameManager>().gameOver &&
        !GameObject.Find("Main Camera").GetComponent<gameManager>().gameFinished &&
        !GameObject.Find("Main Camera").GetComponent<gameManager>().gameOverNoOpenParachute) {
        //check if we are in a no enemy zone, if not
        if (!noEnemyZone)
        {
            //Can create enemy and coins
            createEnemy();
            createCoins();
            //if we are in a clouds zone, we can create clouds
            if (cloudsZone)
            {
                createClouds();
            }
        }
    }

}
```

Fig. XLIV Script EnemySpawner.

Tot seguit, es van implementar les diferents funcions que crearien els objectes i que ja hem anunciat anteriorment. Tot i tenir una estructura molt semblant, cadascuna d'elles té petites diferències; per exemple la funció createEnemy primer de tot redueix el temps que ens indica si s'ha de crear un enemic o no, després comprovem si hem arribat al moment de crear un enemic. Tot seguit, si es pot crear un enemic, reiniciem aquest comptador, calculem la posició aleatòria on s'ha de crear el nou objecte, tenint en compte els límits màxims i mínims de l'eix X. Aquest càlcul es realitza amb la funció Random per tal de generar la posició aleatòriament. Finalment instanciem el nou objecte amb els paràmetres aleatoris creats anteriorment, el tipus d'enemic que havíem de crear segons la zona i especifiquem a aquest nou objecte la velocitat de moviment generada també aleatòriament amb els mínims i màxims de velocitat que podrà prendre.

En canvi, en el cas de les monedes, vam afegir una comprovació extra que controla el nombre de monedes a crear, ja que cada nivell tindrà un límit de monedes. Podem veure la representació del codi en la següent figura.

```
//this function lets create one enemy if timer is the correct
private void createEnemy()
{
    //reduce the timer
    timerEnemics -= Time.deltaTime;
    //if its 0 or les, a enemy has to be created
    if (timerEnemics <= 0)
    {
        //reset the timer
        timerEnemics = delaySpawnTimer;
        //generate a random position inside the bounds zone
        Vector3 randomPosition = new Vector3(Random.Range(_minBounds.x, _maxBounds.x), transform.position.y, transform.position.z);
        //create the game object (enemi)
        GameObject createdGameObject = Instantiate(enemies[idEnemyPerZone], randomPosition, transform.rotation);
        //set a random speed betwen min and max to move the enemy
        createdGameObject.GetComponent<enemyScript>().speed = Random.Range(enemyVelocityMin, enemyVelocityMax);
    }
}

//this function creates the coins
private void createCoins()
{
    ////reduce the timer for the next coin
    timerCoins -= Time.deltaTime;
    //if timmer is 0 and can create more coins
    if (timerCoins <= 0 && numCoins > 0)
    {
        //reduce the number of coins
        numCoins--;
        //generate a random timer for the next coin
        timerCoins = Random.Range(1, 10);
        //generate a random position inside the bounds zone
        Vector3 randomPosition = new Vector3(Random.Range(_minBounds.x, _maxBounds.x), transform.position.y, transform.position.z);
        //set a random speed betwen min and max to move the cloud
        Instantiate(coin, randomPosition, transform.rotation);
    }
}
```

Fig. XLV Script EnemySpawner part 2.

Per a poder controlar quin tipus d'enemic hem de crear, com ja vam introduir anteriorment, es realitzaria a partir de les col·lisions entre les diferents zones. Per exemple, quan el enemySpawner col·lisiona amb l'objecte MetroritsZone, indiquem el valor 0 a la variable idEnemyPerZone que ens indicarà que hem d'agafar l'objecte de la llista d'enemics que es troba en la posició 0. El mateix es realitza amb la resta d'objectes variant el id en funció de la zona en què entrem.

```
//function to determine the zone we are
void OnCollisionEnter2D(Collision2D collision)
{
    //if we are in a meteor zone
    if (collision.gameObject.name == "MeteoritsZone")
    {
        //set the enemy id list to 0
        idEnemyPerZone = 0;
    }
    //if we are in a satelit zone
    else if (collision.gameObject.name == "SatelitsZone")
    {
        //set the enemy id list to 1
        idEnemyPerZone = 1;
    }
    //if we are in a rocket zone
    else if (collision.gameObject.name == "CoetsZone")
    {
        //set the enemy id list to 2
        idEnemyPerZone = 2;
    }
    //if we are in a plane zone
    else if (collision.gameObject.name == "AvionsZone")
    {
        //set the enemy id list to 3
        idEnemyPerZone = 3;
    }
    //if we are in a clouds zone
    else if (collision.gameObject.name == "NuvolsZone")
    {
        //set clouds zone to true
        cloudsZone = true;
    }
```

Fig. XLVI Script enemySpawner part 3.

Ara ja teníem tot el sistema de creació aleatòria d'enemics, monedes o núvols en funció de la zona que ens trobem. La darrera cosa que ens quedava realitzar era informar els valors del script EnemuSpawner, tal com podem veure en la següent imatge. Especificar que els paràmetres NumEnemicsPerZona, EnemyVelocityMin, EnemyVelocityMax i NumCoins s'informaran des de l'objecte GameManager que veurem més endavant i variaran en funció del nivell en què ens trobem.

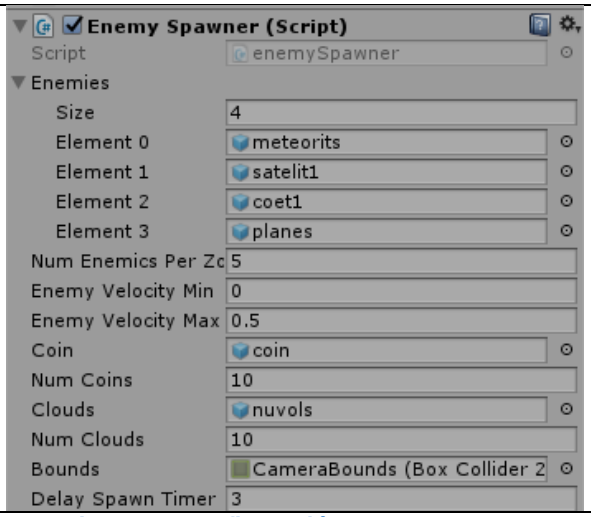


Fig. XLVII Configuració EnemySpawner.

4.2.2.- Control i moviment d'enemics

En aquest punt ja havíem aconseguit crear els objectes enemics durant el descens de l'Ew per la pantalla i en posicions aleatòries, però aquests no tenien moviment ni canviaven d'imatge en funció dels dissenys que havíem realitzat, per exemple, només apareixien coets de color blau; per això vam crear els scripts EnemyScript, que controlarà el moviment i el RandomEnemyImage, que s'encarregarà de la imatge de l'enemic.

El primer ens genera, aleatòriament, la direcció que ha de prendre l'objecte a partir d'un valor entre -1 i 2. Si el valor és inferior o igual a 0, el moviment serà negatiu en l'eix de les X, per tant anirà cap a l'esquerra. En canvi, si el valor és major que 0, el moviment serà positiu i anirà cap a la dreta. El següent pas era assignar la velocitat i direcció del moviment a l'objecte, això ho realitzem en la funció Update per si s'ha de modificar la velocitat i direcció de l'objecte en temps d'execució.

Quan vam realitzar les proves, vam veure que els objectes enemics es movien de la forma esperada, però aquests sortien del plànol de la càmera. Això era perquè no es limitava la zona de moviment, tal com havíem fet en altres objectes. En aquest cas, com que ja sabíem que els enemics sempre es crearien dins del CameraBounds vam preferir no utilitzar-lo per a limitar la zona de moviment. En comptes d'això, es van crear dos objectes nous, anomenats EnemyChangeDirection, que es troben situats a la dreta i esquerra de la càmera, per a detectar la col·lisió amb aquests i girar el moviment multiplicant per -1 la direcció. D'aquesta forma aconseguíem invertir la velocitat de l'objecte.

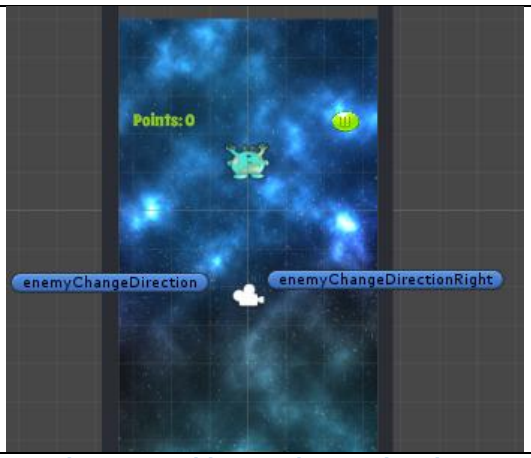


Fig. XLVIII Objectes changeDirection.

Gràcies a això, vam veure que la imatge de l'objecte enemic sempre tenia la mateixa direcció, independentment del seu moviment. Per això, dins de la funció Update, controlem el sentit de l'objecte i invertim la imatge en funció de la direcció que té. Tot seguit podem veure la figura amb el codi que realitza totes aquestes accions i controls.

enemyScript.cs

EAW.CSharp | enemyScript

```
11      private SpriteRenderer mySpriteRenderer;
12
13      private void Awake()
14      {
15          // get a reference to the SpriteRenderer component on this gameObject
16          mySpriteRenderer = GetComponent<SpriteRenderer>();
17      }
18
19      // Use this for initialization
20      void Start () {
21          //Generate a random value to chose the direction, if is les than 0,
22          //will go to left, default is 1 and goes to right
23          if (Random.Range(-1, 2) <=0)
24          {
25              //change default direction
26              direction = -1;
27              //flip the image to change the direcion
28              mySpriteRenderer.flipX = false;
29          }
30      }
31
32      // Update is called once per frame
33      void Update () {
34          //this sets the speed of the object depending of the direction, speed and tim per frame
35          transform.Translate(new Vector3(direction, 0, 0) * speed  * Time.deltaTime);
36          //if sirection has changed
37          if (direction < 0)
38          {
39              //flip the sprite image
40              mySpriteRenderer.flipX = false;
41          }
42          else
43          {
44              //normalize the sprite image
45              mySpriteRenderer.flipX = true;
46          }
47      }
48
49      //This function is called if someone collides with the object
50      void OnCollisionEnter2D(Collision2D colission)
51      {
52          //if the object tat collission is a enemy change diraction with tag "changeDirection"
53          if(colission.gameObject.tag == "changeDirection")
54          {
55              //we change the movement direction by multiply per -1
56              direction = direction * -1;
57          }
58      }
```

Fig. XLIX Script enemyScript.

Ara però, ens faltava modificar la imatge dels diferents dissenys dels enemics i que es realitza des del RandomEnemyImage. Aquest script calcula el nombre d'imatges possibles que tenim, per exemple en el cas dels coets 3 i genera un

nombre aleatori per obtenir quina imatge triar. Aquesta llista d'imatges també l'hem generat de forma dinàmica per si mai es creen nous dissenys, no haver de modificar el codi.

Tot això, però, feia que el BoxCollider2D de l'objecte generat sempre tingués la mateixa mida, independentment de la mida real de l'enemic un cop creat, per tant, hi havia zones més grans de col·lisió que el propi objecte. Això es va corregir calculant la mida real de la imatge creada i aplicant-la al BoxCollider2D. En el següent codi podem veure la implementació.

```
//script to randomize the image of the enemy created and ajust box collider to size of the image
public class randomEnemyImage : MonoBehaviour {

    //list of the enemy images
    public Sprite[] enemySprites;
    //percent to reduce box collider, default 10%
    float reduceBoxColliderSizePercent = 10;

    // Use this for initialization
    void Start () {
        //generate a random value from 0 to lenght of the enemy sprites defined
        int randomSpriteEnemyImage = Random.Range(0, enemySprites.Length);
        //set the sprite selected
        gameObject.GetComponent<SpriteRenderer>().sprite = enemySprites[randomSpriteEnemyImage];

        //As every sprite has a independent size, we have to ajust boxcolider
        //to sprite image size. We get the sprite size
        Vector2 S = gameObject.GetComponent<SpriteRenderer>().sprite.bounds.size;
        //Reduce percent of the size
        S.x = calculatePercent(S.x);
        S.y = calculatePercent(S.y);
        //and set values to the box collider
        gameObject.GetComponent<BoxCollider2D>().size = S;
    }

    //function to calculate the percent of reduced size for the box colliders
    private float calculatePercent(float currentSize)
    {
        return currentSize - (currentSize * reduceBoxColliderSizePercent / 100);
    }

}
```

Fig. L Script RandomEnemyImage.

Finalment podem veure com queda la configuració d'un dels objectes, en aquest cas el coet. En la figura, veurem que hi ha un paràmetre anomenat Speed, dins del EnemyScript, i que conté la velocitat de moviment de l'enemic. Recordem que aquest valor és informat des del EnemySpawner de forma aleatòria un cop s'ha creat l'objecte.

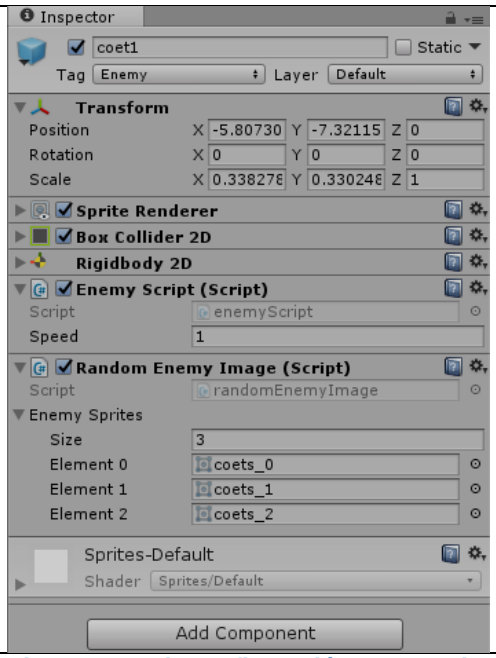


Fig. LI Exemple configuració coet enemic.

4.2.3.- Control de monedes.

Tot i no considerar-se un enemic, i com ja hem introduït anteriorment, les monedes es creen des del enemySpawner per tal d'evitar duplicitat del codi. El que sí que és diferent és el control de la pròpia moneda, ja que aquesta ens ha de reportar punts en comptes de fer perdre la partida, a banda de controlar el moviment de l'animació. Per això es va crear el script coinController. Aquest, bàsicament, té dues funcions, la primera és la Start, on s'inicialitzen el objecte, però a més, fem que l'animació de la moneda comenci en un instant de temps aleatori, gràcies a la funció Random. Ho fem així perquè sinó, totes les monedes que es crearien, girarien en el mateix ordre i de la mateixa forma. En canvi, amb aquesta simple funció, podem indicar en quin instant de l'animació comença la rotació i per tant totes les monedes giraran diferent.

La segona funció que tenim és la de floatingTextPoints. Aquesta, està pensada per a ocultar la moneda un cop l'Ew hi col·lisiona i tot seguit generar un petit text flotant indicant el valor dels punts aconseguits. Amb això volíem un disseny més atractiu i no tan estàtic. Podem veure la implementació en la següent figura.

```
using UnityEngine;

//this script controls the coin
public class coinController : MonoBehaviour {

    void Start () {
        //when a coin is created, we have to play the animation randomly to avoid all coins move equals
        Animator animator = GetComponent<Animator>();
        animator.Play("RotationCoin", 0, Random.Range(0.0f, 1.0f));
    }

    //this function will create the floating text with the points achived to collision with a coin
    public void floatingTextPoints(int points)
    {
        //first we disable the game object
        gameObject.SetActive(false);
        //then we create and set the floating text with the points al te coin position
        FloatingText.Show(string.Format("+{0}!", points), "PointCoinText",
        new FromWorldPointTextPositioner(Camera.main, transform.position, 1.5f, 50));
        //when the text has been displayed, we destroy the gameoject to reduce resources
        Destroy(gameObject);
    }
}
```

Fig. LII Script CoinController.

La generació d'aquest text flotant es fa a partir del script FloatingText, que ens calcula la posició actual de l'objecte, en aquest cas la moneda, genera el text en aquesta mateixa posició, i finalment el fa moure, amb una velocitat especificada cap amunt, durant el temps que li hem indicat; finalment es fa desaparèixer el text i es pot eliminar l'objecte.

```
public class FloatingText : MonoBehaviour {

    //gets the configuration af gameskin
    private static readonly GUISkin skin = Resources.Load<GUISkin>("GameSkin");

    //creates the floating text game object
    public static FloatingText Show(string text, string style, IFloatingTextPositioner positioner)
    {
        var go = new GameObject("Floating Text");
        var floatingText = go.AddComponent<FloatingText>();
        floatingText.Style = skin.GetStyle(style);
        floatingText._positioner = positioner;
        floatingText._content = new GUIContent(text);
        return floatingText;
    }

    private GUIContent _content;
    private IFloatingTextPositioner _positioner;

    public string Text { get { return _content.text; } set { _content.text = value; } }
    public GUIStyle Style { get; set; }

    //prints the text during the time and when finished destrois the text created
    public void OnGUI()
    {
        var position = new Vector2();
        var contentSize = Style.CalcSize(_content);
        if (!_positioner.GetPosition(ref position, _content, contentSize))
        {
            Destroy(gameObject);
            return;
        }
        GUI.Label(new Rect(position.x, position.y, contentSize.x, contentSize.y), _content, Style);
    }
}
```

Fig. LIII Script FloatingText.

En la següent figura, podrem veure un exemple, just després de col·lisionar amb una moneda, de com es genera el text amb la puntuació indicada.

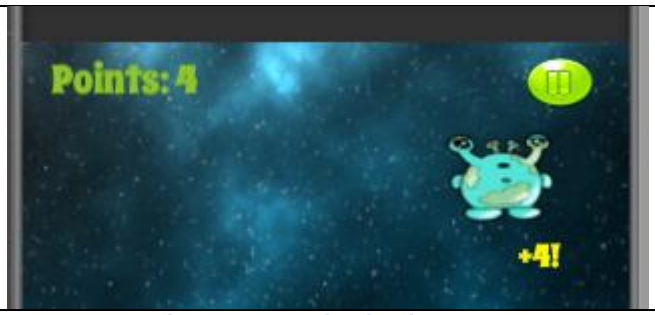
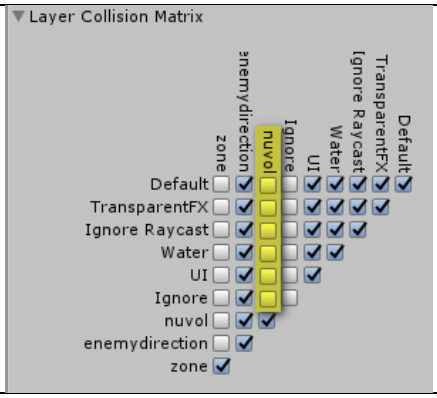


Fig. LIV Exemple FloatingText.

4.2.4.- Control dels núvols.

Els núvols són els últims objectes que es poden crear en el joc. Per evitar tenir codi redundant, es va decidir aprofitar els scripts de control d'enemics per a realitzar el moviment i selecció d'imatges d'aquest. La única diferència en els núvols respecte als enemics és que necessiten una zona d'aparició diferent i que les col·lisions no han d'afectar a altres objectes. Pel primer cas es va definir una nova zona anomenada NuvolsZone, i que es controlarà des del EnemySpawner. Pel segon cas, Unity té una funcionalitat anomenada LayerCollisionMatrix que permet indicar quins objectes poden col·lisionar amb altres. Gràcies a aquesta funcionalitat, simplement desactivant les col·lisions, podíem evitar que l'Ew o els enemics es veiessin afectats per aquests, la única cosa que sí que havia d'afectar als núvols són els objectes de control. Tot seguit veiem la configuració realitzada.



4.2.5.- Destroy Objects.

Per a poder mantenir un joc net i optimitzat, es va crear un nou controlador que ens permetia eliminar objectes un cop haguessin sortit de la zona de la càmera. Seguint els principis anteriors, es va crear un objecte dins de la càmera i situat en la part superior d'aquesta, que s'encarregaria d'anar col·lisionant amb els diferents objectes i els destruiria. D'aquesta forma, un cop ja no el veiem, no el mantenim creat i reduïm els recursos utilitzats pel dispositiu. El script que es va crear s'anomena enemyDestroyerScript i el podem veure a continuació.

```
//Script to destroy object with collission (used for destroy enemys)
public class enemyDestroyerScript : MonoBehaviour {

    //is som object collides with the object that has this function is triggered
    void OnCollisionEnter2D(Collision2D colission)
    {
        //and destroys the object recibed
        Destroy(colission.gameObject);
    }
}
```

Fig. LV Script EnemyDestroyerScript.

Com en els casos anteriors, es tracta d'un rectangle que anirà seguint el moviment de la càmera. Podem veure la seva representació en la següent figura.

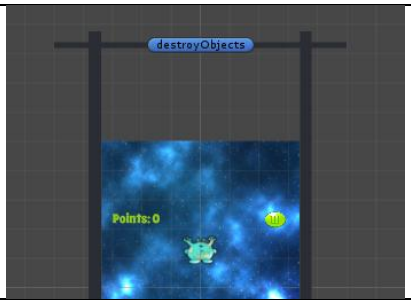


Fig. LVI Objecte DestroyObjects.

4.3.- Zona d'obertura del paracaigudes.

La zona d'obertura del paracaigudes, és el tros del món on hem de prémer el botó eject, per tal de disminuir la velocitat i així aconseguir un aterratge segur. Aquesta zona està situada en la part baixa del fons, però volíem poder modificar la dificultat del joc en funció del nivell i vam decidir reduir-la en funció del nivell en què ens trobéssim.

Per a poder controlar aquests canvis es va crear el script parachuteZone. Aquest té una variable pública i que serà modificada des del GameManager, fet que disminueix l'extensió vertical del BoxCollides2D utilitzat per a delimitar la zona. Aquest script també es va aprofitar per a controlar l'entrada i la sortida de la zona gràcies a les funcions OnCollisionEnter2D i OnCollisionExit2D que es disparen de forma automàtica, així podem saber si entrem o sortim. Per a poder facilitar una mica el joc, en aquestes funcions, també es canvia el color del botó eject, per tal d'indicar en quin moment hem d'obrir el paracaigudes.

```
//This function is used on a parachuteZone to controll the zone where the parachute must be open
public class parachuteZone : MonoBehaviour
{
    //variables and objects to control parachute zone
    GameObject buttonEject;
    public Sprite activeButton;
    public Sprite inactiveButton;
    public float sizeParachuteZone;
    //this boolean lets indicate if we are in the correct zone
    public bool correctParachuteZone = false;

    void Start()
    {
        //get the button eject object
        buttonEject = GameObject.Find("btn_eject");
        //set the image button as innactivate
        buttonEject.GetComponent<Image>().sprite = inactiveButton;
        //change the default size zone in function of the level we are
        transform.GetComponent<BoxCollider2D>().size = new Vector2(transform.GetComponent<BoxCollider2D>().size.x, sizeParachuteZone);
    }

    //this function controlls if EW enter in the parachute zone
    void OnCollisionEnter2D(Collision2D collision)
    {
        //change the image button to indicate Ew is in the correct zone
        buttonEject.GetComponent<Image>().sprite = activeButton;
        //and indicates that we are in the correct zone
        correctParachuteZone = true;
    }

    //this function controlls if EW exit from the parachute zone
    void OnCollisionExit2D(Collision2D collision)
    {
        //change the button image to indicate that Ew is not in the correct zone
        buttonEject.GetComponent<Image>().sprite = inactiveButton;
        //and indicates that we are not in the correct zone
        correctParachuteZone = false;
    }
```

Fig. LVII Script ParachuteZone.

En la següent figura podem observar la zona que ens delimita l'obertura del paracaigudes.

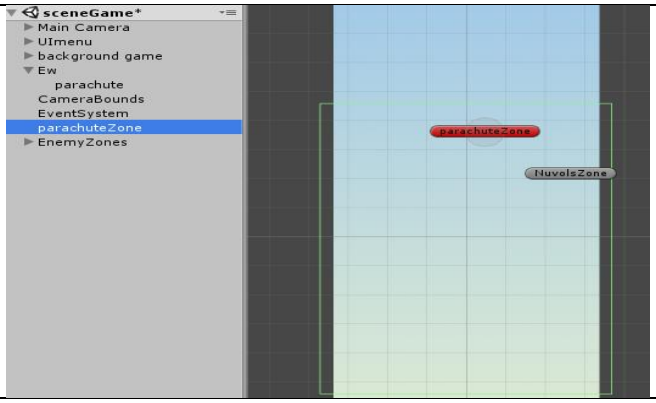


Fig. LVIII Zona d'obertura del paracaigudes.

Finalment només ens quedava generar el script associat al paracaigudes, perquè un cop ens premessin el botó, si ens trobàvem en la zona correcta, fos visible per a l'usuari.

```
//This script lets us control the parachute
public class parachuteScript : MonoBehaviour {

    //By default, the parachute is disabled
    void Awake () {
        gameObject.SetActive(false);
    }

    //this function enables the parachute if we are in the correct zone
    public void openParachute()
    {
        //check if the parachute zone is the correct
        if (GameObject.Find("parachuteZone").GetComponent<parachuteZone>().correctParachuteZone)
        {
            //activates the parachute object
            gameObject.SetActive(true);
        }

    }

}
```

Fig. LIX Script ParachuteScript.

4.4.- Sistema i control de col·lisions.

Un dels últims passos en la creació del joc era controlar les col·lisions amb els diferents objectes que ens podríem trobar. Aquest controlador, hauria d'estar situat en l'Ew, ja que depenent de contra què col·lisionéssim, hauríem de realitzar una acció o una altra. Per a fer-ho vam crear un sistema d'etiquetes per a distingir els diferents objectes contra els que podíem col·lisionar. Així doncs, teníem els objectes amb el tag Enemy, Points i Mountains. En la següent figura podem veure les tres etiquetes representades.

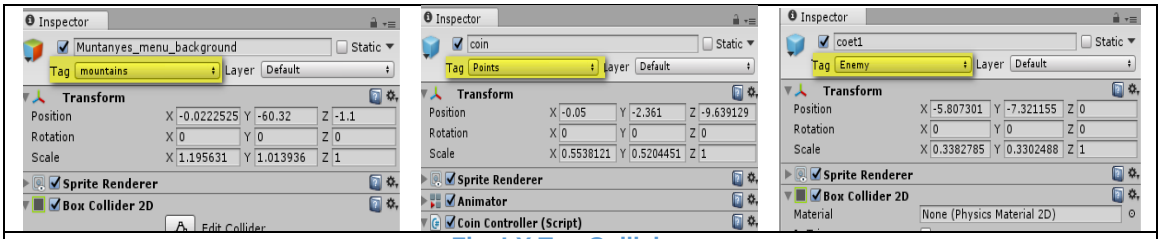


Fig. LX Tag Collisions.

Gràcies a aquest sistema d'etiquetes, podem identificar el tipus d'objecte que es tracta i en cas de crear-ne de nous, per exemple un enemic que fos un ocell, simplement identificant-los amb aquesta etiqueta, el joc sabria quin comportament ha de tenir.

Però únicament amb aquests tags no controlem el joc; per a fer-ho necessitarem el script EwCollisionController. En aquest, utilitzarem la funció OnCollisionEnter2D on rebrem l'objecte amb qui l'Ew ha col·lisionat i extraurem el tag corresponent. En el cas dels enemics, indicarem que hem finalitzat la partida per col·lisió i eliminarem l'Ew. En el cas dels punts, generarem de forma aleatòria el valor d'aquests i li passarem aquest valor al CoinController vist anteriorment perquè generi el text flotant i s'actualitzarà el text amb els punts que tenim a la pantalla. Per últim, en el cas de col·lisionar amb les muntanyes, s'indica que s'ha arribat al final de la pantalla, per tant hem de comprovar si ens han obert el paracaigudes o no per a conèixer si el joc ha finalitzat correctament. Finalment passem els punts aconseguits en aquesta partida al GameManager perquè els guardi en memòria. El codi del controlador de col·lisions de l'Ew és el següent.

```
//if some object collision with Ew will trigger this function and recive the collisioned object
void OnCollisionEnter2D(Collision2D colission)
{
    //if Ew collission with object tagged with Enemy name
    if (colission.gameObject.tag == "Enemy")
    {
        //set gameover by collission
        gameOverByCollision = true;
        //set to game manager a gameover to true
        GameObject.Find("Main Camera").GetComponent<gameManager>().gameOver = true;
        //disable ew object
        gameObject.SetActive(false);
    }
    //if Ew collission is with a object tagged with points, will score points
    else if (colission.gameObject.tag == "Points")
    {
        //generate a random value with de numer of points
        int numPointsToAdd = Random.Range(minPoints, maxPoints);
        //call the function "floatingTextPoints" to create a floating text with the number of points
        colission.gameObject.GetComponent<coinController>().floatingTextPoints(numPointsToAdd);
        //increment the number of points achived in this game
        score += numPointsToAdd;
        //update the score text
        scoreText.text = "Points: " + score;
    }
    //if Ew collission is with a object tagged "mountains" it means we arribed at bottom
    else if (colission.gameObject.tag == "mountains")
    {
        //check if the button eject has clicked in the correct zone
        if (transform.gameObject.GetComponent<characterMovementController>().buttonPatachuteClickedOnCorrectZone)
        {
            //it menas the game finished correclty
            GameObject.Find("Main Camera").GetComponent<gameManager>().gameFinished = true;
        }else
        //if the button has not been clicked or clicked in the wrong zone
        {
            //we set a gameover by no parachute open or open in wrong zone
            GameObject.Find("Main Camera").GetComponent<gameManager>().gameOverNoOpenParachute = true;
        }
        //disable Ew object
        gameObject.SetActive(false);
    }
    //finaly, we pass the score achived in this match to the game manager
    GameObject.Find("Main Camera").GetComponent<gameManager>().playedPoints = score;
}
```

Fig. LXI Script ewCollisionController.

4.5.- Menús, texts i botons en el joc.

Un dels últims passos que ens faltava generar era el control de botons i menús que trobaríem en el joc, per exemple per fer pausa, reprendre la partida, reiniciar-la, sortir al menú principal o mostrar que hem guanyat o perdut la partida, la interfície havia de canviar i sortir textos o botons concrets. Per tot això es va crear un script anomenat buttonGamemanager, que controla tots aquests objectes des d'un únic punt. El primer pas va ser crear les variables per emmagatzemar-los. Tot seguit, desactivàvem tots aquells objectes que no es fessin servir durant el transcurs de la caiguda, per exemple, el botó de retry, o el text de joc perdut, etc.

```
//script to control all the buttons from the game
public class buttonGameManager : MonoBehaviour {

    //Sprite variable object to change images
    public Sprite pauseGame;
    public Sprite playGame;
    //variables with the game object buttons
    public GameObject playPauseIcon;
    public GameObject exitButton;
    public GameObject replayButton;
    public GameObject gameOverText;
    public GameObject levelComplete;
    public GameObject levelText;
    public GameObject completedPointsText;
    public GameObject ejectButton;

    private void Awake()
    {
        //set the pause sprite
        playPauseIcon.GetComponent<Image>().sprite = pauseGame;
        //disable all buttons don't neded to play the game
        exitButton.SetActive(false);
        replayButton.SetActive(false);
        gameOverText.SetActive(false);
        levelComplete.SetActive(false);
    }
```

Fig. LXII Script ButtonGamemanager part 1.

Un cop ja teníem la interfície neta, vam crear una funció que ens permetés parar o continuar el joc quan ens premessin el botó de pausa situat a la part superior dreta. Aquesta funció és la stopGame i para o encén el moviment de tots els objectes modificant el valor del temps de 1 a 0, així s'aconsegueix que no hi hagi moviment de cap objecte. A més aquesta funció activa o desactiva els botons del menú de pausa.

```
//function to stop or start the game. Called from pause button
public void stopGame()
{
    //if game is running
    if (Time.timeScale == 1)
    {
        //pause the game
        Time.timeScale = 0;
        //change the icon
        playPauseIcon.GetComponent<Image>().sprite = playGame;
        //set buttons visible
        exitButton.SetActive(true);
        replayButton.SetActive(true);
    }
    //if game is paused
    else if (Time.timeScale == 0)
    {
        //continue the game
        Time.timeScale = 1;
        //change icon to pause
        playPauseIcon.GetComponent<Image>().sprite = pauseGame;
        //disable buttons
        exitButton.SetActive(false);
        replayButton.SetActive(false);
    }
}
```

Fig. LXIII Script ButtonGamemanager part 2.

Per últim ens quedava una funció per a poder tornar al menú principal, i una altra que anés comprovant l'estat de la partida i en cas de guanyar o perdre, activés el menú corresponent. Podem veure com aquest últim control el fem en la funció update, ja que ens interessa que es vagi comprovant constantment si la partida ha acabat o no.

```
//function to load scene by name when it's paused on the menu
public void loadScene(string sceneName)
{
    Time.timeScale = 1;
    SceneManager.LoadScene(sceneName, LoadSceneMode.Single);
}

//called every frame
void Update()
{
    //if the game finished as ame over
    if (gameObject.GetComponent<gameManager>().gameOver || gameObject.GetComponent<gameManager>().gameOverNoOpenParachute)
    {
        //enable or disable some buttons
        ejectButton.SetActive(false);
        playPauseIcon.SetActive(false);
        exitButton.SetActive(true);
        replayButton.SetActive(true);
        gameOverText.SetActive(true);
    }else
    //if the game finished ok
    if (gameObject.GetComponent<gameManager>().gameFinished)
    {
        //enable or disable some buttons
        ejectButton.SetActive(false);
        playPauseIcon.SetActive(false);
        exitButton.SetActive(true);
        replayButton.SetActive(true);
        levelComplete.SetActive(true);
    }

}
```

Fig. LXIV Script ButtonGamemanager part 3.

Les següents figures que mostrarem tenen una representació de cadascun dels estats que tindrà el joc.

Fig. LXV Joc i menú pausa.

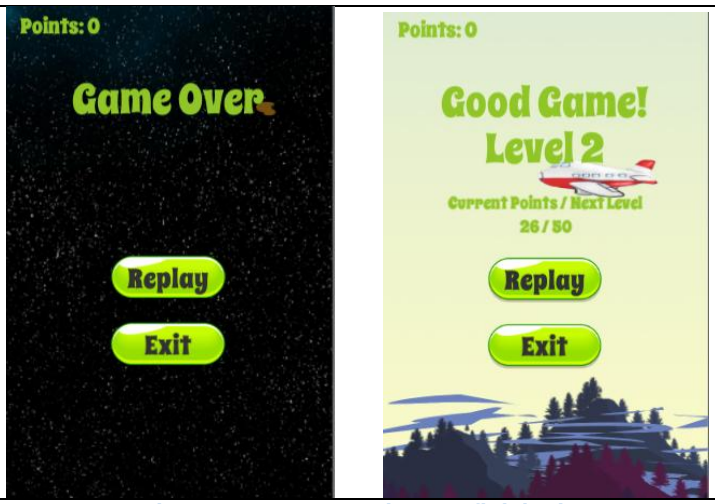


Fig. LXVI Game Over i Game finished.

4.6.- Game Manager.

L'últim sistema o eina que hem anat anunciant al llarg d'aquest capítol i que hem implementat és el GameManager. Aquest script, que l'hem associat a la càmera simplement perquè aquesta mai es destrueix, és l'encarregat de controlar i assignar tots els valors a les variables del joc en funció del nivell en què ens trobem i emmagatzemar l'estat de les partides, els punts aconseguits i el nivell del jugador.

Primer de tot, explicar que volíem aconseguir un sistema de nivells dinàmic, fàcil d'administrar i que es pogués incrementar sense modificar codi. Això ens permetria dedicar temps a altres tasques més importants i millorar el joc en un futur.

Per a fer-ho, primer de tot vam crear totes les variables que necessitaríem, però especialment vam crear una classe, anomenada Level, que contindria totes les variables que conformarien un nivell; per exemple la velocitat dels enemics, el nombre de monedes que haurien d'aparèixer, la velocitat de caiguda de l'Ew. Tot seguit definiríem una llista de classes Level. Gràcies a això, podríem tenir tants nivells com volguéssim. Tot seguit mostrem totes les variables i la classe anterior creada.

```
//This script lets control the game and create new level configurations and set values to all system variables
public class gameManager : MonoBehaviour {

    //Object that stores actual level
    public Level currentLevel;
    //List with all levels created. We can create all level configurations we want
    public List<Level> levelList = new List<Level>();
    //Stores current level
    int currentPlayerLevel = 0;

    //Variables to control game state
    public bool gameOver = false;
    public bool gameOverNoOpenParachute = false;
    public bool gameFinished = false;

    //variable to control stored and actual points
    int playerStoredPoints = 0;
    public int playedPoints = 0;

    //variables to store screen texts
    public Text pointsText;
    public Text pointsFinishText;
    public Text levelText;

    //Level data structure class
    [System.Serializable]
    public class Level
    {
        public float verticalEwSpeed;
        public int numCoins;
        public int numEnemicsPerZona;
        public float enemyVelocityMin;
        public float enemyVelocityMax;
        public float delaySpawnTimer;
        public int levelPoints;
        public float parachuteZoneSize;
    }
```

Fig. LXVII Script GameManager part 1.

El segon pas seria, just en el moment d'arrencar el joc, recuperar el nivell que té el jugador. Si és el primer cop, assignem el valor 0 com a defecte. Tot seguit recuperarem els valors de configuració del nivell i els assignarem a totes les variables de tots els scripts creats l'apartat 4. Amb aquest sistema tenim una forma còmode i fàcil de modificar els valors de configuració de cada objecte.

```
void Awake()
{
    //Recover stored player points
    playerStoredPoints = PlayerPrefs.GetInt("PlayerScore",0);
    //Set screen never sleep to avoid phone lock
    Screen.sleepTimeout = SleepTimeout.NeverSleep;
    //Get current player level
    currentPlayerLevel = PlayerPrefs.GetInt("PlayerLevel", 0);
    //Load current level configuracion
    currentLevel = levelList[currentPlayerLevel];
    //Call function to set all level variables by configuration
    setAllLevelValuesConfiguration();
}

//This function sets the actual configuration to all the variable game objects by this we can modify
//easily all the game variables
private void setAllLevelValuesConfiguration()
{
    //Change Ew vertical speed
    GameObject.Find("Ew").GetComponent<characterMovementController>().ewMovementSpeedVertical = currentLevel.verticalEwSpeed;
    //Set maximum number of coins to apear in the game
    GameObject.Find("EnemySpawner").GetComponent<enemySpawner>().numCoins = currentLevel.numCoins;
    //Set maximum number of enemies to apear in the game
    GameObject.Find("EnemySpawner").GetComponent<enemySpawner>().numEnemicsPerZona = currentLevel.numEnemicsPerZona;
    //Set minimum speed of enemies movement
    GameObject.Find("EnemySpawner").GetComponent<enemySpawner>().enemyVelocityMin = currentLevel.enemyVelocityMin;
    //Set maximum speed of enemies movement
    GameObject.Find("EnemySpawner").GetComponent<enemySpawner>().enemyVelocityMax = currentLevel.enemyVelocityMax;
    //Set delay time betwen enemy spawns
    GameObject.Find("EnemySpawner").GetComponent<enemySpawner>().delaySpawnTimer = currentLevel.delaySpawnTimer;
    //Sets the size of a parachute zone
    GameObject.Find("parachuteZone").GetComponent<parachuteZone>().sizeParachuteZone = currentLevel.parachuteZoneSize;
}
```

Fig. LXVIII Script GameManager part 2.

El GameManager també ha de controlar si ha finalitzat o no el joc per tal de mostrar els texts amb els punts actualitzats, guardar-los i comprovar si el

jugador ha pujat de nivell. Aquesta és la última part del script i que podem veure a continuació.

```
void Update()
{
    //Check if game has finished
    if (gameOver || gameOverNoOpenParachute || gameFinished)
    {
        //Function to control text score
        StartCoroutine(decreaseScore());
        //Change level text to current level
        levelText.text = "Level " + currentPlayerLevel;
        //set text to actual points
        pointsFinishText.text = "Current Points / Next Level \n"+playerStoredPoints + " / " + currentLevel.levelPoints;
    }
}

//this function decreses progresively the value of the points
IEnumerator decreaseScore()
{
    while (playedPoints >= 1)
    {
        playedPoints = playedPoints - 1;
        pointsText.text = "Points: " + playedPoints;
        if (gameFinished)
        {
            playerStoredPoints += 1;
            //Store the value of the points if game finished correctly
            PlayerPrefs.SetInt("PlayerScore", playerStoredPoints);
            checkLevel();
        }
        yield return new WaitForSeconds(2);
    }
}

//Check if the player increases the level
private void checkLevel()
{
    if (currentLevel.levelPoints < playerStoredPoints)
    {
        if (currentPlayerLevel < levelList.Count) {
            currentPlayerLevel += 1;
            currentLevel = levelList[currentPlayerLevel];
            PlayerPrefs.SetInt("PlayerLevel", currentPlayerLevel);
        }
    }
```

Fig. LXIX Script GameManager part 3.

Amb tot això ja podem modificar, de forma dinàmica, els nivells del joc i controlar i emmagatzemar l'evolució del jugador, sense haver de realitzar configuracions o scripts específics per a cadascun dels nivells. L'últim pas és el de crear i informar la llista de variables de la classe Level, que podem veure en la següent figura.

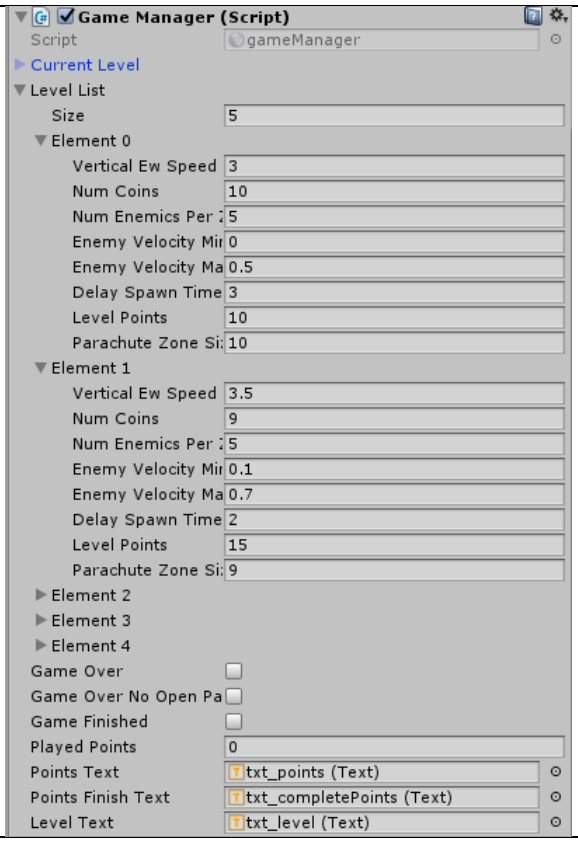


Fig. LXX Configuració nivells.

Com podem veure, la llista està formada per 6 nivells. Si augmentéssim aquest valor, automàticament es generaria un nou element de nivell i podríem informar, tal com hem fet en el Element0, on hem assignat tots els valors a totes les variables de la classe i que quan s'inicií el joc, s'informarà als respectius scripts de forma automàtica.

5.- Anàlisi, detecció i correcció d'errors.

5.1. El personatge surt de la pantalla.

Un cop teníem l'aplicació compilada i la vam poder provar en diferents dispositius, vam veure que en aquells que tenien la pantalla més petita, part del personatge podia sortir pels costats de la pantalla. No sortia del tot perquè el CameraBounds ens limitava el moviment, però més de la meitat del personatge quedava fora. Investigant quin era el problema vam veure que amb pantalles més petites, la zona horitzontal de moviment havia de ser més reduïda que la indicada pel CameraBounds, ja que la càmera era més petita.

La solució que vam aplicar va ser la d'afegir un script anomenat CameraBoundsSize, en l'objecte que rep el mateix nom, que calcula quina és la mida horitzontal exacte de la càmera i modifica aquest paràmetre del CameraBounds. Gràcies a aquesta petita operació, forcem als dos objectes a tenir la mateixa mida, horitzontal, independentment del dispositiu que estiguem utilitzant. Tot seguit veiem la implementació realitzada.

```
using UnityEngine;

//this script controls bounds where the character can move it will ajust the bounds
//depending on the camera size
public class cameraBoundsSize : MonoBehaviour {

    //get the camera gameobject to calculate it's wight depending the phone/tablet size
    public new GameObject camera;

    // Use this for initialization
    void Start () {
        //get the camera object
        Camera cam = camera.GetComponent<Camera>();
        //calculate the height of the camera
        float height = 2f * cam.orthographicSize;
        //with the height calculate the width
        float width = height * cam.aspect;
        //Finaly change the box collider size (x or width) to limit character movements
        GetComponent<BoxCollider2D>().size = new Vector2(width, GetComponent<BoxCollider2D>().size.y);
    }

}
```

Fig. LXXI Script CameraBoundsSize.

I finalment mostrem una figura on es pot visualitzar el problema que teníem.

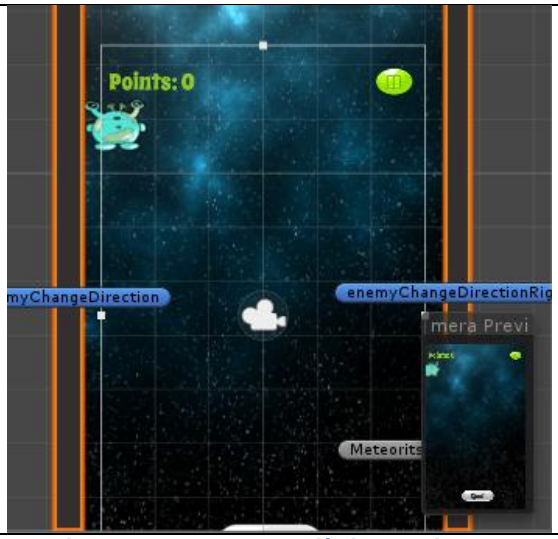


Fig. LXXII Mostra error límits moviment.

5.2.- Reducció box collider enemics.

El segon problema que es va detectar un cop ja teníem el joc operatiu, va ser que tot i haver ajustat la mida dels BoxCollider2D dels objectes enemics, aquests seguien sent massa grans, o en casos concrets com en els coets que estan situats en diagonal, el contenidor sobresortia molt de l'objecte. Per a intentar solucionar aquest problema es va decidir reduir una mica més els contenidors dels objectes, però no ho podíem fer a mà per a cadascun d'ells, sinó que vam plantejar reduir en funció d'un percentatge, la mida tant vertical com horitzontal del col·lisionador. Això ho vam dur a terme creant la funció calculatePercent del script EnemyScript, que ens dóna el resultat final un cop calculat el percentatge. En la següent captura podem veure les modificacions realitzades.

```
        Vector2 S = gameObject.GetComponent<SpriteRenderer>().sprite.bounds.size;
        //Reduce percent of the size
        S.x = calculatePercent(S.x);
        S.y = calculatePercent(S.y);
        //and set values to the box collider
        gameObject.GetComponent<BoxCollider2D>().size = S;
    }

    //function to calculate the percent of reduced size for the box colliders
    private float calculatePercent(float currentSize)
    {
        return currentSize - (currentSize * reduceBoxColliderSizePercent / 100);
    }

}
```

Fig. LXXIII Millora script EnemyScript.

I tot seguit un exemple de com queda després d'aplicar aquesta reducció.

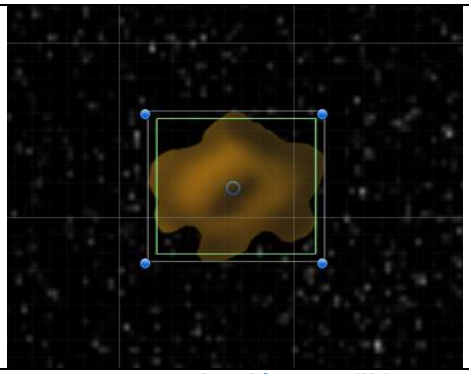


Fig. LXXIV Mostra reducció BoxCollider enemic.

6.- Funcionament de l'aplicació.

En aquest apartat, explicarem el funcionament del joc. El primer pas serà arrancar l'aplicació des del dispositiu. Ens portarà al menú principal del joc.

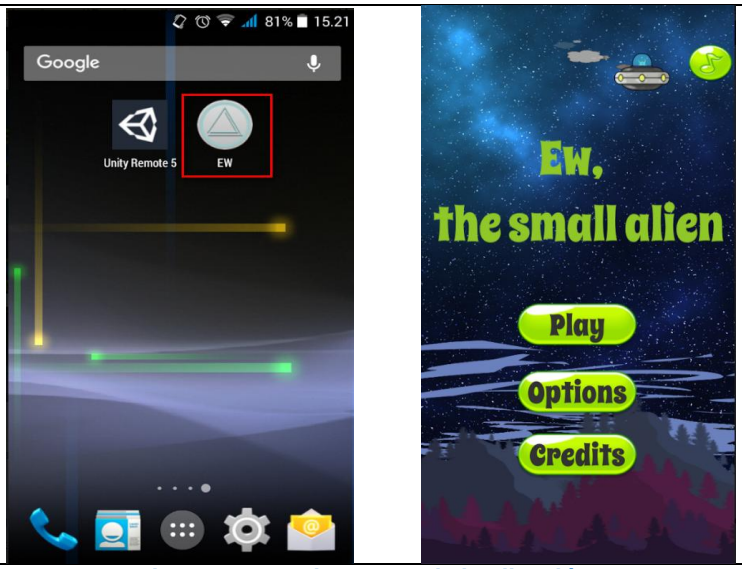


Fig. LXXV Funcionament de l'aplicació 1.

Tot seguit accedirem als diferents menús, prement el botó de Options o Crèdits.

Fig. LXXVI Funcionament de l'aplicació 2.

En el menú d'opcions, podem modificar el volum o la sensibilitat del personatge, simplement movent el Slider cap a un costat o l'altre. Amb el botó de la part superior dreta, podem parar la música. Finalment, amb el botó de la part superior esquerra, tornaríem al menú principal.

L'últim pas és iniciar la partida, per això, un cop situats en el menú principal, seleccionaríem el botó Play. Un cop carregada la pantalla, l'Ew començarà el seu descens. El jugador, haurà de moure el seu dispositiu per canviar la direcció de moviment del personatge i així esquivar els enemics. En la zona d'obertura del paracaigudes, el botó canviarà de color i serà quan l'haguem prémer.

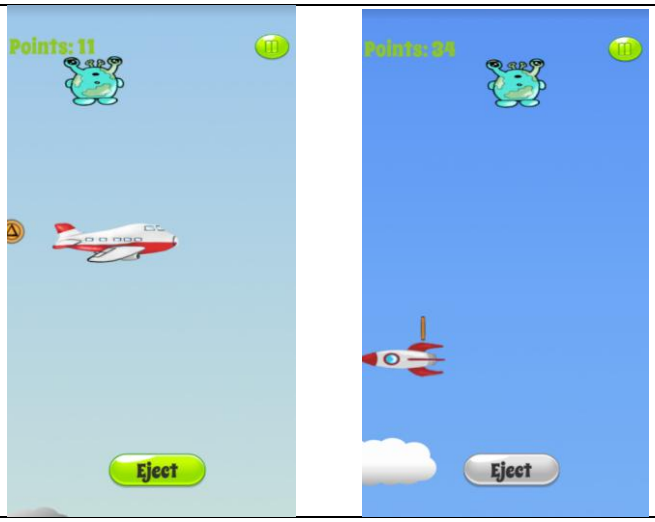


Fig. LXXVII Funcionament de l'aplicació 3.

Si ho hem fet correctament, s'obrirà el paracaigudes. En cas contrari, es perdrà la partida. Finalment, si s'arriba al final de la pantalla amb el paracaigudes obert, s'haurà guanyat.

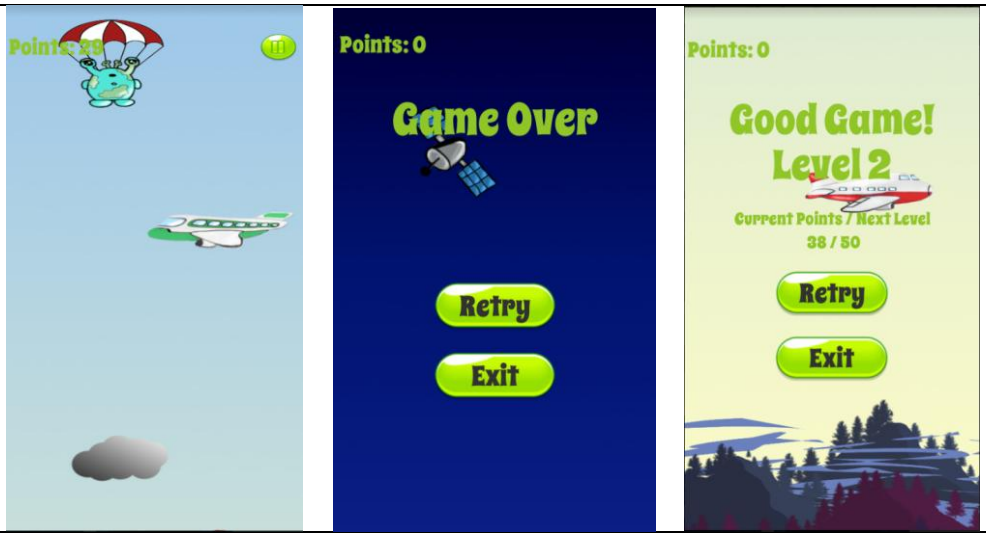


Fig. LXXVIII Funcionament de l'aplicació 4.

7.- Valoració econòmica.

Tot seguit veurem el cost econòmic que podria tenir la implementació d'aquest projecte.

Activitat	Tipus de Recurs	Dedicació (Hores-Persona)	Durada (Dies)	Total Preu Hores-Persona
Estudi del Projecte				
Estudi del Projecte	I1	50h	11 dies	2500€
Requeriments				
Tècnics	I1	20h	4 dies	1000€
Disseny				
Història del joc	D1	5h	1 dia	150€
Dissenys d'interfícies.	D1	15h	3 dies	450€
Dissenys d'objectes.	D1	50h	10 dies	1500€
Disseny o elecció d'altres materials	D1	4h	1 dia	120€
Implementació				
Implementació de Menús	C1	40h	10 dies	2400€
Implementació del joc			29 dies	
Moviment Personatge	C1	25h	5 dies	1500€
Enemics	C1	25h	5 dies	1500€
UI	C1	15h	3 dies	900€
Col·lisions	C1	25h	5 dies	1500€
Altres i proves bàsiques	C1	60h	11 dies	3600€
Proves i millores				
Detecció i disseny de solucions	I1	25 h	6 dies	1250€
Implementació de solucions	C1	30 h	7 dies	1800€
Retoc de disseny	D1	8 h	2 dies	240€
Documentació				
	C1	20 hores	10 dies	1200€
	I1	20 hores	10 dies	1000€
		437h	**133 dies**	**22.610€**

En la següent taula, trobem la relació de preu-hora en relació al tipus de recurs utilitzat.

Tarifes		
Tipus	€/Hora-Persona	Descripció
D1	30€	Dissenyador gràfic, freelance.
I1	50€	Cap de projecte / Consultor Sènior Intern.
C1	60€	Consultor Sènior Extern.

Com podem veure, el projecte té un cost d'implantació de 22.610€ que queden repartits entre l'estudi del projecte, l'estudi de recursos, la implementació, proves i finalment la documentació.

La distribució d'hores/dies realitzats s'ha basat en la taula de planificació de projecte (Fig. II) i en les hores, aproximades, que ha comportat realitzar-lo.

Els recursos que s'han utilitzat, consten d'un cap de projecte, un dissenyador gràfic freelance i finalment un consultor extern.

A tots aquests costos faltaria afegir el cost de llicències i maquinari utilitzat; però al tenir recursos propis i eines gratuïtes, no es tindran en compte en el cost final del projecte.

8.- Conclusions i línies de futur.

Amb aquest projecte s'ha aprés a planificar, dissenyar i implementar un videojoc, per a dispositius Android, amb la plataforma de desenvolupament Unity. Possiblement la part més difícil en el transcurs d'aquest, ha estat tota la part de la creació dels dissenys i música que el conformen, sobretot per a una persona que no està acostumada ni té coneixements al respecte.

Tot i això, estic plenament satisfet del resultat obtingut, tant visual com de jugabilitat. A més, l'experiència extreta ha estat molt gratificant ja que s'ha aconseguit aplicar molts dels coneixements apresos en el transcurs del màster i assolir molts altres coneixements i experiències que es podran aplicar en futurs projectes.

La planificació del projecte va marcar les pautes a seguir a l'hora de plantejar, estudiar i donar solució a les diferents necessitats i objectius plantejats. Concretament, es va definir una estructura de distribució de treball on es plantejaven les fites a aconseguir i els temps que havíem de complir. Gràcies a l'esforç i a les hores dedicades podem dir que hem complert i respectat la planificació plantejada tot i les dificultats de treballar en un entorn desconegut, sobretot pel que fa a la part del disseny i creació dels dibuixos.

Per altra banda, trobem els objectius que vam plantejar al principi i que s'han assolit de forma satisfactòria, deixant temps per a poder aplicar alguna correcció, innovació o funcionalitat que millorarà l'experiència de l'usuari, ja que:

- Totes les pantalles dels menús s'han creat correctament, incloent funcionalitats no planificades, com el moviment del personatge en la pantalla de configuració per ajustar la sensibilitat, o la incorporació d'una pantalla principal dinàmica amb el moviment de la nau i el personatge.
- La creació dels enemics es genera de forma aleatòria, al llarg de tota la pantalla, variant de forma dinàmica els tipus d'enemic que ens anem trobant i els dibuixos d'aquests, fent així un joc més dinàmic i atractiu.
- També s'ha creat un sistema de nivells molt dinàmic, ampliable i que ens permet modificar els paràmetres necessaris per a augmentar la dificultat del joc i que no implica canvis en el codi.
- Els diferents dissenys dels objectes, tot i no tractar-se d'uns dissenys professionals, satisfan les necessitats del joc, creant una estructura uniforme i mantenint el mateix estil.
- El disseny i integració de música i efectes de so, tot i ser l'objectiu que menys s'ha complert, genera molta satisfacció pel fet d'haver creat una melodia sense tenir coneixements ni experiència amb música.
- Finalment, els objectius de compatibilitat entre mòbils o tauletes amb sistema operatiu Android 2.3 o superior, també s'han pogut assolir sense problemes, aconseguint una versió plenament funcional, gràcies a la utilització de funcions estàndards i la plataforma de desenvolupament.

Al final del procés, després d'una revisió i reflexió de tot el desenvolupament creat, ens hem adonat que les possibilitats i millores d'aquest joc són moltes més que les plantejades en aquest projecte, com la inclusió de diferents dissenys de pantalla o la incorporació d'estadístiques de punts; sobretot si es volen assolir nivells de desenvolupament tan alts com els d'altres jocs que ja estan publicats i que tenen un gran èxit mundial. Algunes línies de futur que han anat apareixent i que intentarem aplicar en un futur són les següents:

- Publicació de l'aplicació en diferents botigues virtuals d'aplicacions, com per exemple Google Play, o Amazon apps i veure l'acceptació que té el joc un cop publicat.
- Crear i integrar un sistema d'anuncis de diferents distribuïdores, com per exemple Admob, Leadbolt o Amazon Ads.
- Crear nous continguts, com per exemple diferents dissenys del personatge modificables per l'usuari, més nombre d'enemics, noves recompenses i sistemes de puntuacions, etc.
- Crear una botiga on poder adquirir nou contingut.
- Millorar l'apartat gràfic dels diferents menús i objectes.
- Ampliar les plataformes de creació del joc, per exemple, poder-lo adaptar per a telèfons Apple, aplicacions web o vídeo consoles.
- Crear un sistema de competició entre amics per tal de fer el joc més competitiu i si fos possible multiplataforma.
- Crear nous efectes, per exemple que els avions treguin fum, els coets foc o els meteorits petites partícules.
- Poder destruir els enemics si es prem, de forma continuada, sobre ells i generin petits runes o restes que entorpeixin el moviment.

Gràcies a la realització d'aquest projecte, he pogut millorar els meus coneixements i sobretot aprendre coses totalment desconegudes com la utilització del Photoshop, la creació de videojocs o la gestió de projectes. Tots aquests factors m'han permès veure que aquest és un món molt interessant en el qual m'agradaria mantenir-me actiu ni que fos com a hobby. Però també m'han fet veure, que la creació d'un joc per part d'una sola persona, és molt complex sobretot perquè és quasi impossible tenir habilitats per a tots els camps implicats en la creació de videojocs.

Finalment donar les gràcies a totes les persones que m'han donat suport al llarg d'aquest projecte i la bona acollida que ha tingut per part de les diferents persones que han provat el joc i de les crítiques rebudes que m'han permès millorar diferents aspectes del joc.

9.- Glossari.

Unity: Plataforma especialitzada en el desenvolupament de videojocs.

Android: Programari o sistema operatiu centrat en dispositius mòbils.

Sketches: dibuix fet a mà alçada per a representar les parts essencials d'un disseny.

Scene: Objecte de Unity que té la finalitat de contenidor per emmagatzemar tot el contingut d'una pantalla o menú d'un joc.

Script: Programa o seqüència d'instruccions interpretades per Unity i que ajuden a realitzar les diferents accions del joc.

Guizmos: Funció utilitzada per a la depuració visual en l'editor de Unity.

Particle System: Objecte de Unity per a crear i configurar partícules o efectes visuals.

Audio source: Objecte de Unity per a la configuració i reproducció de sons.

BoxCollider2D: Objecte de Unity per a la detecció de col·lisions d'objectes.

Bounds: Límits o zona límit.

Spawner: Creador d'objectes.

Frame: Imatge que es mostra per la pantalla en un segon. El frame rate, són el nombre de refrescos per segon que es realitza, per exemple, en un joc.

10.- Bibliografia.

[1] **Unity - Scripting API**: https://docs.unity3d.com/ScriptReference/index.html

- [Consultada en línia: Durant tot el projecte].

[2] **Unity Answers**: https://answers.unity3d.com/ - [Consultat en línia: Durant tot el projecte].

[3] **Youtube 3DBuzz canal**: https://www.youtube.com/channel/UCCc85sCjXZish1mGnm8vRFg

- [Consultat en línia: 10/2016].

[4] **Font Squirre, Spicy Rice Tipografy -** https://www.fontsquirrel.com/fonts/spicy-rice - [Consultat en línia: 31 d'octubre del 2016].

[5] **Game Art 2D** - http://www.gameart2d.com/gui.html - [Consultat en línia: 1 de Novembre del 2016].

[6] **Forum Unity** - https://forum.unity3d.com/ - [Consultat en línia: Durant tot el projecte].

[7] **StackOverflow** - http://stackoverflow.com/ - [Consultat en línia: Durant tot el projecte.

[8] **Youtube Charger Games Canal** - https://www.youtube.com/channel/UChvLQbvpw2pfODgOmdS4ZSQ - [Consultat en línia: 16 de desembre del 2016].

11.- Annex.

11.1 – Compressió imatges en Unity.

Per tal de poder reduir la mida de l'aplicació resultant, Unity permet comprimir les imatges importades. Per a fer-ho hem realitzat els següents passos:

1.- Amb el Unity obert, seleccionar l'objecte o imatge que vulguem comprimir.

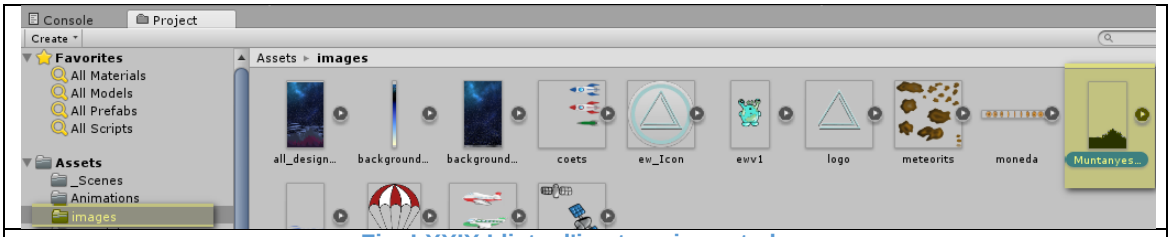


Fig. LXXIX Llista d'imatges importades.

2.- Ens apareixerà la configuració per defecte de la imatge en l'inspector de Unity, i en la part inferior una vista de la imatge amb el pes total, mida i configuració.

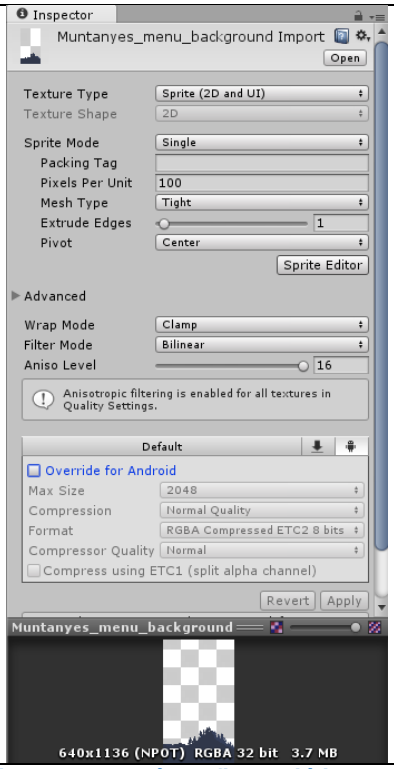


Fig. LXXX Menú configuració imatge.

3.- Activem la opció "Override for Android", modifiquem el "Max Size" reduint el valor tenint en compte que això redueix la qualitat de la imatge, i finalment modifiquem el "Compressor Quality" a Best per assegurar una millor compressió. Per últim, seleccionem la opció "apply".

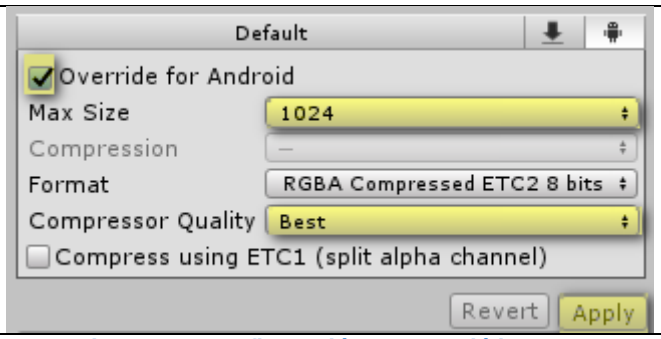


Fig. LXXXI Configuració compressió imatge.

4.- Finalment revisem el resultat obtingut en l'inspector i veiem que la imatge ha reduït de 3.7MB a 3.0MB.

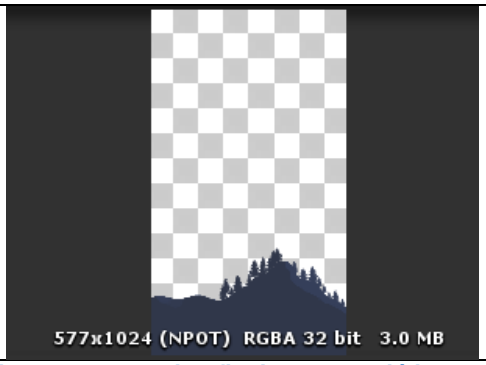


Fig. LXXXII Resultat final compressió imatge.

11.2 – Creació fitxer binari (APK) en Unity.

Els fitxers amb l'extensió "APK" són els fitxers o binaris que es generen per a poder ser instal·lats en els dispositius Android o publicats a la consola de desenvolupament de Google per a la seva posterior comercialització. En aquest apartat, mostrarem els passos a seguir per a generar el fitxer del joc amb Unity.

Primer de tot, en el menú de Unity accedim a l'apartat de "File->Build Settings".

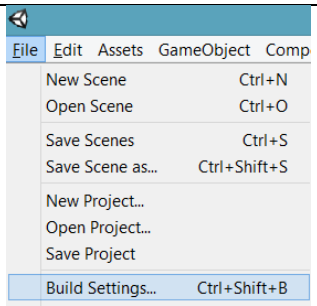


Fig. LXXXIII Menú File Unity.

Ens obrirà la següent pantalla. Aquí haurem de seleccionar la plataforma on volem compilar el joc, en el nostre cas Android, tot seguit seleccionar les "Scenes" que volem exportar i finalment seleccionarem la opció "Build".

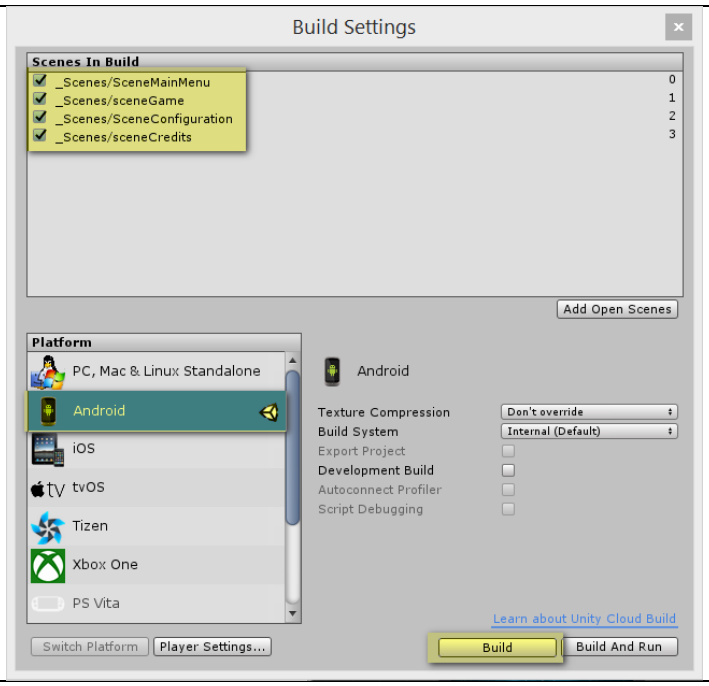


Fig. LXXXIV Build apk file Unity.

L'últim pas serà seleccionar la carpeta destí i ja haurem creat el nostre fitxer "APK" que estarà a punt per a ser instal·lat en qualsevol dispositiu Android.

11.3 – Reducció pes APK en Unity.

Un dels problemes que ens vam trobar era el pes del fitxer APK. Per defecte obteníem un fitxer de 28Mb després de comprimir les imatges tal com hem vist en el punt 10.1 del present document. Per defecte Unity genera el fitxer per a múltiples arquitectures (ARMv7 i x86). La majoria de dispositius mòbils o tauletes Android utilitzen l'ARMv7, cosa que ens permetia dirigir-nos al mercat que ens interessa. Per a canviar aquesta configuració, accedirem al menú de "Build Settings", tal com hem vist en el punt anterior, i seleccionarem la opció "Player Setings".

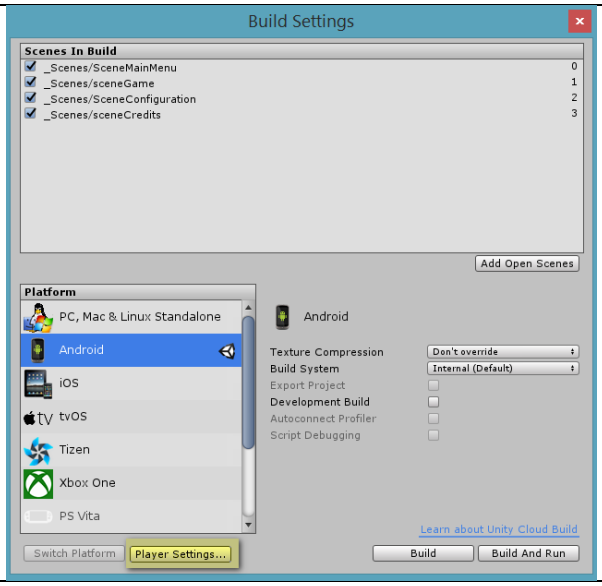


Fig. LXXXV Menú PlayerSettings Unity.

En l'inspector, ens apareixeran les següents opcions, nosaltres ens fixarem en la "Other Settings" i concretament amb "Device Filter", on seleccionarem l'arquitectura ARMv7.

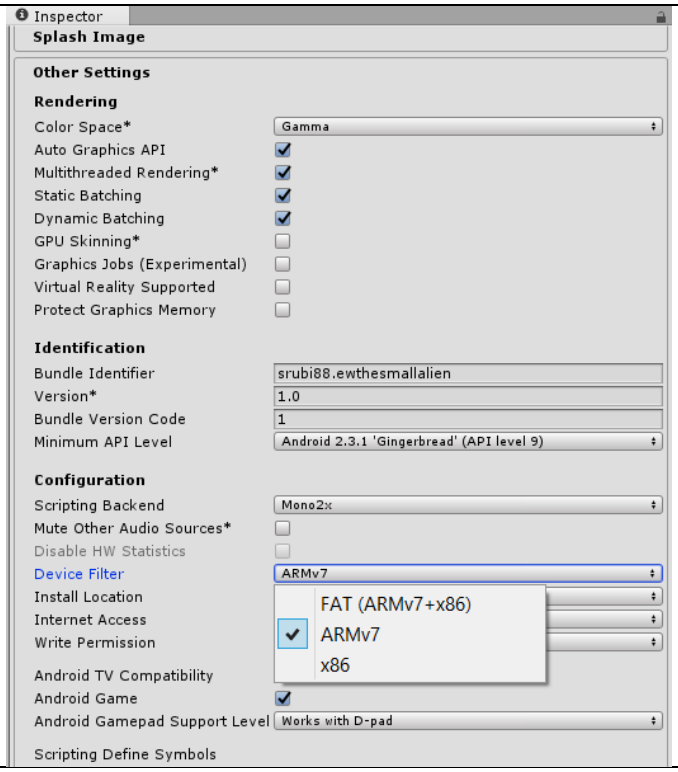


Fig. LXXXVI Menú OtherSettings Unity.

És veritat que d'aquesta forma es redueix la quota de mercat, però el nostre objectiu és centrar-nos en els dispositius mòbils. D'aquesta manera ho garantim, però a més, un cop generem de nou el APK, aconseguim un fitxer resultant de 19MB.

11.4 – Selecció del nivell mínim de compilació d'Android.

Tal com hem indicat en aquest document, l'aplicació havia de ser compatible amb dispositius amb Android 2.3.1 o posterior. Per a poder generar-ho amb aquesta versió mínima, hem d'accedir al menú de "Player Settings", tal com hem vist en els punts anteriors, i buscar la opció "Minumumm API Level" dins del menú "Other Settings". Allà desplegarem la llista i seleccionarem la opció desitjada.

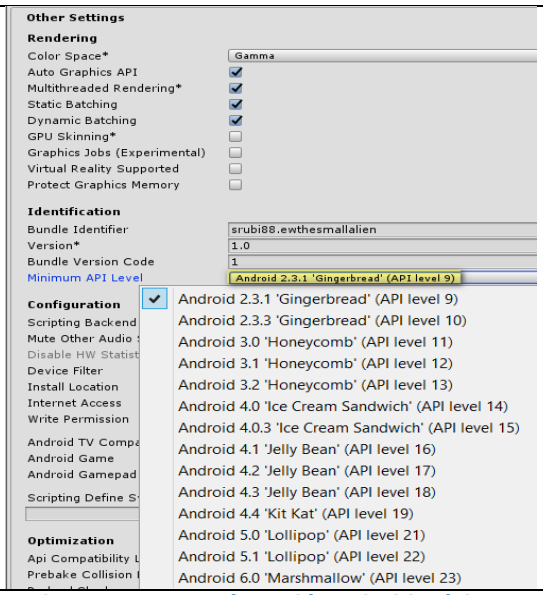


Fig. LXXXVII Menú versió Android mínima.