ALMA MATER STUDIORUM - UNIVERSITA'
DI BOLOGNA CAMPUS DI CESENA

SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE
INFORMATICHE

SHOOTER GAME 2D PER PIATTAFORME MOBILI

Relazione finale in

Mobile Web Design

Relatore

Prof.
Mirko Ravaioli

Presentata da

Stefano Colaci

Sessione III
Anno Accademico 2012/2013

Indice

INTRODUZIONE

L'innovazione introdotta dalle recenti tecnologie portatili sta attraversando un periodo di grande crescita, portando nelle mani di sempre più persone dispositivi più completi e funzionali capaci di eguagliare, e in alcuni casi superare, le prestazioni dei più recenti personal computer.

Questo aumento prestazionale permette agli sviluppatori di confezionare software sempre più complessi e potenti in grado di operare su macchine sempre più piccole e alla portata di tutti.

L'era degli smartphone e dei tablet sta dimostrando che un dispositivo di questo genere può fare molto più che telefonare e mandare messaggi, ma che anzi, queste importanti caratteristiche sono ormai marginali e che un utente medio possa trovare in un "telefonino" un valido sostituto al computer di casa, o a quello portatile.

Un mercato molto importante come quello dei videogiochi non poteva non essere condizionato dal mondo smartphone, specie quando questi piccoli dispositivi raggiungono prestazioni equiparabili con quelle delle più moderne console.

Obiettivo della tesi è quello di analizzare quali sono le innovazioni introdotte dagli smartphone nel mercato dei videogiochi. Verranno poi approfondite quali sono le problematiche da affrontare per la loro realizzazione in vista anche della varietà di sistemi operativi disponibili per i diversi dispositivi e la diversificazione delle configurazioni hardware presenti sul mercato.

L'importanza che le nuove tecnologie hanno portato al mercato videoludico evidenzia quanto sia in costante crescita e abbia ormai superato il mercato delle console per quanto riguarda la distribuzione di applicazioni. Importante è anche la capacità di monetizzazione che si è raggiunta grazie ai dispositivi mobili, al punto che videogiochi da pochi centesimi realizzati da team estremamente

ridotti ed in tempi ristretti guadagnino più di titoli importanti realizzati da software house prestigiose per le più potenti console del momento.

Obiettivo della tesi è anche quello di illustrare la realizzazione di un semplice videogioco compatibile con più sistemi operativi mobili.

1

STATO DELL'ARTE

In questo capitolo verranno analizzati i diversi dispositivi mobili e i diversi sistemi operativi disponibili sul mercato. Verranno inoltre approfondite quali siano le diverse configurazioni hardware disponibili e le principali vie di distribuzione digitale del software.

1.1 Dispositivi Mobili

Con il termine “dispositivi mobili” intendiamo tutti quei dispositivi che sono pienamente utilizzabili seguendo la mobilità dell'utente. Storicamente i primi dispositivi di questo tipo sono stati i telefoni cellulari di prima generazione. Alcuni di questi dispositivi quali smartphone e tablet assumono importanza all'interno del contesto del cosiddetto web mobile con fette di mercato in continua espansione.

1.1.1 Smartphone

Uno smartphone è un telefono cellulare basato su un sistema operativo per dispositivi mobili, con capacità di calcolo e di connessione molto più avanzate rispetto ai più comuni e ormai datati telefoni cellulari. Con l'introduzione del touchscreen, ormai presente sulla stragrande maggioranza di dispositivi disponibili sul mercato, hanno finito con il soppiantare i vecchi palmari (PDA). Combinando le caratteristiche di questi ultimi con quelle dei telefoni cellulari hanno raggiunto lo stato di telefono intelligente, “smart” appunto.

La loro grande portabilità e il loro prezzo, più accessibile di quello di un PC, lo rendono il dispositivo per eccellenza con numeri di vendite da record e sempre in crescita. In totale, nel mondo sono stati venduti lo scorso anno (2013) 1,82 miliardi di telefonini (+4,8%), di cui 488,4 milioni nel quarto trimestre.

1.1.2 Tablet

Diversamente dagli smartphone un altro dispositivo che trova anch'esso il punto di forza nella portabilità e nella semplicità di utilizzo è il "tablet computer" o più comunemente chiamato "tablet".

Caratterizzato da dimensioni compatte e dal fatto che utilizza come unico sistema di input uno schermo controllato da una penna o tramite dita invece che una tastiera e un mouse.

Il loro nome deriva dalla forma che assomiglia a quella di una tavoletta utilizzata per la scrittura. Non essendo dotati di tastiera, questi dispositivi utilizzano in genere una tastiera virtuale su schermo a comparsa, quando necessario.

La vendita di questi terminali sta soppiantando quella dei comuni Notebook portando questi ultimi a dover adottare soluzioni ibride (solitamente pc portatili, con schermo touchscreen e tastiera rimovibile) per restare sul mercato.

1.1.3 Altri dispositivi mobili

In forte espansione è anche il mercato dei dispositivi così detti "indossabili". Terminali dotati di sistemi operativo e, il più delle volte, provvisti delle stesse specifiche hardware di uno smartphone.

I più comuni sono gli Smartwatch, veri e propri "orologi da polso" dotati delle più comuni funzioni intelligenti da renderli appunto "smart". Il mercato offre diverse soluzioni per questo tipo di prodotto, possono essere terminali

capaci di operare in piena autonomia, dotati quindi di Sim per le telefonate e per la connessione ad internet, oppure possono essere terminali di supporto agli smartphone. In questo caso non sono dotati di un modulo telefonico ma si interfacciano al dispositivo primario tramite Bluetooth e permettono, ad esempio, la visualizzazione delle notifiche senza dover estrarre il telefono dalla tasca.

Un altro dispositivo di questo genere è lo "smart band", o braccialetto intelligente, capace di collegarsi allo smartphone e di fruire diverse tipologie di informazioni utili a seconda dei sensori installati.

Non è obiettivo della tesi approfondire questo tipo di dispositivi in quanto allo stato attuale non offrono particolari spunti per la realizzazioni di videogiochi.

1.2 Architetture Hardware

L'hardware di riferimento per questo tipo di dispositivi è basato su architettura ARM.

Attualmente la famiglia ARM copre il 75% del mercato mondiale dei processori a 32 bit per applicazioni embedded, ed è una delle più diffuse architetture a 32 bit del mondo. I processori ARM vengono utilizzati in cellulari, tablet, lettori multimediali, videogiochi portatili, PDA e periferiche per computer (come router, hard disk di rete ecc).

Tutti i dispositivi iPhone e iPad montano, fin dai primi modelli, processori basati su questa architettura, la stessa cosa vale per gli smartphone e i tablet di riferimento per il mondo Android e Windows Phone.

ARM non produce direttamente i chip, si limita a vendere licenze per il loro utilizzo. Le licenze permettono di usare i core ARM come componenti base

attorno ai quali poi costruire uno specifico circuito integrato. Tipicamente su un chip di uno smartphone compaiono oltre alla CPU anche memorie e GPU.

Le caratteristiche cardine dell'architettura ARM: semplice, studiata per il basso consumo e fortemente modulare; fanno sì che venga scelta da quasi tutti i produttori di smartphone e tablet. In particolare l'ultima versione: ARMv7 e il suo set di istruzioni sono stati scelti da Apple, Htc, Nokia, Samsung e molti altri per i propri dispositivi.

1.2.1 Principali processori

Di seguito verranno analizzati alcuni dei processori più importanti presenti sul mercato. E' interessante notare come l'aumento di prestazioni raggiunto favorisca lo sviluppo di videogiochi sempre più importanti al pari delle più recenti console.

Qualcoom Snapdragon

Il core del processore Snapdragon, soprannominato Scorpion, è stato progettato dalla stessa Qualcomm. Ha molte caratteristiche simili all'architettura ARM di tipo Cortex-A8 ed è basato sul set di istruzioni dell'ARM v7, ma in teoria ha una potenza molto più elevata per quanto riguarda le operazioni SIMD per applicazioni multimediali.

Tutti i processori Snapdragon contengono la circuiteria necessaria per decodificare video ad alta definizione (HD) alla risoluzione di 720p o 1080p a seconda del chipset Snapdragon usato. Adreno, la tecnologia GPU proprietaria dell'azienda, integrata nei chipset Snapdragon è progettata interamente da Qualcomm, usando gli asset che l'azienda ha acquisito da AMD.

Il processore più recente è lo SnapDragon 800, dotato di una CPU Krait 400 quad-core con frequenze fino a 2.3GHz e una GPU Adreno 330.

Nvidia Tegra

Tegra è il nome di una famiglia di SoC sviluppata da NVIDIA per dispositivi mobili come gli smartphone e i tablet. Tegra integra una CPU con architettura ARM, un'unità di elaborazione grafica, northbridge, southbridge e controllore di memoria in un singolo package. Annunciata per la prima volta a metà 2008, tali processori hanno alte prestazioni di riproduzione audio e video (es: supporto per l'alta definizione) e permettono un'esperienza internet completa, con un basso consumo di corrente.

Tegra 4, annunciato il 6 gennaio 2013, è un SoC con una CPU quad-core, ma include anche un quinto core di supporto a basso consumo di tipo Cortex A15 che risulta invisibile al sistema operativo, predisposto per occuparsi dei processi che vengono eseguiti in background per risparmiare energia. La migliorata GPU a 72 core prevede il supporto per DirectX 11+, OpenGL 4.X e PhysX.

Mediatek

Si tratta di CPU di fascia bassa in grado di offrire prestazioni interessanti unite a consumi ridotti. Nella soluzione quad core inoltre, Mediatek ha visto un ottimo successo anche su smartphone di buon livello e con i Dual SIM la CPU ha sicuramente una marcia in più grazie all'ottima gestione di due (o addirittura 3) SIM card contemporaneamente.

Tutte le soluzioni sono basate su architettura Cortex-A7 a 28nm e la nuova versione Octa Core MT6592 dovrebbe portare ancora più potenza riuscendo a mantenere bassa la richiesta di energia attivando 1 o tutti e 8 i core a seconda della potenza richiesta dal sistema

Intel Atom

Sono microprocessori x86 sviluppati da Intel espressamente per il settore dei dispositivi ultra portatili, UMPC e MID, e notebook/desktop di fascia molto economica (conosciuti rispettivamente come Netbook e Nettop). Si tratta quindi di una famiglia di CPU dalle piccolissime dimensioni e dai consumi estremamente contenuti.

Di recente sono stati introdotti anche su smartphone, in particolare sulla linea Asus ZenPhone.

Processori Apple

I processori Apple A6, dual-core con frequenza fino a 1.5 GHz, sono basati su un core ARMv7 sviluppato interamente da Apple e chiamato Swift. Le informazioni fornite da Apple sui suoi processori sono poche, ma è noto che l'A6 utilizza un set di istruzioni ARMv7 ottimizzato e che l'architettura interna utilizza alcuni elementi dei processori ARM Cortex-A15.

L'A7, al contrario del predecessore, oltre ad essere più performante è il primo processore a 64bit ad essere montato su uno smartphone. Questi processori sono montati esclusivamente su dispositivi Apple, quindi Iphone e Ipad.

1.2.2 Sensori

Grazie alla grande quantità di sensori integrati sui dispositivi mobili è stato possibile rivoluzionare l'interazione con l'utente e quindi, per quanto riguarda i videogames, il modo di giocare ha subito una forte evoluzione.

Display multi-touch

Multi-touch è una tecnologia che rappresenta un'evoluzione dello schermo tattile. Uno schermo tattile multitocco si differenzia dai precedenti per il fatto che è sensibile al tocco in più punti diversi della superficie contemporaneamente.

L'introduzione di questa tecnologia ha permesso di sviluppare interfacce sempre più intuitive e naturali per l'utente e, nel caso devi videogiochi, ha arricchito l'esperienza di gioco offrendo all'utente nuove opportunità di interazione con la scena senza l'utilizzo di tasti fisici.

Accelerometro

L'accelerometro è uno strumento in grado di misurare l'accelerazione subita ad esempio, da un dispositivo nel quale è installato.

Negli ultimi anni l'importanza di questi sensori ha subito un notevole incremento, questo perché, accanto alle tradizionali applicazioni in ambito scientifico e aerospaziale, si è sviluppato il loro uso in molti campi civili (automobilistico, testing, analisi meccanica, ludico, ecc.).

Non è difficile immaginare la rivoluzione che ha avuto nel modo di pensare i videogiochi. Basti pensare ai giochi di guida dove è possibile controllare la propria vettura come fosse un volante per sterzare.

Giroscopio

Il giroscopio è un sensore in grado di misurare la rotazione del dispositivo rispetto ad un asse. I dispositivi di ultima generazione montano giroscopi a tre assi, in gradò di misurare i gradi di rotazione del dispositivo rispetto ad una terna di assi cartesiani.

Anche in questo caso, l'impiego di sensori di questo tipo offre molti spunti nella realizzazione di videogiochi, simulando ad esempio l'esplorazione di un ambiente 3d semplicemente muovendo il terminale.

Camera

Accessorio ormai comune anche su dispositivi di fascia bassa è la fotocamera. Capace di catturare foto e video può essere sfruttata nella realizzazione di giochi in realtà aumentata. Grazie al riconoscimento di oggetti inquadrati, ad esempio, può fruire informazioni multimediali in tempo reale degni dei più famosi film di fantascienza.

GPS

Il "Sistema di Posizionamento Globale" (Global Positioning System, abbreviato GPS) è un sistema di posizionamento e navigazione satellitare civile che, attraverso una rete satellitare dedicata di satelliti artificiali in orbita, fornisce ad un terminale mobile o ricevitore GPS informazioni sulle sue coordinate geografiche ed orario, in ogni condizione meteorologica, ovunque sulla Terra o nelle sue immediate vicinanze ove vi sia un contatto privo di ostacoli con almeno quattro satelliti del sistema. La localizzazione avviene tramite la trasmissione di un segnale radio da parte di ciascun satellite e l'elaborazione dei segnali ricevuti da parte del ricevitore.

Altri sensori

Sono presenti altri tipi di sensori su smartphone e tablet, come il sensore di prossimità e di luminosità in grado di regolare lo spegnimento e la luminosità dello schermo, molto utili per il risparmio energetico. Non hanno ancora

trovato un uso specifico per quanto riguarda i videogiochi ma non è escluso che possano essere sfruttati nella realizzazioni di videogame particolari e originali.

1.3 Piattaforme software

Al contrario dei primi modelli sul mercato con sistema operativo proprietario, dove la realizzazione di software da parte di terzi era rara e poco proficua data la loro diversificazione, oggi la presenza di pochi sistemi operativi mobili ha dato la possibilità agli sviluppatori di confezionare applicazioni sempre più sofisticate e remunerative dato l'alto numero di utenza finale raggiungibile.

Un traguardo, raggiunto solo dai personal computer, dove chi produce i calcolatori e chi produce del software operativo sono due entità distinte (salvo casi particolari).

Di seguito verranno analizzati i più importanti e diffusi sul mercato e per i quali lo sviluppo di un videogame sarebbe più indicato per il raggiungimento di un maggior numero di utenti.

1.3.1 iOs

iOS è un sistema operativo sviluppato da Apple per iphone, ipod touch e ipad.. Come Mac OS X, è una derivazione di UNIX (famiglia BSD) e usa un microkernel XNU Mach basato su Darwin OS. iOS ha quattro livelli di astrazione: il Core OS layer, il Core Services layer, il Media layer e il Cocoa Touch layer. Il sistema operativo occupa meno di mezzo Gigabyte della memoria interna del dispositivo.

Ogni App del mondo Apple ha bisogno del sistema operativo iOS per andare in esecuzione.

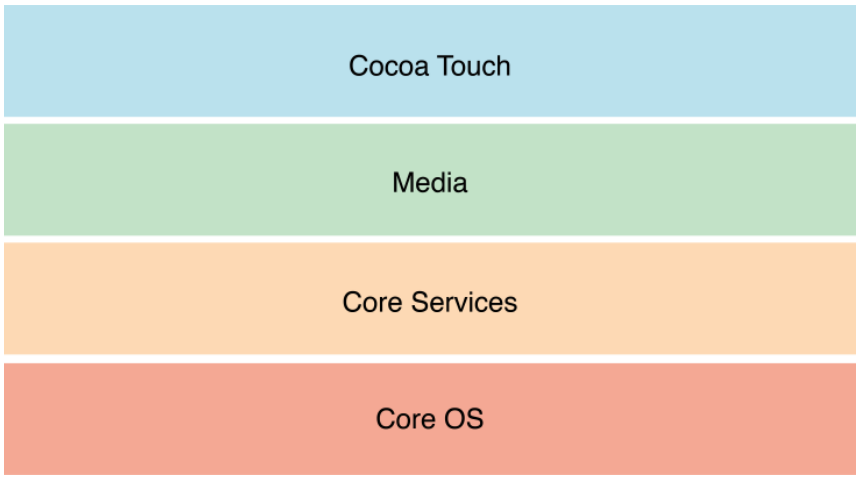


Figura 1.1 – Livelli di astrazione di iOS

Il sistema Apple ha delle peculiarità di funzionamento che ne determinano una diversa fruizione delle applicazioni e dei dispositivi. Uno studio della Symantec evidenzia come questo sistema operativo sia più sicuro del maggiore avversario Android. Anche se ha evidenziato che è più sicuro soprattutto per via dei minori tentativi di attacco da parte degli hacker, che concentrano gli sforzi sul suo antagonista in quanto maggiormente diffuso a livello mondiale. E' il secondo sistema operativo mobile al mondo per diffusione.

1.3.2 Android

Android è un sistema operativo per dispositivi mobili organizzato in un'architettura software che include un sistema operativo di base, i middleware per le comunicazioni e le applicazioni di base.

Si caratterizza per la struttura open source (escluse alcune versioni intermedie), e il suo basarsi su kernel Linux. La caratteristica open source ed il tipo di licenza (Licenza Apache) permette di modificare e distribuire liberamente il codice sorgente. Inoltre, Android dispone di una vasta comunità

di sviluppatori che realizzano applicazioni con l'obiettivo di aumentare le funzionalità dei dispositivi. Queste applicazioni sono scritte soprattutto con una versione modificata del linguaggio di programmazione Java.

Nell'ottobre 2012 le applicazioni disponibili presenti sul market ufficiale Android (Google Play) hanno raggiunto le 700.000 unità. Questi fattori hanno permesso ad Android di diventare il sistema operativo più utilizzato in ambito mobile, oltre a rappresentare, per le aziende produttrici, la migliore scelta in termini di bassi costi, personalizzazione e leggerezza del sistema operativo stesso, senza dover scrivere un proprio sistema operativo da zero. Il progetto Open Source Android è guidato da Google.

Android è costituito da un Kernel basato sul kernel Linux 2.6 e 3.x (da Android 4.0 in poi), con middleware, Librerie e API scritte in C (o C++) e software in esecuzione su un framework di applicazioni che include librerie Java compatibili con librerie basate su Apache Harmony. Android utilizza la Dalvik virtual machine con un compilatore just-in-time per l'esecuzione di Dalvik dex-code (Dalvik Executable), che di solito viene tradotto da codice bytecode Java. La piattaforma hardware principale di Android è l'architettura ARM. L'architettura x86 è supportata grazie al progetto Android .

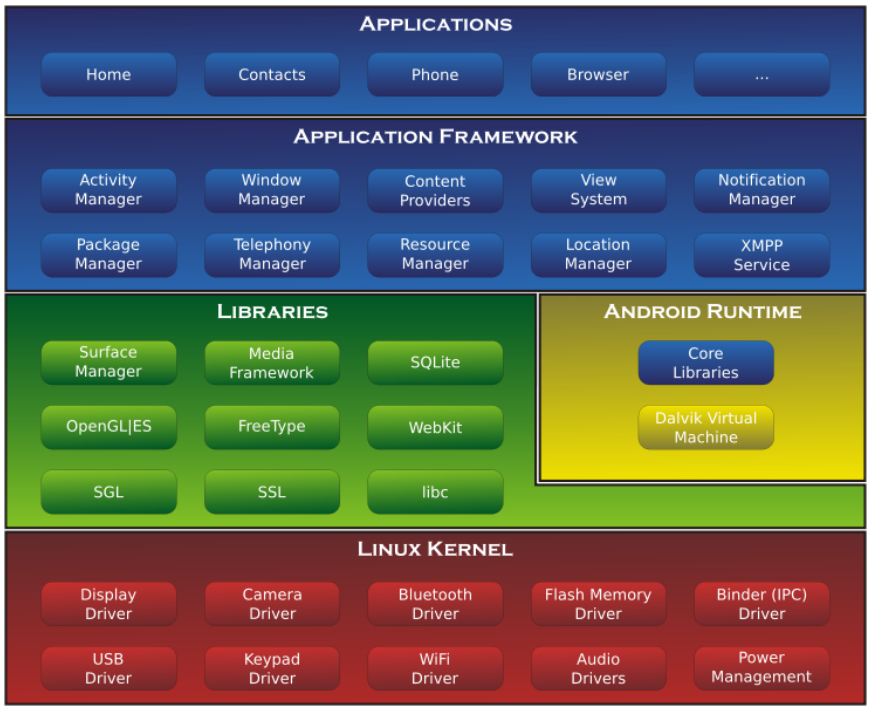


Figura 1.2 – Livelli di astrazione Android

Il kernel Linux di Android mette a disposizione modifiche all'architettura create da Google al di fuori del ciclo di sviluppo del kernel. Un tipico sistema Android non possiede un X Window System nativo, né supporta il set completo standard di librerie GNU, e nel caso del C++ vi è solo un'implementazione parziale delle STL.

Le applicazioni Android sono Java-based: tutte le porzioni di codice, anche in C/C++, devono essere inserite all'interno di un progetto Java, il quale produce alla fine un pacchetto Android (APK).

1.3.3 Windows Phone

Windows Phone è il sistema operativo per smartphone di Microsoft, presentato al Mobile World Congress il 15 febbraio 2010. Si rivolge al mercato consumer invece che al mercato enterprise come il suo predecessore, abbandonando alcune caratteristiche in dotazione a Windows Mobile.

Il problema principale di Windows Phone è la presenza di un minore numero di applicazioni presenti nel suo Store rispetto ai principali concorrenti, tuttavia il gap si sta progressivamente riducendo.

La versione 8 è basata sul kernel Windows NT, lo stesso utilizzato da Windows 8 (con cui condivide anche file system, driver, gestione rete, sicurezza e componenti multimediali e grafiche, logica dell'interfaccia "a tiles"), permette l'utilizzo di linguaggi di programmazione standard come il C++ e C#. Porta grandi novità anche per quanto rugarda la personalizzazione, infatti sarà possibile regolare la grandezza delle tiles in tre formati (piccole, medie e grandi); ad esempio le tiles di sms o e-mail visualizzano un'anteprima del contenuto appena ricevuto se ridimensionate nel formato grande. Oltre ai miglioramenti grafici, Windows Phone 8 introduce dal lato hardware il

supporto ai processori multicore, alle schede MicroSD, ai pagamenti NFC, ai display di diverse dimensioni e risoluzioni (HD e Full HD) ed è compatibile con nuovi standard Bluetooth che ampliano l'utilizzo di quest'ultimo. Integra la crittografia BitLocker e migliora gli strumenti Office; garantisce una serie di API per l'integrazione di servizi aggiuntivi di photosharing, messaggistica e videochiamate (come Instagram, WhatsApp, Skype ecc.)

Applicazioni e giochi per Windows Phone possono essere basati su XNA o su specifiche versioni di Microsoft Silverlight e da Windows Phone 8 anche in codice nativo. Windows Phone supporta molti potori grafici 3d per lo sviluppo di Giochi come l'Unreal Engine.

1.4 Distribuzione

E' importante sapere quali siano le vie di distribuzione di un software sviluppato per questo tipo di dispositivi. Tutte le applicazioni per smartphone e tablet sono fruite all'utente in via digitale attraverso appositi negozi online a seconda del sistema operativo per cui sono state create.

1.4.1 App Store

L'App Store è un servizio realizzato da Apple disponibile per iPhone, iPod touch e iPad che permette agli utenti di scaricare e acquistare applicazioni disponibili su iTunes Store. Le applicazioni possono essere sia gratuite che a pagamento, e possono essere

scaricate direttamente dal dispositivo o su un computer. L'App Store è stato aperto il 10 luglio 2008 tramite un aggiornamento software di iTunes. Nel gennaio 2011 erano disponibili in App Store più di 325.000 applicazioni sviluppate da terze parti, con oltre 25 miliardi di download.

L'App Store è accessibile da iPhone, iPod touch, e iPad solo attraverso l'applicazione nativa "App Store".

Le applicazioni possono esclusivamente essere vendute tramite l'iTunes Store per Mac e Windows oppure tramite l'applicazione "App Store" presente su iPhone, iPod touch, e iPad. Dal 6 gennaio 2011 lo troviamo anche su Mac, disponibile con "OS X 10.6 Snow Leopard", chiamato semplicemente "Mac App Store".

1.4.2 Google Play

Google Play detto anche Google Play Store (precedentemente denominato Android Market) è un negozio virtuale di applicazioni, brani musicali, pellicole cinematografiche, libri e riviste online sviluppato da Google per i dispositivi Android, per i sistemi desktop e notebook che abbiano disponibilità della connessione Internet. Ha sostituito l'Android Market, il quale trattava solo applicazioni per Android.

Un programma applicativo chiamato Play Store viene pre-installato sulla maggior parte dei dispositivi Android e permette agli utenti di cercare e scaricare le applicazioni pubblicate da sviluppatori di terze-parti e leggerne le relative informazioni e aggiornamenti.

1.4.3 Amazon Appstore

Amazon Appstore è un negozio virtuale alternativo che distribuisce applicazioni per dispositivi basati su sistemi operativi Android. La scelta da parte di Amazon di creare un proprio store deriva dal fatto che negli ultimi anni l'azienda ha realizzato una linea di tablet, denominati Kindle Fire, che montano al loro interno un sistema operativo basato su Android 2.3 Gingerbread ma con una forte personalizzazione da parte di Amazon.

1.4.4 Windows Phone Store

Windows Phone Store (precedentemente chiamato Marketplace) è un servizio, sviluppato dalla Microsoft, disponibile per Windows Phone che permette agli utenti di cercare e scaricare applicazioni sviluppate per il sistema operativo.

Come tutti i software per Windows Phone presenta l'interfaccia grafica nel linguaggio di design Microsoft, e gli utenti possono cercare tra le categorie, vedere le caratteristiche, visualizzare i commenti e le immagini di preview, controllare il prezzo.

Ad un anno dal lancio, il 21 ottobre del 2011 il Marketplace contava oltre 34.000 applicazioni scaricabili. Il 25 febbraio sono state superate le 70 mila app.

Ad aprile 2013 contava 120.000 applicazioni. Con il lancio di Mango (alias Windows Phone 7.5) la Microsoft ha svelato il sito online del Marketplace, questo offre numerose caratteristiche come la possibilità di installare applicazioni sul proprio telefono senza collegarlo. Nel novembre del 2013 ha superato le 190.000 applicazioni.

1.5 Applicazioni simili

Nei prossimi capitoli verrà preso in esame lo sviluppo di un'applicazione per smartphone, è bene quindi analizzare quali siano le applicazioni simili presenti sul mercato e le varie peculiarità che le rendono di successo.

Il caso di sviluppo preso in esame consiste in un videogame del genere sparatutto a scorrimento orizzontale nel quale l'utente controllerà una torretta difensiva e dovrà respingere le ondate di alieni che li si porranno di fronte evitando i colpi del nemico attraverso il tocco sullo schermo che provocherà l'innalzamento della torretta in verticale.

1.5.1 Metal Slug

Metal Slug è un videogioco del genere sparatutto a scorrimento orizzontale prodotto dalla SNK Playmore per Neo Geo e arcade. Famoso per il suo senso dell'umorismo e per l'animazione estremamente fluida, costituisce il primo videogioco della serie omonima.

Rilasciato anche per iOs e Android, tramite un porting, è uno degli Shooter Game di maggior successo. Lo scopo del gioco è quello di eliminare i vari nemici che si troveranno di fronte al personaggio controllato dall'utente durante il proseguire del livello prima di affrontare i Boss per il suo completamento.

Figura 1.3 – Gameplay di Metal Slug

1.5.2 Jetpack Joyride

Jetpack Joyride è un videogioco a scorrimento orizzontale sviluppato dalla Halfbrick Studios. È stato pubblicato il 1° settembre 2011 per iOS, il 19 agosto 2012 per Android e il 23 ottobre 2012 per Windows 8. Insieme a "Age of Zombies" e "Monster Dash" fa parte della serie Barry Steakfries. Nel novembre 2012 è stato rilasciato per PSP, mentre nel dicembre 2012 è stato reso gratuito, dalla Big Ant Studios, per PlayStation 3 e PS Vita.

La base del meccanismo di gioco è basata sul tocco dello schermo da parte del giocatore per dare potenza al jetpack e quindi far sollevare Barry. Il giocatore deve andare il più lontano possibile da dove ha rubato il jetpack, raccogliendo le monete e evitando missili, raggi laser ed elettrodi che generano scariche, che tentano di ucciderlo. Durante la corsa, il giocatore può accumulare anche i Gettoni Slot, che appaiono a mezz'aria di tanto in tanto e servono per poter giocare e tentare di vincere premi alla slot machine che appare dopo la morte del personaggio (solo se giocando ha preso almeno un gettone). Di tanto in tanto appaiono anche delle scatole color arcobaleno, che contengono dei veicoli, che possono aiutare Barry durante la sua fuga.

Figura 1.4 – Gameplay di Jetpack Joyride

2

PROGETTAZIONE

Nelle prossime pagine verrà illustrato il caso di sviluppo di riferimento per questa tesi. Verrà descritta l'applicazione nei suoi particolari e mostrate le varie schermate che la comporranno nonché i problemi progettuali affrontati e le metodologie adatte a risolverli.

2.1 Descrizione del videogame sviluppato

Come precedentemente accennato il software sviluppato per questa tesi è un videogame in 2d per smartphone di genere sparatutto a scorrimento orizzontale.

L'utente dovrà controllare una torretta difensiva, capace di utilizzare differenti tipologie di armi nel corso dei livelli, per difendere la terra dall'invasione aliena. Anche se si è parlato di sparatutto a scorrimento in realtà lo stato del livello sarà fermo, non si dovrà avanzare per proseguire, saranno le sprite nemiche a comparire sullo schermo da destra verso sinistra avvicinandosi ed attaccando la torretta.

Il gioco non farà uso di particolari sensori se non, ovviamente, il touchscreen per evitare i colpi e sparare ai nemici. Lo scopo dell'utente sarà quello di ottenere il maggior numero di “punti dna” abbattendo gli alieni.

Ogni livello offrirà varie tipologie di nemici da dover affrontare e di armi da poter raccogliere, abbattendo le “casse” che compariranno ogni tanto sullo schermo. La quantità di livelli e di diverse tipologie di nemici è fondamentale per la buona varietà dell'esperienza di gioco, per questa si cercherà di scrivere un codice abbastanza elastico da poter supportare l'aggiunta di diverse porzioni di codice senza però modificare la struttura portante.

L'arma principale del giocatore sarà una mitragliatrice con munizioni infinite. Com'è solito in questo tipo di situazioni, per non rendere il gioco troppo facile, si implementa una sorta di "surriscaldamento" dell'arma. In questo modo se l'utente sparerà con troppa frequenza il surriscaldamento impedirà all'arma di sparare ulteriormente finché non si sia "raffreddata". Questa soluzione è adottata in numerosi giochi sparatutto in cui il giocatore si trovi a dover usare un'arma con munizioni che, data la situazione, occorre che siano infinite. Mentre le altre armi avranno munizioni finite, ed una volta esaurite l'arma verrà persa e si potrà riutilizzare la mitragliatrice principale. Ogni tanto si presenteranno anche nemici con un prigioniero e abbattendoli, questo verrà liberato e ripristinerà i danni subiti dalla nostra torretta.

Per quanto riguarda l'interfaccia, l'utente dovrà poter:

- visionare i propri risultati di gioco
- selezionare il livello desiderato
- sospendere e riprendere il gioco durante la sua esecuzione
- visualizzare a video le informazioni di gioco, quindi i suoi punti vita e le munizioni rimaste
- condividere i propri risultati di gioco
- terminare l'applicazione.

2.2 Struttura del progetto

Una volta avvenuta l'installazione dell'applicazione sul proprio dispositivo, sarà avviabile tramite l'apposita icona con il nome del gioco. Dopo la visualizzazione di un'immagine iniziale detta “splash” (quale di solito raffigura il logo della software house, motore grafico e vari sponsor) e durante la quale vengono caricati in memoria le risorse necessarie, verrà mostrato il menu principale da dove sarà possibile avviare il gioco.

Figura 2.1 – Schermata iniziale di gioco

Dopo aver selezionato il tasto “gioca” verrà mostrata la schermata di selezione livello. Qui l'utente potrà selezionare il livello da affrontare a seconda di quelli disponibili o tornare alla schermata precedente.

Figura 2.2 – Schermata di selezione livello

Selezionato il livello si avvierà la scena di gioco. Gli oggetti necessari popoleranno la scena e l'utente sarà nel pieno del gameplay. Questa è la schermata principale dell'applicazione e quella che occuperà la maggior parte delle risorse hardware richieste.

Per quanto riguarda i controlli, l'utente tramite un tap più o meno prolungato sulla parte sinistra dello schermo può far innalzare la torretta per schivare i proiettili dei nemici. Mentre con un tap sulla parte destra, dove avanzeranno i nemici, provocherà lo sparo. La torretta cambierà la sua inclinazione a seconda delle coordinate del tap sullo schermo e il proiettile seguirà quella direzione o nel caso di armi da lancio soggette a gravità e quindi ad una parabola determinerà la sua ampiezza.

Figura 2.3 – Schermata di gioco

Ovviamente per abbattere i nemici bisognerà colpirli un numero di volte tale da azzerare i loro punti vita. Alcuni nemici, i boss di gioco, per essere abbattuti dovranno essere colpiti in determinati punti altrimenti la collisione non avrà luogo e non verranno inflitti danni.

Liberando gli scienziati tenuti prigionieri questi verranno verso la torretta e, dopo una piccola animazione, ripristineranno i danni che l'utente ha subito fino a quel momento. Distruggendo le casse che talvolta cadranno si otterranno altre armi, più efficaci ma con un quantitativo di munizioni limitate.

Figura 2.4 – Schermata di gioco

Mettendo in pausa il gioco, tramite apposito tasto in alto a destra, verrà mostrata la relativa schermata da dove l'utente potrà riprendere il gioco o terminare l'applicazione.

Figura 2.5 – Pausa durante il gioco

Nel caso la vita del giocatore si esaurisca verrà mostrata la schermata di "game over" con il punteggio ottenuto.

Figura 2.6 – Schermata di Game Over

2.2.1 Elementi principali della scena di gioco

E' importante analizzare quali siano gli elementi principali che comporranno la schermata principale e per i quali sarà necessaria una rappresentazione grafica sulla scena.

Torretta difensiva

La torretta è l'elemento con cui il giocatore è rappresentato nella scena. Come già accennato, può innalzarsi tramite un tap prolungato sulla parte sinistra dello schermo per poi abbassarsi una volta che il dito viene rilasciato.

Questo meccanismo permette al giocatore di schivare i proiettili nemici e quindi proseguire nel livello.

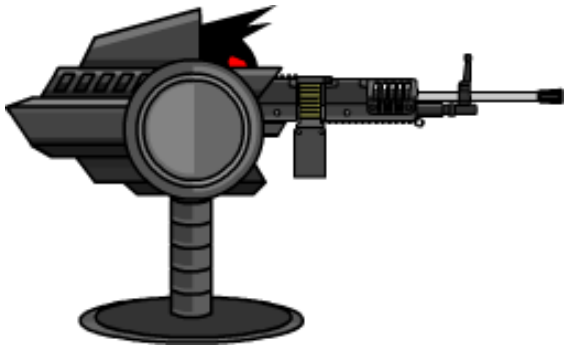


Figura 2.7 – Torretta difensiva

Arma

L'arma è l'oggetto che va a comporre la torretta. Con il tap sulla parte destra dello schermo l'utente provocherà lo sparo, il quale genererà uno o più proiettili pronti a segnalare le proprie collisioni con gli altri oggetti della scena. Sono presenti diversi tipi di armi, ognuna con particolarità diverse.

Alieni

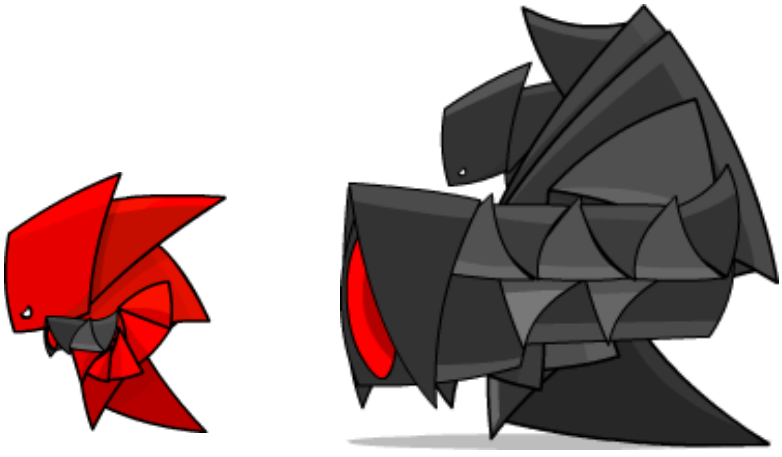


Figura 2.8 – Nemici di gioco

L'alieno rappresenta il nemico da cui il giocatore dovrà difendersi. Più istanze di questo oggetto verranno generate contemporaneamente nel corso del livello e popoleranno la scena aumentando il grado di difficoltà. Gli alieni sono provvisti di un'arma con la quale attaccano il giocatore generando anch'essi oggetti di tipo proiettile.

Scienziato

Figura 2.9 – Prigioniero da liberare

Lo scienziato funge da supporto al giocatore, se liberato, ripristina i danni subiti. Quest'oggetto verrà generato dall'istanza del nemico designato prima che questi venga rimosso dalla scena poiché abbattuto dal giocatore.

Cassa di armi

Figura 2.10 – Cassa

Se distrutta fornisce al giocatore una nuova arma con munizioni limitate. Diverse istanze dell'oggetto verranno generate in vari momenti del livello.

Punti DNA

Ogni nemico colpito fornirà al giocatore un certo numero di punti DNA, la somma dei punti ottenuti a fine partita determinerà il punteggio ottenuto. I punti assegnati da ogni nemico varieranno in base al tipo, ad esempio un boss fornirà un maggior numero di punti.

Sfondo

Lo sfondo rappresenta graficamente il livello corrente. Può essere formato da una o più immagini e sarà posizionato dietro agli oggetti sopra elencati. Dovrà essere possibile caricare sfondi diversi a seconda del livello selezionato.

2.3 Problematiche progettuali

2.3.1 Level Director

Per poter decidere quando generare cosa, e quindi dare forma al livello, è necessario un sistema più o meno complicato che ci permetta di delineare una linea temporale in cui far comparire a schermo i diversi oggetti che comporranno la scena durante il proseguimento del gioco.

Il Level Director sarà l'entità che si occuperà di definire questi eventi, avrà quindi accesso a determinati oggetti in modo da poterli generare.

Nel caso del videogame che si andrà a sviluppare, il proseguo del livello è definito tramite l'uso di contatori temporali che per un determinato lasso di

tempo genereranno un numero di nemici ad una certa frequenza, quello che in gergo viene chiamato “enemy spawn”.

Oltre ai nemici, dovranno essere generati altri oggetti, quali le casse di armi, gli scienziati ed i boss.

Il problema da dover affrontare in questo caso era se rendere la generazione degli oggetti in modo casuale, oppure se definire a priori il design del livello. Tale scelta incide non poco sull'esperienza di gioco, in quanto uno sviluppo casuale può creare situazioni sempre diverse ma al tempo stesso non garantisce sempre una certa sequenza di eventi in grado di definire una difficoltà crescente.

Si è scelto quindi di accantonare la casualità in favore di una definizione ben precisa degli eventi che accadranno nel corso del livello. In particolare si è deciso di suddividere il livello per “ondate” di nemici.

Un'ondata sarà rappresentata dal numero di nemici da generare, la loro tipologia, la loro frequenza di “spawn” in secondi, un flag per rappresentare la generazione di una cassa e il tipo di arma che conterrà, un flag per rappresentare la presenza di uno scienziato da liberare e un delay prima dell'ondata successiva.

L'insieme delle ondate comporrà il livello. I particolari riguardo lo sviluppo saranno espressi nel capitolo riguardante l'implementazione.

2.3.2 Nemici semplici e boss di livello

Un altro problema su cui si è ragionato è la gestione delle diverse tipologie di nemici, in particolare la distinzione tra nemici semplici, quelli più frequenti, e i boss, nemici particolarmente difficili da battere e con peculiarità diverse rispetto ai normali nemici da affrontare dal giocatore.

Si è optato anche qui per una soluzione che permettesse una maggior varietà all'esperienza di gioco, quindi ogni tipologia di nemico verrà gestita da un'entità

dedicata le quali però erediteranno le caratteristiche comuni, quali ad esempio la gestione dello sparo in direzione della torretta difensiva, i punti vita e il conteggio del punteggio, da una macro entità enemy.

Non vi è quindi una distinzione significativa a livello progettuale tra boss e nemici più comuni.

2.3.3 Controlli utente

In videogiochi di questo tipo spesso ci si trova davanti a controlli utente classici, ovvero un joypad digitale a schermo per il controllo del personaggio principale e un altro per il controllo della mira e dei tasti a schermo per sparare. Tale approccio però non è sempre condiviso da tutti i video giocatori in quanto non è altro che un'emulazione di quello che sono i controlli tramite joypad e quindi dotati tasti fisici con un feedback tattile per cui l'utente non avrà mai il dubbio di non aver premuto il tasto al contrario del touch screen dove non vi è alcun feedback.

Pur essendo comunque una soluzione largamente adottata dalla gran parte dei videogames per smartphone e tablet, si è deciso di rendere i controlli utente più intuitivi e meglio attinenti al tipo di tecnologia utilizzata avendo a disposizione un intero schermo interattivo.

Si avrà quindi un’area adibita al controllo della torretta, con un tap prolungato, e un'altra alla mira e allo sparo poiché il proiettile seguirà le coordinate del tap del giocatore.

2.3.4 Armi utente

Altro argomento da affrontare è stato la gestione delle diverse armi utilizzabili dalla torretta e quindi dall'utente. Come per i nemici si è deciso di

definire una macroclasse Weapon che definirà le principali caratteristiche comuni a tutte le altre come il calcolo della direzione del proiettile.

Ogni arma sarà quindi un'istanza dell'oggetto Weapon e permetterà un'implementazione dedicata in modo da attribuirne un diverso comportamento.

3
IMPLEMENTAZIONE

Nelle prossime pagine verrà illustrato il processo di implementazione del caso di sviluppo sopra descritto. Verranno presi in esame i diversi strumenti utilizzati e le alternative su cui la scelta sarebbe potuta ricadere spiegandone le ragioni.

Anche la scelta della piattaforma di riferimento per cui sviluppare il videogame merita di essere presa in esame, in quanto determinante per l'ambiente di sviluppo e per la determinazione del target di utenza a chi si potrebbe proporre l'applicativo.

Una volta appurati gli strumenti, verranno descritte le varie fasi di sviluppo a partire dalla realizzazione delle componenti grafiche fino ai segmenti di codice più importanti che delineeranno la struttura dell'applicazione.

3.1 Software e Tecnologie utilizzate

3.1.1 Piattaforma di riferimento

Una delle scelte più importanti da prendere prima di dedicarsi all'implementazione di un'applicazione è quella della piattaforma di riferimento per cui svilupparla.

Il sistema operativo è determinante nella scelta dell'ambiente di sviluppo e del motore grafico da utilizzare.

Come precedentemente descritto, i principali sistemi operativo mobili presenti sul mercato sono Android, iOS e Windows Phone, rispettivamente in ordine di distribuzione.

La scelta più ovvia potrebbe essere quella di implementare l'applicazione per Android, data la fetta di mercato su cui si affaccia, considerando in un secondo momento un porting sulle altre due piattaforme.

La strategia utilizzata con maggior frequenza però era del tutto contraria. Lo sviluppo veniva fatto su iOs e successivamente se i numeri di download, e quindi la popolarità dell'app, lo consentivano si portava il software anche su Android. Questo avveniva perchè la piattaforma di riferimento era quella Apple, sia per prestazioni che per fetta di mercato.

Questo trend strategico è via via mutato. Negli ultimi anni il sistema operativo di Google ha trovato spazio in tantissimi dispositivi e il progredire della tecnologia lo ha reso un diretto concorrente ad iOS.

Stesso discorso vale per il sistema operativo di casa Microsoft che, anche se più lentamente del previsto, sta avendo una diffusione non trascurabile soprattutto per le performance ottime su dispositivi di fascia medio-bassa.

Occorre anche prendere in esame i diversi ambienti di sviluppo. Nel caso di Android tutti gli strumenti sono scaricabili gratuitamente e fruibili sia su Windows, sia per Mac che su Linux.

Per quanto riguarda iOS, invece, è possibile sviluppare applicazioni solo su macchine con OSX installato, è necessario quindi un Mac. Anche per Window Phone lo sviluppo è vincolato, è necessaria una macchina Windows per poter utilizzare le SDK di riferimento.

Nel caso di sviluppo, oggetto di tesi, si è deciso di cimentarsi nell'implementazione di codice fruibile su più piattaforme. Esistono diversi ambienti di sviluppo e motori grafici capaci di rendere lo stesso codice eseguibile su più dispositivi con sistemi operativi differenti con tutti i pro e i contro del caso di seguito descritti.

Applicazioni Multi piattaforma, pro e contro

Un'applicazione mobile nativa fornisce agli utenti un'esperienza interattiva più ricca ed affidabile. Inoltre, essendo sviluppata con il linguaggio ufficiale, sfrutta al meglio le funzionalità che il dispositivo dispone.

L'utilizzo di strumenti per la creazione di software cross-platform non sempre garantisce le stesse prestazioni di una scritta in linguaggio nativo. Bisogna considerare anche il tipo di applicazione che si va a sviluppare. Nel caso di videogame non sempre è possibile riscrivere del codice per ogni tipo di piattaforma, soprattutto nel caso di applicazioni per il mondo mobile, dove i team di sviluppo sono molto ridotti nella maggior parte dei casi.

Scrivere quindi del codice, fruibile poi su altre piattaforme risulta essere un buon compromesso laddove l'applicazione non debba avere grosse pretese e non debba avere accesso a funzionalità particolari e specifiche per determinati dispositivi.

Per la realizzazione di videogiochi, lo sviluppo cross-platform, è una formula largamente utilizzata nel mondo mobile proprio per l'eterogeneità del mercato.

3.1.2 Corona SDK

Un game engine è un sistema progettato appositamente per lo sviluppo di videogiochi. Comprende generalmente al suo interno un motore di rendering 2d o 3d per la grafica, un motore fisico capace di simulare un ambiente soggetto a gravità e di rilevare eventuali collisioni tra i diversi oggetti e altri strumenti per la realizzazione di suoni, animazioni e la gestione delle diverse scene di gioco.

Corona è un kit di sviluppo software progettato appositamente per lo sviluppo cross-platform, particolarmente indicato nella realizzazione di videogame in 2d. Il linguaggio di riferimento è il Lua.

Corona non dispone di un editor grafico in cui sia possibile trascinare oggetti sulla scena, ma mette a disposizione un emulatore capace di eseguire il codice e quindi dare la possibilità allo sviluppatore di visualizzare la resa finale senza dover testare il codice su un apposito dispositivo.

Il motore fisico utilizzato è il noto Box2D. Box2D è un physic engine open source per simulazioni di fisica in due dimensioni. Questo motore fisico scritto in C++ è indipendente dalla piattaforma. Include al suo interno il rilevamento delle collisioni. Può gestire solo la fisica dei corpi rigidi.

Corona SDK è un software proprietario disponibile in diverse licenze: Starter, Basic, Pro e Enterprise.

La licenza Starter in particolare, permette di creare l'applicazione e pubblicarla gratuitamente e senza vincoli per quanto riguarda icone e link da dover forzatamente visualizzare nella schermata iniziale. Non permette però l'implementazione di acquisti in-app, l'uso di plugin di terze parte illimitato e altri strumenti messi a disposizione per il monitoraggio e l'analisi. E' necessaria una connessione ad internet per la compilazione dell'applicazione avviabile sul dispositivo fisico.

Tutte le licenze danno la possibilità di esportare il proprio codice per Android e per iOs ma anche per il Kindle Fire di Amazone e l'ebook reader Nook prodotto da Barnes & Noble. E' prevista l'esportazione anche per Windows Phone per l'anno 2014.

Non è possibile utilizzare altri linguaggi oltre al Lua e nemmeno bypassare le api messe a disposizione dall'SDK che non sempre danno la possibilità al programmatore di capire come esse operano essendo di alto livello.

E però presente una buona documentazione e un alto numero di tutorial che, uniti al linguaggio, consentono una curva di apprendimento molto ripida capace di velocizzare notevolmente i tempi di sviluppo.

Elementi principali del motore di gioco

Display Object - Tutte le componenti grafiche che appariranno a schermo non sono altro che istanze della classe Display Object. Possono essere raggruppate in *display group* e gestite in blocco. Per ogni scena farà riferimento un display group contenente tutte le sprites necessarie.

Storyboard - Lo storyboard è la soluzione interna implementata da Corona per la gestione, creazione e navigazione delle scene. Le scene non sono altro che le diverse schermate dell'applicazione.

Sono l'oggetto principale delle API dello Storyboard, dispongono di diversi *event listener* che rispondono a specifici eventi. Lo *scene object* ha un unica proprietà *scene.view* ovvero il riferimento al display group associato alla scena.

In questo caso è possibile individuarne diverse: una per la schermata principale, una per la selezione del livello e la scena di gioco principale.

Event Listener - Gli event listener sono usati per creare l'interazione tra l'utente e l'applicazione. Sono usati ad esempio per rilevare il tap sullo schermo ed altre fasi durante il tocco sullo schermo, quali il momento di inizio e di fine.

Physics Library - Come accennato precedentemente, Corona fa uso del motore fisico Box2D. La libreria in poche righe di codice permette di creare oggetti multipli soggetti a gravità, rilevare le collisioni, applicare joints, raycast e altro.

Tables - Le tables sono le uniche strutture dati messe a disposizione dal linguaggio Lua. Implementano array associativi i quali possono essere indicizzati sia tramite numeri interi, sia attraverso altri valori escluso *nil*. Le tables possono essere eterogenee, quindi contenere diversi tipi di variabili e possono essere usate per rappresentare, array, tabelle, grafi e alberi.

Ambiente di sviluppo

Una volta scaricato ed installato il pacchetto di installazione di Corona SDK sul calcolatore sarà possibile avviare il software dedicato e sul desktop apparirà una situazione di questo tipo:

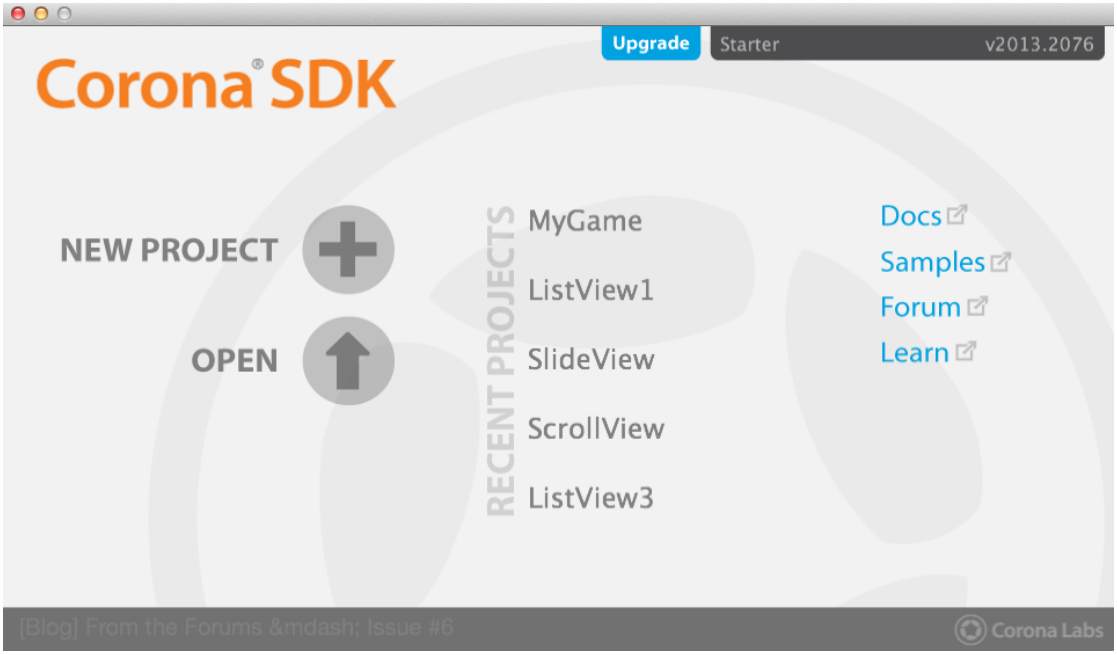


Figura 3.1 – Schermata principale di Corona

Da questa schermata è possibile creare un nuovo progetto ed aprire quelli precedentemente creati.

All'apertura del progetto il codice verrà eseguito su un emulatore, è possibile decidere che tipo di emulatore utilizzare per testare la resa del progetto su diversi tipi di dispositivi con risoluzioni diverse.

Figura 3.2 – Emulatore iPhone 5

Figura 3.3 – Emulatore Samsung Galaxy S3

Corona non dispone di un IDE interno ma permette l'utilizzo di editor testuali di terze parti. In questo caso si è fatto uso di Sublime Text 2, un noto editor per sistemi Mac molto flessibile su cui è possibile installare diversi plug-in per il riconoscimento del linguaggio di programmazione utilizzato. In questo caso Corona ne mette a disposizione uno dedicato per Lua ben realizzato e che facilita la stesura del codice.

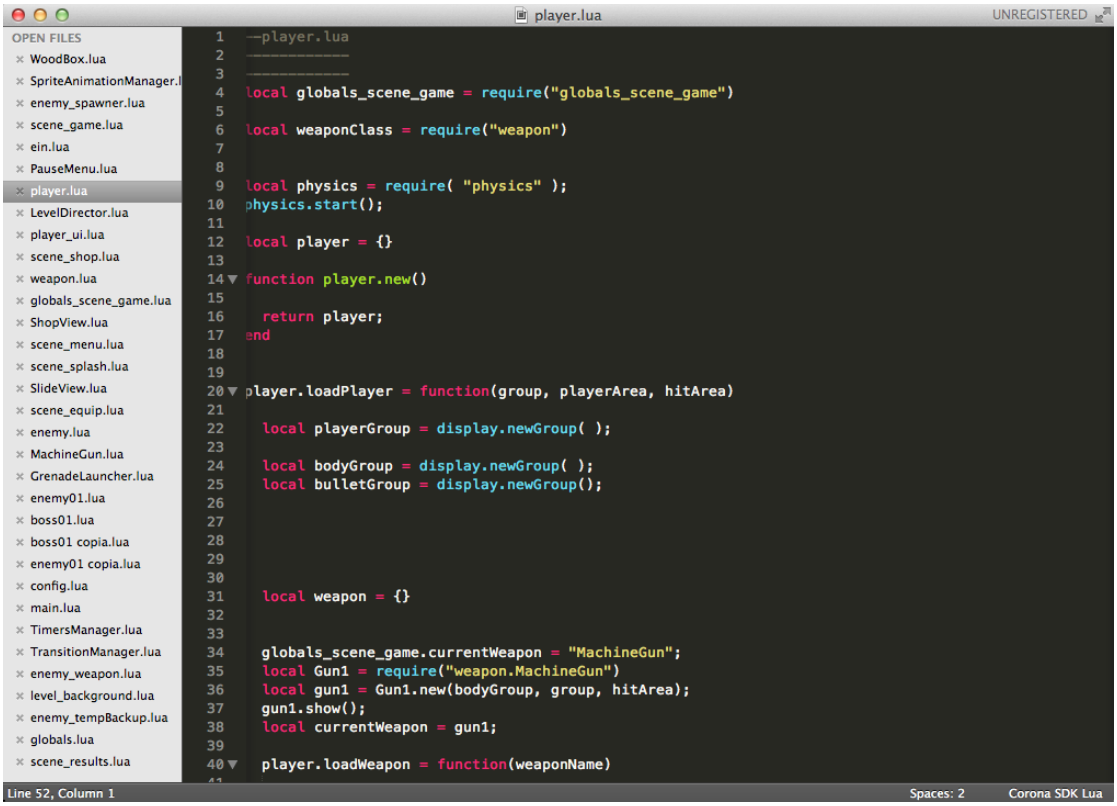


Figura 3.4 – Sublime Text 2

3.1.3 Risorse grafiche

Per la realizzazione delle risorse grafiche quali, sprites, animazioni, sfondi e logo, si è fatto uso di importanti strumenti Adobe come Flash CS6 e Photoshop CS6. Nei videogame la componente grafica è molto importante e un buon approccio è determinante per la resa finale del progetto.

Adobe Flash CS6

Flash è un software per uso prevalentemente grafico che consente di creare animazioni vettoriali principalmente per il web. Viene utilizzato inoltre per creare giochi o interi siti.

Una caratteristica importante di questo prodotto consiste nella possibilità di raggruppare insiemi di elementi vettoriali in simboli riutilizzabili poi in una libreria. Esistono diverse tipologie di simboli, fra i quali semplici oggetti grafici, clip filmato (sotto-animazioni con una timeline propria e indipendente da quella principale), pulsanti, oggetti bitmap, ecc

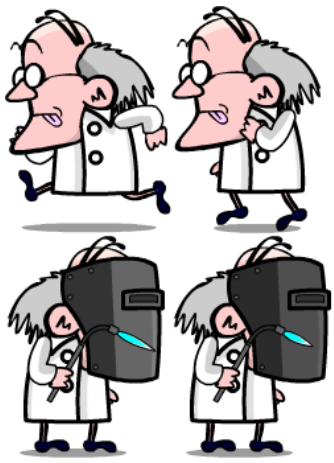


Figura 3.5 – Esempio di spritesheet di gioco

Nel caso dei nemici si è scelto di comporre la loro rappresentazione grafica lato codice per permettere una maggior flessibilità ed un maggior controllo. E' stato possibile, ad esempio, far in modo che la testa dell'alieno si inclinasse a seconda della posizione della torretta dando l'idea di contatto visivo. Stessa cosa per l'arto imbracciante l'arma.

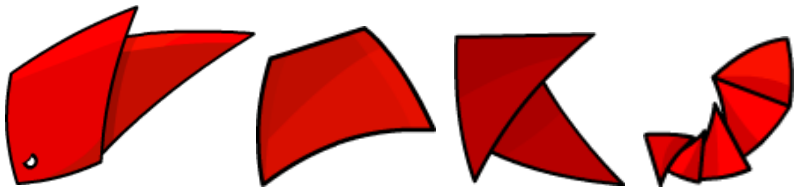


Figura 3.6 – Elementi grafici che compongono l'alieno

Per la torretta difensiva si è proceduto analogamente, è composta infatti da diverse componenti grafiche per permettere l'inclinazione, l'innalzamento e l'aggiunta di diverse armi.

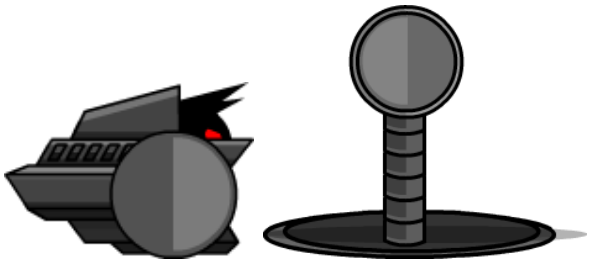


Figura 3.7 – Elementi grafici che compongono la torretta

Adobe Photoshop CS6

Photoshop è un software proprietario prodotto dall'Adobe Systems Incorporated specializzato nell'elaborazione di fotografie (fotoritocco) e, più in generale, di immagini digitali. E' in grado di effettuare ritocchi di qualità professionale alle immagini, offrendo enormi possibilità creative grazie ai

numerosi filtri e strumenti che permettono di emulare le tecniche utilizzate nei laboratori fotografici per il processamento delle immagini, le tecniche di pittura e di disegno. Un'importante funzione del programma è data dalla possibilità di lavorare con più "livelli", permettendo di gestire separatamente le diverse componenti che costituiscono l'immagine principale. È un software di grafica raster e in quanto tale si differenzia dai software di grafica vettoriale poiché agisce direttamente sui pixel. Nel caso specifico è stato utilizzato in coppia con Flash per la realizzazione di alcuni sfondi, icone e per l'immagine iniziale di gioco.

3.2 Principali aspetti implementativi

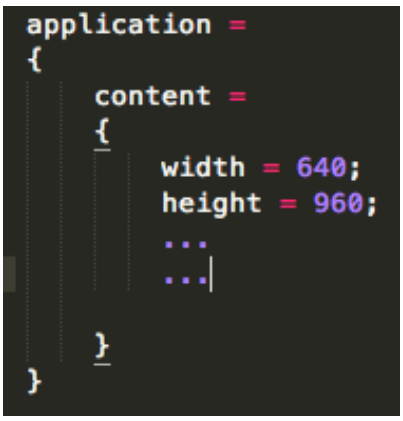
Nelle prossime pagine verranno proposti i principali segmenti di codice sviluppati con la relativa descrizione e funzione all'interno dell'applicazione. Verranno individuate le classi principali e quelle di supporto per alcune funzioni. Si analizzerà l'uso delle Scene messe a disposizione da Corona per rappresentare le diverse schermate di gioco e la realizzazione delle principali.

3.2.1 Classi principali

Configurazione

Il file *config.lua* è necessario per determinare le specifiche di configurazione dell'applicazione. A seconda dei parametri che si andranno ad inserire si potrà definire il comportamento che assumerà il progetto su dispositivi con differenti risoluzioni.

Sono stati impostati dei valori di default per quanto riguarda la grandezza e la larghezza:



Il valore *scale* definirà il metodo di scanalatura dello schermo in modo da adattarlo a tutte le risoluzioni:

Letterbox – il contenuto viene scalato preservando lo stesso aspect-ratio. L'intero contenuto risiederà nello schermo e non verrà tagliato ma per alcune risoluzioni può accadere di avere della bande nere ai lati che però possono comunque essere occupate, ad esempio, da immagini di sfondo.

ZoomEven – il contenuto viene scalato per aderire allo schermo e mantenendo lo stesso aspect ratio. Può accadere però che alcuni contenuti non vengano visualizzati correttamente e che finiscano fuori dai bordi dello schermo.

ZoomStretch – il contenuto viene fatto aderire allo schermo in modo da occuparlo interamente. E' il metodo di default anche se poco raccomandato in quanto i contenuti possono venire distorti orizzontalmente e verticalmente.

In questo caso si è optato per il metodo “letterbox”, il più diffuso e più raccomandato. Con una buona programmazione si può ovviare al problema delle bande nere ed il risultato è più che soddisfacente.

Main

Main.lua è il file da cui partirà l'esecuzione del codice, al suo interno possono essere inizializzate alcune variabili globali ed in questo caso, attraverso lo storyboard, si è fatto in modo di navigare automaticamente fino alla scena iniziale del gioco:

```
-- Start up Storyboard
local storyboard = require( "storyboard" )
storyboard.gotoScene( "SceneMenu" )
```

Globals

E' stata definita una classe per le variabili globali. Tali variabili saranno utili per la memorizzazione temporanea di alcuni valori che dovranno essere raggiungibili da diverse classi. Ad esempio è stata inserita una variabile che assumerà un valore a seconda del livello selezionato e una per il punteggio corrente del giocatore.

Per poter utilizzare queste variabili ogni classe non dovrà far altro che dichiarare un oggetto *globals*:

```
-- Initialize our global variables
local globals = require( "globals" )
globals.levelNum = 1
globals.score = 0
```

SceneMenu

La classe SceneMenu definisce la schermata di menu iniziale. Il codice implementato permette di visualizzare un'immagine iniziale con il logo e il nome del videogame.

Premendo sul bottone Play, la classe attraverso lo storyboard e il relativo metodo di navigazione, visualizzerà la scena di selezione livello (SceneLevels).

SceneLevels

E' la classe che gestisce la scena di selezione del livello, implementa una slide view appositamente creata per visualizzare le “copertine” dei diversi livelli. Con uno slide verso destra o sinistra si naviga tra i livelli. Una volta selezionato, viene registrata la variabile globale corrispondente e sempre attraverso lo storyboard viene visualizzata la scena di gioco.

SceneGame

SceneGame è la classe principale, implementa la scena di gioco e si occupa di caricare tutte le classi necessarie al suo funzionamento.

Al momento della creazione della scena, si occuperà di inizializzare il LevelDirector, la classe Player, la classe relativa alla UI di gioco e la generazione del background a seconda del livello selezionato.

Inoltre si occuperà di creare le due aree per l'input dell'utente. Un'area a sinistra per il controllo della torretta e un'area più grande a destra per lo sparo.

```
local playerArea = display.newRect( display.screenOriginX,
                display.contentCenterY, screenWidth/3.5, screenHeight)
  playerArea.isVisible = false;
  playerArea.isHitTestable = true;


  group:insert(playerArea);

  local hitArea = display.newRect( display.screenOriginX + playerArea.width,
                  display.contentCenterY+80, screenWidth-playerArea.width, screenHeight)
  hitArea.isVisible = false;

  hitArea.isHitTestable = true;
```

Le due aree non sono altro che due display object di tipo rettangolo, non visibili ma comunque interagibili. La gestione dei relativi Event Listener è stata relegata alla classe Player che si occuperà di definire i relativi comportamenti.

Player

E' la classe principale per la gestione della torretta e del suo funzionamento. Come precedentemente accennato si occupa di implementare i listener per l'input utente.

Si è deciso di gestire la torretta come un corpo fisico, di tipo “dynamic (soggetto a gravità), per la gestione dell'innalzamento. A seguito del tap prolungato vengono generati diversi impulsi dall'alto verso il basso sulla torretta in modo da spingerla verso l'alto.

```
playerArea:addEventListener( "touch", function( event )
  ...
if (event.phase == "began")then


          upPlayer();

          timPlayer = timer.performWithDelay(1, upPlayer, 0);

        end

        if (event.phase == "ended" or event.phase == "cancelled")then
          ...
          timer.cancel(timPlayer)
          ...

        end
end

function upPlayer()
    bodyGroup:applyLinearImpulse( 0, -1000, bodyGroup.x, bodyGroup.y )
  end
```

All'inizio del tap dell'utente viene generato il primo impulso e creato un timer che ad ogni frame in maniera ciclica genera una serie di altrettanti impulsi che man mano faranno alzare la torretta. Al momento del rilascio del dito da parte dell'utente il timer viene cancellato e la torretta per via della gravità torna verso il basso. Sono stati imposti dei limiti fisici per evitare che la torretta finisse oltre i limiti di gioco.

Per la gestione delle collisioni è stato creato un altro corpo fisico, invisibile di forma circolare ancorato alla torretta e di tipo “kinematic” quindi non soggetto a gravità. Sarà questo corpo a registrare le collisioni e quindi rilevare i danni subiti. Questo approccio è utile al fine di avere una visualizzazione dell'area di collisione e di poterla controllare escludendo alcuni elementi della torretta. In sostanza solo la parte centrale, esclusa l'arma sono oggetto di collisione da parte dei colpi nemici.

Figura 3.8 – Area di collisione della torretta

La classe espone anche un metodo per il caricamento della classe relativa all'arma da equipaggiare.

In un primo momento viene caricata l'arma predefinita, la MachineGun. Nel momento in cui, ad esempio dopo la distruzione di una cassa di armi, debba essere sostituita con una nuova arma, l'arma precedente viene nascosta e sostituita con la nuova. Quando le munizioni della nuova arma finiscono viene caricata di nuovo quella predefinita.

Per quanto riguarda lo sparo, la gestione dell'event listener relativo all'area per l'input utente viene rimandata alla classe dell'arma corrente, la quale si occuperà effettivamente di generare il proiettile e di calcolare la direzione.

Weapon

La classe Weapon si occupa della gestione delle armi a disposizione dell'utente. Contiene alcuni valori di default, quali: frequenza di sparo ed entità dei danni.

Implementa l'Event Listener relativo all'area di input dello sparo. Si occupa quindi di calcolare a seconda delle coordinate del tap l'angolazione da attribuire

alla torretta e al proiettile in modo che segua tali coordinate. Lo sparo genererà un proiettile, un corpo fisico di tipo *Bullet* messo a disposizione da Corona in quanto ottimizzato per il rilevamento continuo delle collisioni senza pregiudicare le prestazioni, che seguirà appunto la direzione del tap dell'utente nella relativa area.

MachineGun

Il linguaggio Lua non mette a disposizione un vero e proprio metodo per estendere le classi e per l'override dei metodi. E' possibile però simulare tale comportamento.

La classe MachineGun è inizializzata come oggetto di tipo Weapon, quindi ne eredita tutte le proprietà sopra descritte. E' stato possibile a questo punto estendere le sue caratteristiche e modificare quelle di default. In particolare dovendo simulare una mitragliatrice è stato gestito il tap prolungato dell'utente con la conseguenza di generare proiettili a ripetizione. Inoltre è stato implementato il sistema di surriscaldamento descritto nel capitolo della Progettazione, è stato creato un contatore che viene incrementato ad ogni sparo, nel caso arrivi ad un valore prestabilito (in questo caso 100) impedisce all'arma di generare altri proiettili finché l'utente non rilascia il dito dallo schermo. Al momento del rilascio il contatore decrementa progressivamente ad una certa velocità fino a tornare al suo valore iniziale di zero. Se durante il decremento l'utente ricominciasse a sparare, il contatore incrementerebbe di nuovo dal valore fin ora raggiunto.

Enemy

La classe Enemy rappresenta la classe di riferimento da cui le subclassi di nemici erediteranno le proprietà più comuni. Implementa alcune variabili

comuni quali i punti vita del nemico, i punti DNA che fornirà dopo la sua uccisione, la frequenza di sparo ecc...

Si occupa di calcolare l'angolazione del nemico rispetto alla posizione della torretta per la direzione dello sparo.

EnemySoldier

EnemySoldier è una classe di tipo Enemy, implementa il codice per la gestione del nemico più comune. Graficamente è composto da più elementi grafici:

```
enemy.eLegs = display.newImage( enemy, "assets/enemy/enemy01_legs.png" );
enemy.eBody = display.newImage( enemy, "assets/enemy/enemy01_body.png" );
enemy.eHead = display.newImage( enemy, "assets/enemy/enemy01_head.png" );

local enemyArm = display.newGroup();
enemyArm.eArm = display.newImage( enemyArm, "assets/enemy/enemy01_arm.png" );
local weapon = display.newImage( enemyArm, "assets/enemy/enemy_weapon01.png" )
```

L'animazione viene gestita da codice come anche l'inclinazione della testa a seconda della posizione della torretta per simulare un contatto visivo. Come per la classe Player, è stato creato un corpo fisico, denominato *collisionBound* per determinare l'area di collisione del nemico.

Figura 3.8 – Area di collisone del nemico

EnemyBoss

Anche questa classe è di tipo Enemy ed in questo caso implementa gli stessi comportamenti di EnemySoldier, salvo per le risorse grafiche, la frequenza di sparo e l'entità dei danni. Il vantaggio di utilizzare un collisionBound indipendente dalla sprite permette di determinare diversi punti di collisione. Per questo tipo di boss, la collisione è stata abilitata solo in prossimità della testa per aumentare il livello di difficoltà.

Figura 3.9 – Area di collisione del boss, situata in prossimità della testa

LevelDirector

Il LevelDirector è la classe che gestisce la generazione dei nemici sullo schermo e quindi il proseguo del gioco. A seconda del livello selezionato viene caricato il relativo file JSON contenente l'insieme delle ondate con le relative variabili. In particolare ogni ondata sarà formata dalle seguenti informazioni:

wSize – il numero di nemici da generare;
spawnFreq – la frequenza con cui generare i nemici;
enemyType - il nome della classe da generare;
prisoner – se true viene generato il prigioniero da liberare;
woodBox – se true viene generata la cassa di armi contenente l'arma appartenente alla classe specificata;
DelayToNextWave – il tempo che deve trascorrere prima della prossima ondata.

Una volta fatto il parsing del file viene generata una table come struttura dati per contenere le informazioni relative ad ogni ondata. Si avrà quindi una situazione di questo tipo:

```
local levelPattern = {

    {
      wSize = 5,
      spawnFreq = 800,
      enemyType = "EnemySoldier",
      prisoner = false,
      woodBox = {true, "GrenadeLauncher"},
      delayToNextWave = 2000
    },

    {
      wSize = 10,
      spawnFreq = 500,
      enemyType = "EnemySoldier",
      prisoner = true,
      woodBox = {false, nil},
      delayToNextWave = 2000
    },

    ...
    ...
    ...
```

A questo punto la classe si preoccuperà di attivare i timer per la gestione della generazione dei diversi oggetti. I timer saranno settati a seconda delle informazioni contenute nel levelPattern:

```
function mainTimer()


  if(levelPattern[LevelDirector.currentWave].wSize > 0)then
    levelPattern[LevelDirector.currentWave].wSize = levelPattern[LevelDirector.currentWave].wSize -1;
    local mainTimer = globals_scene_game.timersManager:add( levelPattern[LevelDirector.currentWave].spawnFreq,
                      function() mainTimer(); spawnEnemy(); end, 1);
  elseif(LevelDirector.currentWave < LevelDirector.nWave)then

    --Next Wave
    globals_scene_game.timersManager:add( levelPattern[LevelDirector.currentWave].delayToNextWave,

      function()
        LevelDirector.currentWave = LevelDirector.currentWave + 1;
        mainTimer()
      end,
      1);

  else
    print("WAVES ENDED")
  end
end

--First Timer
mainTimer();
```

3. 3 Problematiche implementative riscontrate

3.3.1 Pausa durante l'esecuzione della scena di gioco

Corona non mette a disposizione un sistema per l'implementazione della pausa di gioco. E' possibile soltanto sospendere l'esecuzione della fisica. Per la gestione delle transazioni, dei timer e dell'animazione delle sprite è stato necessario implementare delle classi per poter fermare la loro esecuzione durante la pausa e riprenderla in un secondo momento.

In particolare sono state implementate tre classi: *TransitionManager, TimerManager, SpriteAnimationManager*.

Grazie a queste classi e alla possibilità di sospendere l'esecuzione dell'ambiente fisico, e quindi soggetto a gravità, è stato possibile mettere in pausa il gioco per poi riprenderlo.

Inoltre, sono risultate utili per rimuovere dalla memoria tutti gli oggetti relativi ad una determinata istanza di gioco, così da poter ottimizzare le prestazioni al momento dell'esecuzione di una successiva partita da parte dell'utente.

TransitionManager

TransitionManager è la classe che si occupa di sospendere le transazioni. Le transazioni sono dei metodi messi a disposizione da Corona per gestire alcune animazioni per i display object. Ad esempio è possibile determinare una transazione da un punto x ad un altro per un determinato periodo di tempo.

Non essendo disponibile un metodo per mettere in pausa le transazioni sia singolarmente che globalmente è stato necessario implementarli.

Il riferimento ad ogni transazione viene salvata in una table dalla classe al momento della creazione; ovviamente l'istanziazione delle singole transazioni

non avviene più direttamente ma attraverso il TransitionManager con un metodo dedicato in quale si occuperà di salvarne tutti i parametri tra i quali lo stato iniziale e quello finale.

Il metodo *pauseAll* scorre la table e per ogni transazione salva lo stato attuale e lo stato finale. Mentre il metodo *resumeAll*, scorrendo sempre la table, istanzierà una nuova transazione con lo stato iniziale uguale a quello attuale precedentemente salvato.

Il metodo *removeAll* rimuove tutte le transazioni presenti in tabella.

TimerManager

Per i timer, essendo implementato un metodo per sospenderli e riprenderli singolarmente, è bastato gestire una table contenente tutti i riferimenti ai timer creati. In questo modo è possibile attraverso i soliti metodi *pauseAll* e *resumeAll* sospendere tutti i timer e riprenderli contemporaneamente.

SpriteAnimationManager

Anche per quanto riguarda le animazioni delle sprites è stato necessario realizzare una struttura, una table, per salvarne i riferimenti. Corona mette a disposizione dei metodi per sospendere l'animazione e per riprenderla. Nel caso specifico della pausa globale della scena di gioco si è dovuto operare come per i timer.

3.3.2 Collision filtering

Di default in Corona tutti gli oggetti fisici collidono tra di loro. L'engine mette però a disposizione uno strumento per decidere con quali oggetti ognuno di essi debba collidere.

Il Collision Filtering permette proprio questo. Durante la creazione dell'oggetto fisico è possibile associare una coppia di valori *categoryBits* e *maskBits* per filtrare la rilevazioni delle collisioni da parte dell'oggetto. Più propriamente un oggetto si scontrerà con altri solo se i loro categoryBits sono tra i maskBits assegnatoli.

Ad esempio per il collisionBound del proiettile del giocatore è stata attribuito un collisionFilter con categoryBits 16 , in questo modo la collisione verrà rilevata soltanto con il collisionBound del nemico in quanto possiede un maskBits di 16.

3.3.3 Gestione della memoria

Uno dei vantaggi, ma al tempo stesso degli svantaggi, di Corona è il suo mettere a disposizione delle API di alto livello. Questo velocizza si la stesura del codice ma non permette sempre di avere una gestione ottimale di quello che avviene nei livelli sottostanti. L'eliminazione delle variabili non più utilizzate viene gestita tramite un garbage collector.

L'uso di variabili dichiarate *local* è una buona prassi da utilizzare per la salvaguardia delle prestazioni, come anche la rimozione dal *display hierarchy* degli oggetti non più necessari all'applicazione. Se la rimozione non dovesse avvenire non è garantito che il garbage collector provveda a rimuoverli in quanto potrebbero essere riferiti da altri oggetti.

Per quanto riguarda le diverse immagini, Corona si occupa di gestire la cache delle texture. Per non compromettere l'utilizzo della memoria, le immagini con lo stesso filename non vengono salvate nuovamente, così da evitare ripetizioni e quindi uno spreco inutile.

Per quanto riguarda i proiettili, ad esempio, anche se il motore mette a disposizione una classe dedicata per la loro gestione si è scelto di attribuire un

tempo di vita ad ognuno di essi. Nel caso della MachineGun può presentarsi il caso in cui i proiettili generati siano in grande quantità e non tutti vengano rimossi dalla memoria a seguito di una collisione, poiché se non avviene continuerebbero a "viaggiare" fuori dai limiti dello schermo fino a che non verranno rimossi automaticamente dall'engine. Tramite un tempo di vita per ognuno di essi si è potuto assicurarne la rimozione entro un determinato periodo così da poter alleggerire la memoria.

Anche l'utilizzo delle classi per la gestione della pausa ha permesso di eliminare, qualora non fossero più necessarie, le transazioni e i timer attivi.

4
CONCLUSIONI E SVILUPPI FUTURI

4.1 Conclusioni

Il mercato dei videogiochi per dispositivi mobili è in piena crescita, ma anche saturo di nuove idee. Non è facile per un'applicazione emergere tra le tante presenti.

In questa tesi si è cercato di sviluppare un videogioco per poter analizzare le potenzialità dello sviluppo multi-piattaforma ma anche per approfondire le conoscenze di progettazione e implementazioni comuni al settore gaming.

Le meccaniche di gioco proposte e l'intuitività dei comandi giocano un ruolo fondamentale sulle possibilità di ottenere un discreto successo per piccoli giochi sviluppati per smartphone e tablet. In questo caso si è cercato di seguire proprio questi presupposti nella fase di progettazione proponendo comandi utente di facile apprendimento.

Anche la scalabilità, può essere decisiva per un buon prodotto di questo tipo. In fase d'implementazione e progettazione si è deciso di sviluppare un codice flessibile e aperto a nuovi sviluppi. Il numero di livelli e la varietà di nemici da proporre al giocatore può essere incrementata senza compromettere l'integrità progettuale del codice.

Gli strumenti utilizzati hanno confermato la validità degli stessi. Corona SDK è uno strumento che velocizza la stesura del codice dando più spazio alla creatività e allo sviluppo di risorse grafiche più ricercate. I test effettuati su diversi dispositivi con diverse risoluzioni hanno mostrato come lo stesso codice sia eseguito perfettamente da piattaforme diverse e, con una buona

programmazione, i contenuti mantengano lo stesso aspetto garantendo un'esperienza utente solida.

Anche se le potenzialità del motore di gioco sul prodotto finale sono evidenti, i limiti imposti dalla versione Starter e il linguaggio di alto livello utilizzato non consentono sempre di utilizzarlo per lo sviluppo di progetti più complessi e composti da team numerosi.

Questo è comunque un limite concettuale intrinseco allo sviluppo cross-platform, anche se data la moltitudine di sistemi operativi presenti sul mercato risulta essere largamente utilizzato.

Scrivere del codice nativo per l'implementazione di applicazioni risulta essere comunque la soluzione più corretta per ottenere il massimo delle prestazioni ed un maggior controllo su cosa accade anche ai livelli sottostanti del linguaggio di programmazione.

4.2 Sviluppi futuri

L'applicazione permette numerose aggiunte e modifiche sia per quanto riguarda la stabilità e la compatibilità, sia per ampliarne le potenzialità e le caratteristiche. Infatti, per limiti di tempo, non tutti gli obiettivi preposti al fine di rendere il videogioco pubblicabile, sono stati raggiunti.

Tra questi uno dei più importanti è sicuramente la condivisioni dei propri risultati con i diversi social network. In particolare sia su Android che su iOS sono presenti piattaforme dedicate, come Play Games e Game Center, che permettono al giocatore di visionare i risultati e le classifiche dei propri amici. Il confronto dei propri punteggi con quelli di altri giocatori sono un ottimo strumento per aumentare le potenzialità di successo di un videogame.

Queste aggiunte non richiedono la stesura di grandi porzioni di codice, i principali social network mettono già a disposizione degli strumenti per integrare le loro funzioni nell'applicazione.

Come già accennato, il progetto permette l'aggiunta di numerosi livelli e varietà di nemici affrontabili dall'utente, in modo da aumentarne la longevità. L'implementazione in questo caso richiederà più tempo in quanto saranno necessarie nuove risorse grafiche e nuove porzioni di codice per definire i comportamenti dei nuovi oggetti. Una buona soluzione potrebbe essere quella di implementarne di nuovi periodicamente tramite aggiornamenti. In questo modo a seconda del successo dell'applicazione si potrà decidere se spendere più risorse per la loro realizzazione.

Anche l'innesto di nuove armi è da prendere in considerazione, come la realizzazione di un negozio dedicato per l'acquisto tramite i punti DNA raccolti. Molto comune in questo tipo di videogiochi è la possibilità di acquistare potenziamenti all'inizio di ogni livello. I potenziamenti non sono altro che oggetti di supporto al giocatore, in modo da aumentare ad esempio i punti vita a disposizione, i danni inferti dall'arma e per aumentare la frequenza di generazione di casse e prigionieri da liberare.

In un sistema di questo tipo il giocatore sarà sempre più invogliato a proseguire il gioco aumentando il prestigio dell'applicazione e quindi le possibilità offerte da prodotto finale.

Altro aspetto importante da considerare è la possibilità di guadagno che potrebbe avere. Si potrebbero aggiungere acquisti in-app, che non sono altro che acquisti tramite denaro reale di punti per acquistare potenziamenti e livelli. Questo tipo di strategie sono molto diffuse sul mercato e possono generare discreti guadagni.

SITOGRAFIA

- Dispositivo mobile – Wikipedia, 2014, *http://it.wikipedia.org/wiki/Dispositivo_mobile*
- Smartphone – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Smartphone*
- Tablet Computer – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Tablet_computer*
- Un miliardo di nuovi smartphone nel 2013 (+38,4%). Traguardo storico per le vendite globali – il Sole 24 Ore, 2013, *http://www.ilsole24ore.com/art/finanza-e-mercati/2014-01-28/un-miliardo-nuovi-smartphone-2013-+384percento-traguardo-storico-le-vendite-globali-samsung-ne-vende-piu-doppio-apple-072421.shtml?uuid=ABtXzis*
- Infographic: 2013 Mobile Growth Statistics – DigitalBuzzBlog, 2013, *http://www.digitalbuzzblog.com/infographic-2013-mobile-growth-statistics/*
- Smartphone games account for 75% of app store spending – MCV 2013, *http://www.mcvuk.com/news/read/3ds-drives-handheld-growth-with-10m-shipment-rise/0128722*
- ARM architecture – Wikipedia, 2014, *http://en.wikipedia.org/wiki/ARM_architecture*
- Android – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Android_(operating_system)*
- iOS – Wikipedia, 2014, http://en.wikipedia.org/wiki/IOS
- Windows Phone – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Windows_Phone*
- App Store – Wikipedia, 2014, *http://en.wikipedia.org/wiki/App_Store_(iOS)*

- Google Play – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Play_Store*
- Windows Phone Store – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Windows_Phone_Store*
- Mobile Game – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Mobile_game*
- Metal Slug – Wikipedia, 2014, *http://en.wikipedia.org/wiki/Metal_Slug*
- Jetpack Joyride – Google Play, 2014, *https://play.google.com/store/apps/details?id=com.halfbrick.jetpackjoyride&hl=it*
- Android API Guides – Android Developer, 2014, *http://developer.android.com/guide/index.html*
- iOS Developer Library – Apple Developer, 2014, *https://developer.apple.com/library/ios/navigation/*
- Corona SDK API Reference – Corona Labs, 2014, *http://docs.coronalabs.com/api/index.html*
- Getting Started with Corona SDK – Corona Labs, 2014, *http://coronalabs.com/resources/tutorials/getting-started-with-corona/*