Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea in Scienze di Internet

L'AMBIENTE ANDROID PER LO SVILUPPO DI GIOCHI

Tesi di Laurea in Programmazione di Internet

Relatore:
Chiar.mo Prof.
ANTONIO MESSINA

Presentata da:
DANIELE D'ADAMO

III Sessione
Anno Accademico 2009-2010

L'Ambiente Android per lo Sviluppo di Giochi

Daniele D'Adamo

Indice

Introduzione

L'obiettivo di questa tesi è di realizzare un gioco per la piattaforma Android, in questa introduzione verrano espresse le motivazioni di tale scelta.

Perché sviluppare per i device mobili?

L'introduzione dei cosiddetti "smartphone"[1] ha cambiato radicalmente la concezione di telefono cellulare, rivoluzionando completamente il modo in cui le persone interagiscono con questi ultimi. Inizialmente concepiti come "semplici" mezzi di comunicazione, oggi sono divenuti congegni dalle molteplici funzioni che incorporano molti altri dispositivi tecnologici come fotocamere, navigatori satellitari, lettori multimediali. Questi dispositivi mobili danno la possibilità alle persone di avere accesso a tutte le informazioni di cui hanno bisogno, in qualunque posto ed in qualunque momento. Grazie alla loro enorme versatilità ed ubiquità e alla loro ampia dotazione di strumenti come microfoni, fotocamere, touchscreen, sistemi di geolocalizzazione e sensori di ogni tipo, gli smartphone stanno diventando a tutti gli effetti delle estensioni delle nostre percezioni.

È stato stimato [30] che entro cinque anni gli accessi al web da dispositivi mobili supereranno quelli effettuati da PC. Il mercato degli smartphone è caratterizzato da una fortissima domanda che coinvolge in maniera indiscriminata tutte le classi di consumatori. Sviluppare applicazioni per dispositivi mobili è indubbiamente un'importante opportunità in quanto permette di posizionarsi nel lato dell'offerta nel mercato sopracitato.

[1] Con il termine smartphone si definisce un telefono cellulare che abbia anche le funzioni e le potenzialità di un computer palmare in grado di funzionare con un sistema operativo.

Perché sviluppare per Android?

Android [25] è una tra le tante realtà che si sono venute a creare durante l'evoluzione dei device mobili, la sua natura aperta priva di barriere lo ha portato rapidamente ad essere tra le piattaforme mobili più apprezzate attualmente.

La parola "open" ruota intorno ad Android in tutti i suoi aspetti, inclusa la sua piattaforma di sviluppo, molto potente e allo stesso tempo estremamente semplice da utilizzare. Gli sviluppatori hanno libero accesso a tutti gli strumenti usati dai creatori stessi di Android, pertanto il potenziale delle applicazioni non è soggetto a limitazioni di alcun tipo, visto anche che non viene fatta discriminazione tra software nativo e di terze parti. "Essere" uno sviluppatore non richiede certificazioni particolari, inoltre le applicazioni non vengono sottoposte a procedure di approvazione.

Perché sviluppare un gioco?

I giochi sono e sempre saranno una grande fonte di intrattenimento, a prescindere dalla loro natura e dal loro grado di elaborazione.

Si è scelto di realizzare un gioco, al fine di ottenere un'applicazione che fosse motivo di intrattenimento, indipendentemente dalla sua pura complessità a livello informatico, solitamente legata all'entità delle risorse di cui si dispone. Spesso realizzare progetti software esclusivamente a scopi didattici, può significare essere consapevoli fin dall'inizio che il frutto di quel determinato progetto non avrà applicazioni pratiche. Dunque in questo caso la speranza è quella di oltrepassare questo limite puntando a realizzare un prodotto che non sia fine a se stesso.

Nella prima parte di questa tesi verrà fatta innanzitutto un'introduzione al mondo dei giochi destinati ai dispositivi mobili (capitolo 1), seguita da una descrizione dei vari aspetti della piattaforma Android (capitolo 2).

Nella seconda parte verrà esposto l'intero processo di sviluppo dell'applicazione "Catching Cherries", il gioco che si è scelto di sviluppare per questa tesi.

Parte I

Capitolo 1

I giochi su device mobili

Nello scenario delle applicazioni per dispositivi mobili, i giochi costituiscono una categoria a parte e meritano quindi un'analisi dedicata. Recenti statistiche[1] dimostrano che le applicazioni per device mobili più scaricate in assoluto sono i giochi, i quali stanno diventando popolari sia tra i giocatori classici che tra i soggetti che non avevano avuto altri contatti col mondo dei videogiochi. Il loro grande successo deriva in parte dal fatto che i giochi per dispositivi mobili sono sempre a portata di mano degli utenti, visto che sono contenuti da dei dispositivi che ci assicuriamo di avere sempre in tasca, dato che li usiamo primariamente per comunicare. Un esempio analogo è costituito dai cosiddetti social game[2], che sfruttano la loro integrazione con i più grandi social network per ottenere un immenso bacino di utenti. Risulta evidente l'importanza di rendere l'accesso a tali prodotti il più semplice e immediato possibile.

In questo capitolo verrà innanzitutto esplorata la lenta evoluzione [59] dei giochi destinati ai device mobili (paragrafo 1.1), dopodiché questi ultimi verrano messi a confronto con i giochi per le piattaforme tradizionali come PC e console (paragrafo 1.2), ed infine verrà analizzato "Angry Birds", uno tra i giochi più affermati nel mondo del mobile gaming (paragrafo 1.3).

[1] Dati provenienti da Distimo [17], tra le più attive società che si occupano di ricerche nel mercato delle applicazioni per device mobili.

[2] Un esempio di social game è FarmVille [26], che viene giocato da decine di milioni di utenti ogni giorno, grazie alla sua integrazione con il social network Facebook.

1.1 Breve storia

Il primo gioco ad apparire su un telefono cellulare fu il classico snake, preinstallato nei telefoni Nokia dal 1997. Questo è sicuramente il più famoso gioco per piattaforme mobili, rappresenta l'intera generazione di giochi per cellulare degli anni novanta, i quali erano monocromatici ed avevano un gameplay[3] minimale.

Nel 2002 vennero rilasciati i primi telefoni con supporto Java, che è tra i primi esempi di piattaforme portabili e permise anche l'implementazione di animazioni molto più avanzate di quelle disponibili all'epoca.

Con la diffusione in larga scala dei primi cellulari con display a colori, avvenuta durante il 2003, i giochi per device mobili iniziarono ad ottenere maggiore considerazione. Al punto che Nokia propose un telefono interamente dedicato al gaming, lo N-Gage, capace di supportare giochi stile console, ma evidentemente il mercato non era pronto per dei dispositivi ibridi, lo N-Gage venne subito stroncato dall'arrivo della Sony PSP[4] , che era anni luce più avanti da tutti i punti di vista.

Nel frattempo il mondo del mobile gaming catturò l'interesse di nomi importanti come Electronic Arts (EA), che iniziarono ad investire in quel mercato. Nel 2005 i giochi 3D cominciarono ad essere un fenomeno di massa, grazie all'aumento della potenza elaborativa dei telefoni.

Il problema della frammentazione delle piattaforme divenne sempre più rilevante man mano che la tecnologia progrediva, di conseguenza l'obiettivo delle compagnie iniziò a non essere più quello di realizzare il maggior numero possibile di giochi di qualità, ma raggiungere il maggior numero di utenti, focalizzandosi nel distribuire i videogiochi sulle tante piattaforme esistenti.

L'industria del gaming per dispositivi mobili stava maturando ma l'esperienza generale di gioco era ancora piuttosto limitata e primitiva rispetto a quella fornita dalle console portabili come PSP e Nintendo DS.

Nel 2007 ci fu una grande svolta, costituita dall'arrivo dell'iPhone, rilasciato da una compagnia fino ad allora estranea al mercato della telefonia.

[3]Per gameplay si intende tutto l'insieme delle meccaniche di gioco.
[4]Console portatile, acronimo di Playstation Portable.

Apple riuscì a rivoluzionare il mondo del mobile gaming non tanto grazie al dispositivo in sé, né alle sue avanzate caratteristiche hardware, ma con il lancio dell'App Store [7] nel 2008.

L'App Store ridefinisce completamente i sistemi di distribuzione del software, rendendo l'acquisto di un gioco una procedura semplice quanto scaricare un mp3 da iTunes. Gli sviluppatori in questo modo possono raggiungere milioni di utenti senza dover avere a che fare con compagnie di publishing.

Oggi i principali canali di distribuzione del software per dispositivi mobili seguono l'esempio di Apple: lo Android Market [2], L'OVI store di Nokia [35], il Windows Phone MarketPlace [58] di Microsoft e l'App World di RIM [39]. Questo dimostra quanto sia importante fornire all'utente uno strumento che renda immediato l'accesso al software.

L'evoluzione dei giochi per dispositivi mobili continua in maniera incessante, spinta contemporaneamente dall'avanzamento tecnologico dei dispositivi e dei sistemi operativi e dall'innovazione dei sistemi di gioco.

1.2 Confronto con altre piattaforme di gioco

L'esperienza di gioco fornita da un dispositivo mobile può essere sostanzialmente diversa da quella fornita da un PC o da una console. I principali fattori che determinano questa distinzione possono essere raggruppati sotto due categorie, la prima riguardante alcuni aspetti economico-sociali del mondo dei giochi, la seconda riguardante aspetti intrinseci dei dispositivi mobili.

1.2.1 Differenze derivanti dalla struttura del mercato

Il target[5] dei giochi per dispositivi mobili è molto più ampio di quello relativo ai giochi per console e per PC, i quali attraggono prevalentemente giocatori appassionati. I giochi destinati alle piattaforme più tradizionali sono arrivati ad un livello di evoluzione tale che i loro acquirenti, essendo abituati a continui e nutriti progressi tecnologici del settore, riescono ormai ad apprez-

[5]Per target si intende un gruppo di consumatori al quale un impresa si rivolge con una strategia di marketing. [54]

zare soltanto giochi molto avanzati dal punto di vista tecnico, mettendo in secondo piano aspetti come ad esempio lo spessore della trama o l'originalità del sistema di gioco. L'aspetto grafico quindi costituisce sempre più spesso il parametro di qualità predominante per i videogiochi e questo fa si che gli sforzi dei loro creatori siano continuamente concentrati sullo sviluppo di nuove tecnologie.

Affinché questo sia possibile, è necessario che le case produttrici di videogiochi dispongano di enormi capitali da investire nella realizzazione dei loro prodotti, questo determina la configurazione di un mercato con alte barriere all'ingresso in cui soltanto poche grandi società sono in grado di competere, si tratta di una forma di oligopolio. Inoltre questa dinamica tende a far convergere la varietà dei giochi in poche categorie ben consolidate e a generare titoli vicini ad essere cloni di se stessi, visto che nessuno si assumerebbe il rischio di spendere decine di milioni di dollari senza avere la minima sicurezza di ottenere un prodotto che venga apprezzato. Tuttavia ci sono alcune piattaforme come la Nintendo Wii che rappresentano delle eccezioni a questo approccio, i videogiochi vengono indirizzati ad un target totalmente diverso e di conseguenza i loro requisiti sono di diversa natura.

Quest'ultimo scenario presenta alcuni aspetti comuni al contesto dei device mobili. Queste piattaforme di gioco meno tradizionali sono caratterizzate da un vasto bacino di utenti, ovviamente molto segmentato. L'obiettivo degli sviluppatori di giochi per dispositivi mobili è di rendere questi ultimi appetibili a tutte le fasce di consumatori, focalizzandosi sui giocatori non hard-core [6] che hanno un livello di aspettativa generalmente più basso e prediligono i giochi non troppo seri e impegnativi.

Non essendoci particolari requisiti di complessità tecnologica, non si viene a creare la necessità di ingenti investimenti e di conseguenza i rischi legati allo sviluppo di giochi sono considerevolmente più bassi rispetto a quanto avviene per le piattaforme classiche. Questi bassi rischi incoraggiando la creatività degli sviluppatori, che sono incentivati a realizzare giochi innovativi, dato che sono liberi di sperimentare nuove forme di intrattenimento.

[6] Un giocatore molto appassionato che dedica buona parte del suo tempo libero ai giochi.

1.2.2 Differenze derivanti dall'hardware

L'altro insieme di fattori che differenziano profondamente i giochi per piattaforme mobili da quelli per PC e console derivano dall'hardware per cui tali giochi sono destinati. L'hardware è di fondamentale importanza per i giochi dato che generalmente essi mettono sotto sforzo tutti i componenti interni dei dispositivi più di quanto siano in grado di fare altre tipologie di applicazioni.

Limitazioni dei dispositivi mobili

Per essere definito "mobile", un dispositivo deve soddisfare dei requisiti di compattezza e portabilità, oltre ad avere una certa indipendenza da collegamenti via cavo, ed è proprio questa natura mobile ad imporre a tali dispositivi consistenti limitazioni in termini di dotazione hardware.

Attualmente i device mobili sono capaci di prestazioni di gran lunga inferiori rispetto a quelle fornite da un qualunque calcolatore moderno, visto che per garantire a tali dispositivi un adeguato livello di autonomia e scongiurare danni derivanti da elevate temperature, è necessario limitarne gli assorbimenti energetici da cui dipendono le emissioni di calore, due fattori strettamente legati alla potenza elaborativa e alle dimensioni dei chip.

Inoltre il loro minore ingombro forza i dispositivi mobili ad adottare sistemi di input/output alternativi.

- Input: tastiere fisiche di dimensioni ridotte oppure tastiere virtuali su dei touch-screen;
- Output: display di dimensioni ristrette e basse risoluzioni, altoparlanti se presenti sono di piccole dimensioni;

Infine ci sono altre limitazioni generali a cui sono soggetti i device mobili:

- Quantitativi di memoria RAM ridotti, con basse velocità di trasferimento;
- Modeste capacità di storage permanente, con basse velocità di trasferimento;

- Velocità di connessione relativamente basse;

Tutte queste limitazioni si ripercuotono in maniera più o meno diretta sull'esperienza di gioco.

Contenere le eventuali conseguenze negative derivanti da tali limitazioni è un'importante compito che spetta in parte agli sviluppatori del sistema operativo, in parte agli sviluppatori (si veda il paragrafo 4.3) dei giochi destinati alla piattaforma mobile.

Strumenti per compensare le limitazioni

Le numerose carenze dei device mobili possono essere in una qualche maniera colmate attraverso lo sfruttamento di alcune particolari funzionalità di cui dispongono.
Alcune di esse sono:

- Accelerometri;
- Sistemi di riconoscimento vocale;
- Magnetometri;
- Antenne GPS;
- Giroscopi;

Questi strumenti possono essere integrati all'interno di un gioco per ottenere un risultato più completo e coinvolgente.

1.3 Un esempio di gioco per device mobili

Angry Birds è uno dei videogiochi per piattaforma mobile più diffusi di sempre, costituisce una prova delle dimensioni raggiunte dal mercato dei giochi per device mobili. Il suo successo lo sta portando a diventare un fenomeno crossmediale. Sviluppato dalla software house Rovio [40], ha destato perfino l'interesse di EA, che nell'ottobre 2010 acquisì il publisher di Rovio, Chillingo [10], per 20 milioni di dollari.

Figura 1.1: Angry Birds [23]

Angry Birds appare sull'App Store nel 2009, dopo essere divenuto un best seller per iPhone, viene rilasciato anche per Android nell'ottobre 2010.

L'obiettivo del gioco è gestire delle squadre di uccellini contro dei maialini che hanno rubato le loro uova. Il gioco consiste nell'utilizzare una fionda per scagliare varie specie di volatili contro le strutture difensive dei maialini verdi, come è possibile vedere in figura 1.2. Il senso di sfida viene fornito dal sistema di gestione della fisica, che è allo stesso tempo anche la principale fonte di divertimento.

Con una trama lineare, un gameplay alquanto impegnativo ma molto originale, Angry Birds è un gioco al quale non si riesce a smettere di giocare facilmente. Grazie ad una grafica semplice ma accattivante, effetti sonori divertenti ed un sistema di premi appagante, ha tutte le carte in regola per essere definito un "casual game".[7]

[7] I casual game sono giochi appetibili a tutte le fasce di consumatori, tra cui anche adulti, che vengono giocati molto intensamente ma per brevi momenti. I cosiddetti "entertainment snacking" occupano soprattutto tempi vuoti tra un impegno ed un altro. Un classico esempio di casual game è tetris.

A questa categoria di giochi il Wall Street Journal [56] ha dedicato un articolo in cui viene spiegato come giocare ad essi possa ridurre lo stress degli utenti. Una comune

Angry Birds è destinato ad essere portato a breve anche sulle principali console, inoltre è probabile che presto diventi anche un cartone animato.

strategia su cui si basano i casual games è quella di inserire un meccanismo di premiazione che faccia sentire il giocatore appagato.

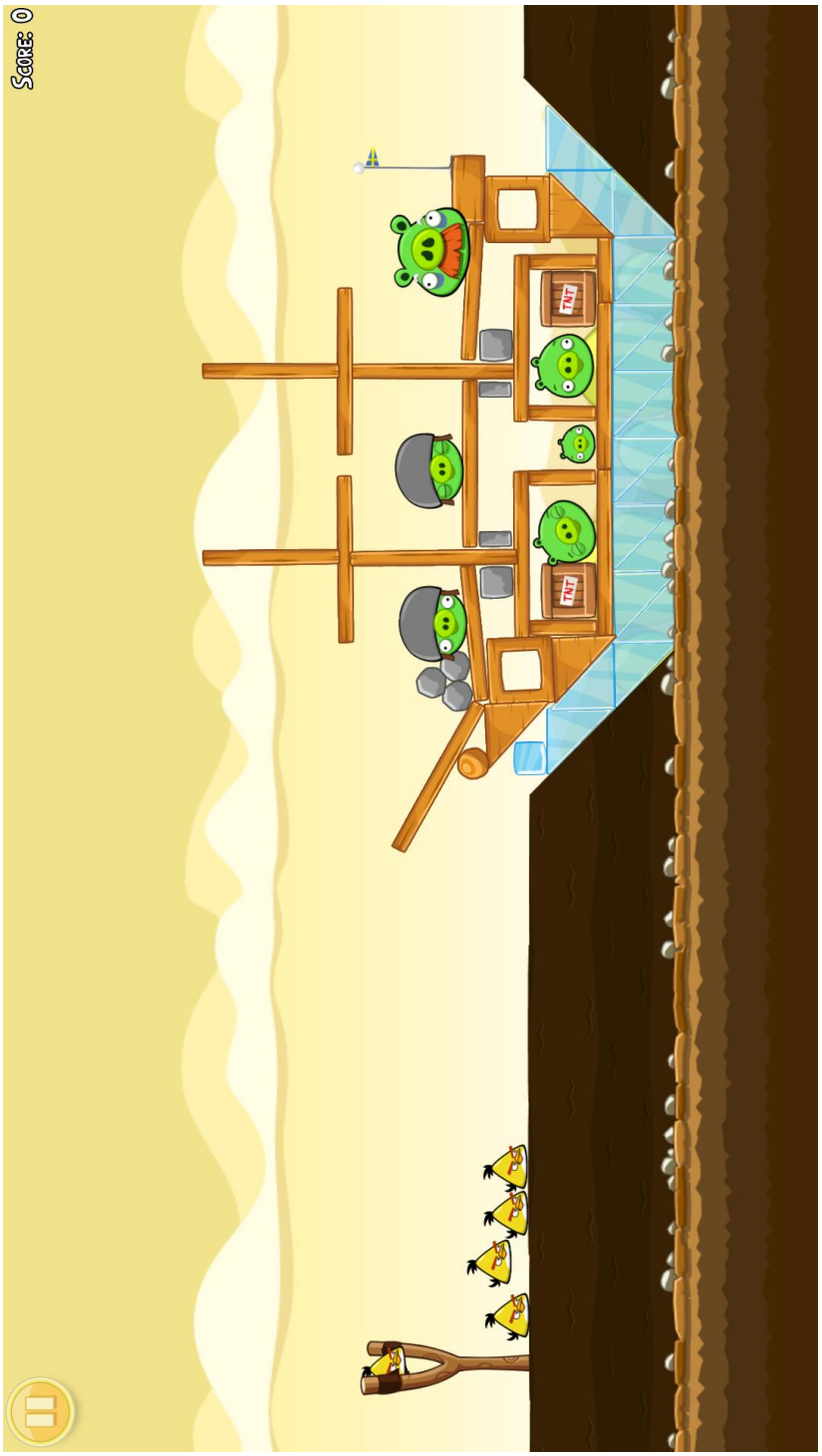


Figura 1.2: Schermata di gioco di Angry Birds [57]

Capitolo 2

Android

2.1 Cos'è Android?

Quando si parla di Android ci si riferisce ad un sistema operativo open source [6] per dispositivi mobili, ad una piattaforma di sviluppo open source [6] ed inoltre all'insieme di dispositivi e applicazioni basati su Android, concetto schematizzato in figura 2.1. Ecco come viene descritto da uno dei suoi creatori, Andy Rubin:[1]

> "The first truly open and comprehensive platform for mobile devices, all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation"[2]

Infatti tutti gli aspetti del sistema operativo Android, da quelli più superficiali a quelli più critici possono essere modificati liberamente. Questo significa che è consentito ai produttori di dispositivi mobili di adattare il sistema operativo ai propri smartphone, per esempio rimpiazzando l'interfaccia grafica standard con una fornita da essi oppure andando semplicemente a sostituire il sistema di gestione delle chiamate con un altro ritenuto migliore. Quanto detto vale sia per gli utenti finali che hanno la piena facoltà di

[1] Andy Rubin [42] è uno dei fondatori di Android Inc., la società che ha sviluppato Android, adesso è vicepresidente della sezione ingegneria a Google. Per la sua dichiarazione su Android vedere il blog ufficiale di Google [48]

[2] "La prima piattaforma per dispositivi mobili realmente aperta e completa, tutto il software che fa funzionare un telefono ma senza gli ostacoli proprietari che hanno frenato l'innovazione del mondo mobile."

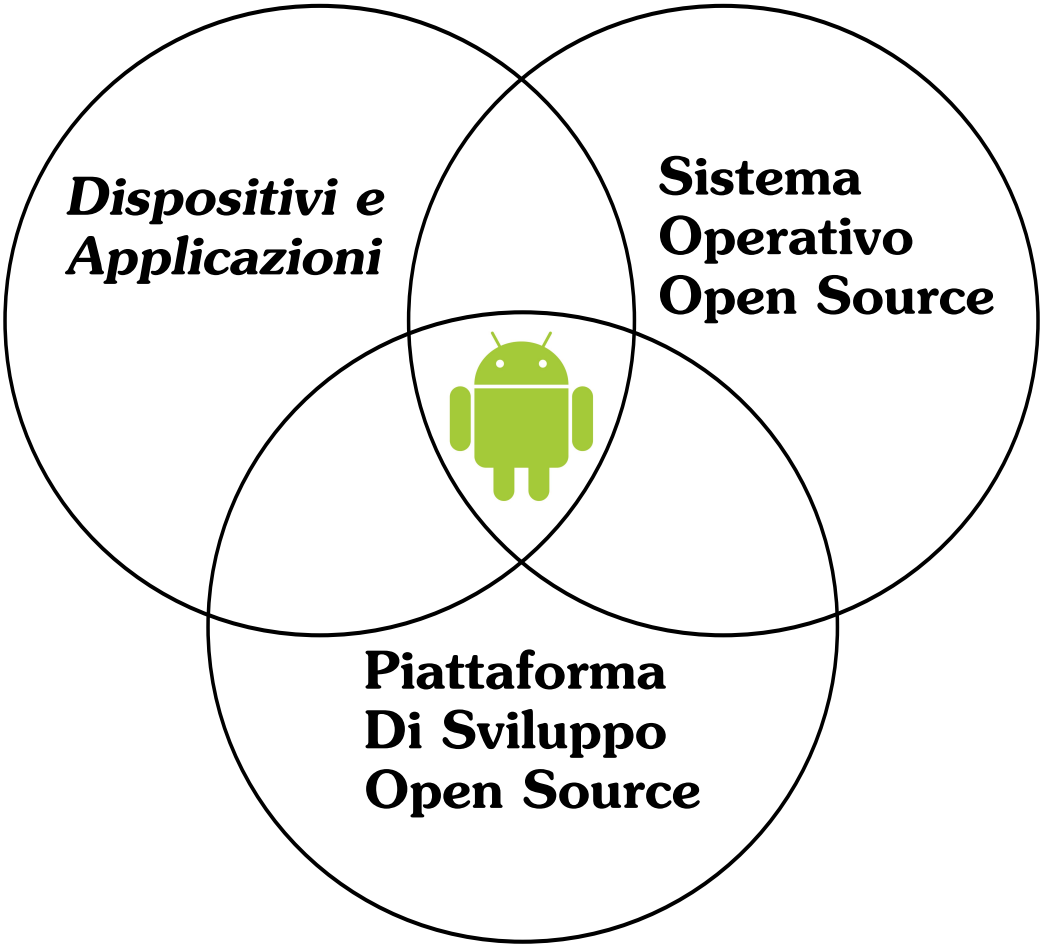


Figura 2.1: I tre domini di Android.

personalizzare in ogni aspetto il prodotto acquistato che per gli sviluppatori che hanno la possibilità di rilasciare applicazioni in grado di modificare qualunque caratteristica del sistema.

> "What really makes Android compelling is its open philosophy, which ensures that you can fix any deficiencies in user interface or native application design by writing an extension or replacement. Android provides you, as a developer, with the opportunity to create mobile phone interfaces and applications designed to look, feel, and function exactly as you imagine them."[3]

[3]"Ciò che rende Android davvero irresistibile è la sua filosofia aperta, che permet-

Reto Meier[4]

Un'alleanza per definire uno standard. Nei mesi precedenti alla presentazione ufficiale di Android, voci di corridoio avevano creato la convinzione che Google stesse lavorando su un cosiddetto GPhone [43], o "Googlefonino", queste voci furono smentite il giorno dell'annuncio [48] di Android:

> "Despite all of the very interesting speculation over the last few months, we're not announcing a Gphone. However, we think what we are announcing – the Open Handset Alliance and Android – is more significant and ambitious than a single phone"[5]

Andy Rubin.

> "Our vision is that the powerful platform we're unveiling will power thousands of different phone models."[6]

Eric Schmidt.[7]

La Open Handset Alliance (OHA) [49] è un insieme di società operanti nell'ecosistema della telefonia mobile costituito da operatori di telefonia mobile, produttori di dispositivi, produttori di software, produttori di semiconduttori e compagnie per la commercializzazione. Dei più di 70 membri se ne elencano alcuni tra i più rilevanti: Google, Motorola, HTC, T-Mobile, Samsung, Nvidia, Qualcomm.

Direttamente dal sito della OHA, dove è possibile trovare la lista completa dei membri:

---

te di compensare ogni carenza dell'interfaccia grafica o nel sistema di progettazione di applicazioni native attraverso l'attuazione di estensioni o sostituzioni. Android da agli sviluppatori l'opportunità di creare interfacce e applicazioni per cellulare progettate per essere proprio come essi le immaginano"

[4]Reto Meier [31] è uno sviluppatore Android, Advocate di Google ed autore del libro Professional Android 2 Development (vedi [32])

[5]"Nonostante tutte le speculazioni molto interessanti che ci sono state nel corso degli ultimi mesi, non stiamo annunciando alcun Gphone. Tuttavia riteniamo che ciò che stiamo annunciando – la Open Handset Alliance e Android – sia più significativo e ambizioso di un singolo telefono."

[6]"La nostra visione è che la potente piattaforma che stiamo svelando sarà la base di migliaia di diversi modelli di telefono."

[7]Eric Schmidt è presidente e Chief Executive Officer (CEO) di Google [19]

> "A commitment to openness, a shared vision for the future, and concrete plans to make the vision a reality"..."To accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience."[8]

l'OHA spera di riuscire a offrire ai consumatori di dispositivi mobili una migliore esperienza, fornendo una piattaforma che è necessaria per garantire una veloce innovazione dell' intero mondo mobile, senza il peso delle licenze software.

2.2 Breve storia

Il sistema operativo fu inizialmente sviluppato da Android Inc. che venne acquisita nel 2005 da Google. I fondatori di Android Inc., Andy Rubin, Rich Miner, Nick Sears e Chris White passarono alle dipendenze di Google e svilupparono una piattaforma basata sul Kernel Linux.

Il 5 novembre 2007 venne annunciata la costituzione dell'Open Handset Alliance, la quale lo stesso giorno presentò Android, basato sulla versione 2.6 del Kernel Linux. Pochi giorni dopo, l'OHA rilasciò una versione di anteprima del Software Development kit (SDK) di Android, molto prima dell'arrivo di una qualsiasi forma di hardware in grado di supportare il nuovo sistema operativo di Google.

Il 23 settembre 2008 contemporaneamente all'aggiornamento della SDK alla versione 1.0, viene annunciato il primo dispositivo Android: il T-Mobile G1.

Da quel momento la piattaforma Android ebbe una crescita incredibile, subendo numerosi aggiornamenti in poco tempo:

Versione 1.1, Febbraio 2009

- Risolve numerosi problemi e aggiunge nuove API.

[8] "Una visione condivisa per il futuro e piani concreti per rendere la visione una realtà"..."Accelerare l'innovazione nel mondo mobile e offrire ai consumatori un'esperienza nel mondo mobile più ricca e meno costosa."

Versione 1.5 (Cupcake), Aprile 2009

- Introduzione della tastiera virtuale.
- Possibilità di aggiungere Widget e cartelle alla schermata Home.

Versione 1.6 (Donut), Settembre 2009

- Aggiornamento della funzione di ricerca vocale.
- Miglioramenti all'Android Market.
- Aggiunta di un sintetizzatore vocale.
- Possibilità di effettuare ricerche direttamente dalla schermata Home.

Versione 2.0 e 2.1 (Eclair), Ottobre 2009 e Gennaio 2010

- Supporto di risoluzioni multiple.
- Nuova interfaccia grafica.
- Supporto al Multi-Touch.
- Supporto ad HTML 5.

Versione 2.2 (Froyo), Maggio 2010

- Ottimizzazione delle prestazioni.
- Implementazione del JIT.
- Integrazione del motore JavaScript V8 di Chrome.
- Supporto ad Adobe Flash 10.1.
- Possibilità di installare applicazioni sulla memoria SD.
- Tethering USB e WI-FI hotspot

Versione 2.3 (Gingerbread), Dicembre 2010

- Nuova interfaccia grafica.
- Introduzione Garbage Collector concorrente.
- Miglioramenti relativi allo sviluppo di giochi.
- Supporto a più tipi di sensori.
- Migliore gestione energetica.

Nel mese di gennaio 2011 è stata annunciata la versione 3.0 denominata Honeycomb (vedi figura 2.2), questa nuova generazione di Android è destinata ai Tablet PC, il cui mercato al momento è totalmente dominato da Apple. Nei pochi anni che sono passati dalla sua nascita, Android ha raggiunto dimensioni considerevoli in termini di quantità di applicazioni disponibili (oltre duecentomila), numero di dispositivi basati su di esso (decine di modelli diversi) e presenza totale nel mercato mobile, nel 2010 ha superato [37] perfino Symbian[9].

2.3 Il Sistema Operativo

Il sistema operativo Android è basato su Kernel Linux e consiste in una struttura formata da vari livelli o layer, ognuno di essi fornisce al livello superiore un'astrazione del sistema sottostante. La figura 2.3 raffigura il cosiddetto Software Stack di Android.

2.3.1 Strato Applications

Il livello più alto dello Stack è costituito dalle applicazioni: non soltanto quelle native come per esempio il sistema di gestione dei contatti, l'applicazione per l'invio di SMS, il calendario, ecc... ma anche quelle provenienti da altre fonti. Android non differenzia le applicazioni di terze parti da quelle

[9]Symbian è il sistema operativo presente nei dispositivi mobili Nokia, che di recente ha stretto un'alleanza con Microsoft per contrastare Android e iPhone.

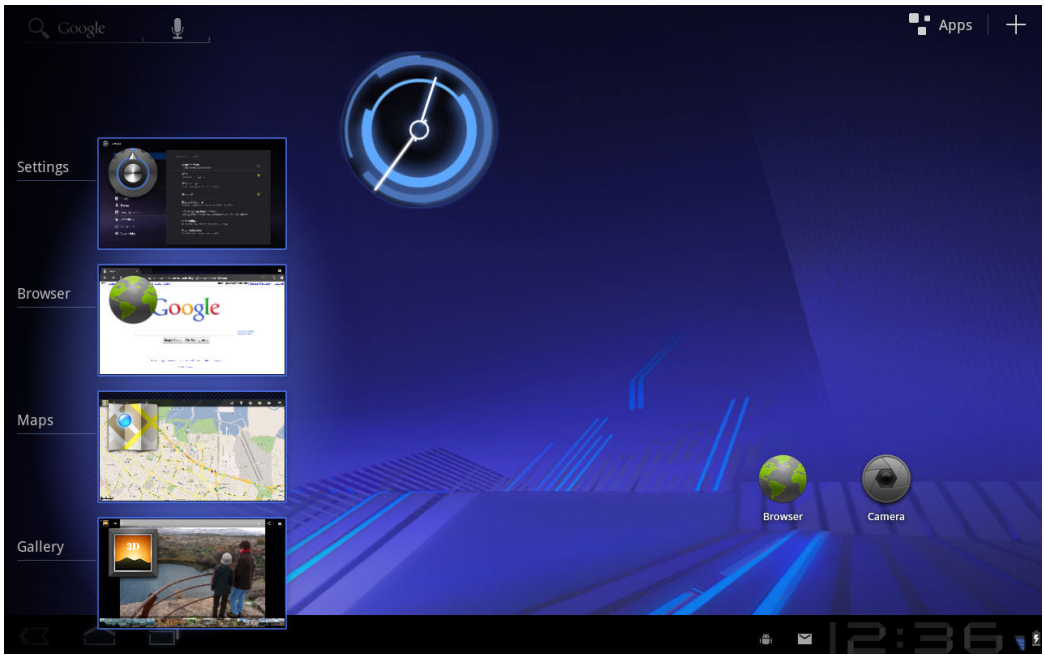


Figura 2.2: La schermata principale di Android 3.0 Honeycomb. [24]

già incluse nel telefono, infatti garantisce gli stessi privilegi a entrambe le categorie.

2.3.2 Strato Application Framework

L'architettura di Android incoraggia il riutilizzo di componenti rendendo possibile la condivisione di servizi tra più applicazioni. In questo modo l'Application Framework permette agli sviluppatori di concentrarsi nella risoluzione di problemi non ancora affrontati avendo sempre a propria disposizione il lavoro già svolto da altri.

Questo framework è basato su classi Java che però non vengono eseguite in un classico ambiente Java. Infatti è stato possibile evitare gli oneri derivanti dall'adozione della Java Virtual Machine (JVM) realizzando una macchina virtuale ad hoc denominata Dalvik Virtual Machine (DVM), essa verrà analizzata nel paragrafo 2.3.4.

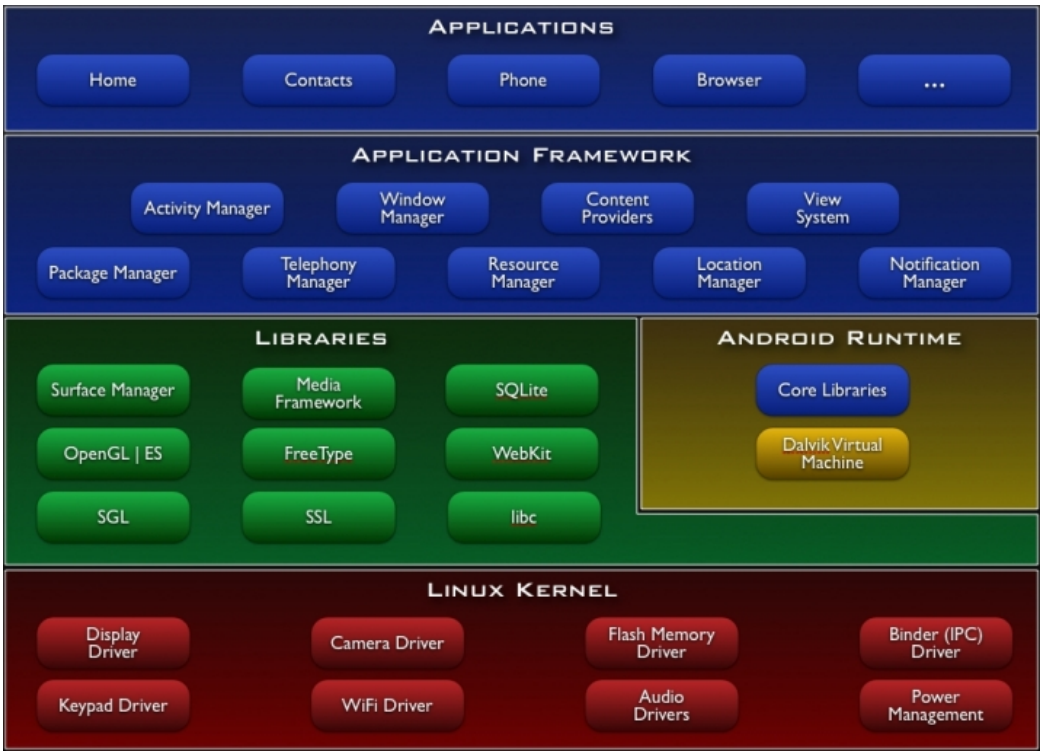


Figura 2.3: Lo stack di Android [6].

Activity Manager

La Activity è uno degli elementi che compongono una applicazione, la quale ne avrà una per ogni schermata della sua interfaccia grafica. Lo Activity Manager si occupa di controllare il ciclo di vita delle Activity, tiene traccia quindi di tutte le schermate associate ad una determinata applicazione.

Window Manager

Il Window Manager gestisce le finestre relative a differenti applicazioni, astrae alcuni servizi del Surface Manager dello strato Libraries.

Content Providers

I Content Provider permettono alle applicazioni di avere accesso a dati condivisi.

View System

Il View System è il componente che fornisce gli strumenti per gestire elementi grafici di diverso tipo. Ad esempio un'area di testo può essere facilmente implementata attraverso la classe TextView che astrae la gestione di elementi grafici di più basso livello.

Package Manager

Il Package Manager (PM) si occupa della gestione delle applicazioni dal punto di vista della loro installazione/disinstallazione nei dispositivi. Ogni applicazione Android è un pacchetto APK che racchiude tutti i dati e le informazioni relative ad essa. Il PM è in grado di riconoscere le applicazioni prima di tutto dalla loro estensione .apk.

Telephony Manager

Il Telephony Manager fornisce vari servizi per un'interazione più diretta con le funzionalità telefoniche del dispositivo.

Resource Manager

Il Resource Manager gestisce tutte le risorse appartenenti ad un'applicazione. Rende quindi molto semplice per lo sviluppatore effettuare varie operazioni su dati come immagini, traccie audio, ecc...

Location Manager

Rende immediato l'accesso a tutti gli hardware in grado di individuare la posizione del dispositivo. Permette quindi di realizzare con pochi passaggi servizi location-based come mappe, giochi che sfruttano la posizione degli utenti, navigatori satellitari ed altri.

Notification Manager

Il Notification Manager da la possibilità allo sviluppatore di interagire con vari sistemi utilizzati per notificare determinati eventi. Al momento le notifiche all'utente possono avvenire nei seguenti modi:

- attraverso un'icona nella barra di notifica;
- attraverso il LED del dispositivo;
- attraverso l'attivazione della retroilluminazione del display;
- attraverso la riproduzione di suoni;
- attraverso la generazione di vibrazioni.

2.3.3 Strato Libraries

Android include una serie di librerie C/C++ che vengono usate da vari componenti del sistema. Attraverso l'Application Framework gli sviluppatori hanno accesso ai servizi forniti da queste librerie, di seguito viene riportata una lista con alcune di esse:

Libc

Un'implementazione di derivazione Berkeley Software Distribution (BSD) della libreria di sistema standard C, ottimizzata per dispositivi basati su versioni embedded di Linux.

Media Framework

Permette la riproduzione e registrazione di media, dando supporto ai più diffusi formati audio, video e immagini attraverso l'utilizzo delle librerie OpenCORE messe a disposizione da PacketVideo.

Surface Manager

È il componente che si occupa dell'interfaccia grafica del dispositivo, da accesso al sottosistema display gestendo vari livelli grafici 2D e 3D provenienti da diverse applicazioni.

WebKit

È un moderno web browser engine che sta alla base del browser di Android, il quale può essere utilizzato da qualunque applicazione sotto forma di finestra-browser.

SGL

Acronimo di Scalable Graphics Library, è il motore grafico 2D di Android, permette al Window Manager e al Surface Manager di realizzare le proprie funzioni.

OpenGL | ES

Si tratta dell'implementazione della versione 1.0 delle API OpenGL ES. Queste librerie grafiche usano l'efficiente rasterizer software a tre dimensioni presente di default, ma sono anche in grado di sfruttare l'accelerazione hardware 3D qualora essa sia disponibile.

FreeType

Il sistema di gestione dei font è affidato al motore FreeType che è stato scelto perché particolarmente leggero, efficiente, personalizzabile e portabile ma allo stesso tempo capace di produrre immagini di alta qualità. Tutto questo mantenendo un approccio indipendente dal tipo di file da visualizzare.

SQLite

Il Database Management System (DBMS) utilizzato da Android. SQLite è un DBMS relazionale molto potente e affidabile che si distingue per la sua leggerezza, i motivi della sua sua scelta risultano chiari in quanto i DBMS

sono solitamente prodotti molto esigenti in termini di risorse vista la loro complessità a livello software.

SSL

Sono le librerie che implementano il protocollo di sicurezza crittografico Secure Sockets Layer (SSL), un servizio fondamentale che garantisce un certo livello di sicurezza per tutte le comunicazioni effettuate tramite il dispositivo.

2.3.4 Strato Android Runtime

Ciò che distingue il sistema operativo Android da una implementazione mobile di Linux è il runtime, che è formato dalle cosiddette core libraries e dalla Dalvik Virtual Machine. Le core libraries includono buona parte delle funzionalità fornite dalle librerie standard di Java a cui sono state aggiunte librerie specifiche di Android.

Dalvik Virtual Machine

Come accennato nel paragrafo 2.3.2 le applicazioni Android non vengono eseguite all'interno di una Java Virtual Machine, esse girano sulla Dalvik Virtual Machine che è una particolare macchina virtuale progettata appositamente per dispositivi mobili, essa garantisce una certa indipendenza del software dalle varie architetture hardware possibili.

Un'essenziale caratteristica della Dalvik Virtual Machine è quella di riuscire a gestire in maniera molto parsimoniosa ed efficiente le poche risorse messe a disposizione dai dispositivi mobili.

Questo risultato è stato raggiunto grazie a numerosi sforzi e accorgimenti, come per esempio la rimozione di numerose librerie Java non necessarie ai dispositivi mobili o l'utilizzo di un particolare tipo di Bytecode, diverso dal Bytecode Java.

L'adozione di un'architettura Register-Based[10] per la DVM, in contrapposizione alla natura delle JVM che sono stack machines, costituisce un'altro esempio di ottimizzazione.

La DVM è stata concepita per poter essere eseguita in più istanze contemporaneamente sullo stesso dispositivo, ogni applicazione viene associata ad un processo che gira all'interno di una DVM ad esso dedicata. Dato che le varie DVM sono isolate, un eventuale malfunzionamento di un'applicazione non influenza le altre né mette in pericolo la stabilità del sistema operativo stesso.

Un'altra delle molte differenze della Dalvik Virtual Machine rispetto alla classica realtà Java sta nei file eseguibili: la DVM non esegue file .class ma file .dex (dalvik executable). I file .class vengono convertiti in file.dex che sono ottimizzati per avere dimensioni sensibilmente minori.

Tra gli ultimi aggiornamenti che sono stati fatti alla DVM troviamo l'introduzione della compilazione Just In Time (JIT)[11] che offre consistenti incrementi prestazionali.

La DVM si appoggia al Kernel Linux per funzionalità quali gestione di thread e gestione della memoria a basso livello.

2.3.5 Strato Linux Kernel

Android si affida alla versione 2.6 del Kernel Linux che risiede in fondo allo stack del sistema operativo, di conseguenza fornisce al resto del software un'astrazione dell'hardware sottostante. Il Kernel costituisce il nucleo di Android in quanto eroga i servizi di sistema di livello più basso come:

- Gestione della memoria.
- Gestione dei processi
- Sicurezza.

[10] Per una descrizione dettagliata vedi Wikipedia [54]

[11] La compilazione Just In Time permette un tipo di compilazione, conosciuta anche come traduzione dinamica, con la quale è possibile aumentare le prestazioni dei sistemi di programmazione che utilizzano il bytecode, traducendo il bytecode nel codice macchina nativo in fase di run-time. (per approfondimenti vedi Wikipedia [54])

- Stack di rete.
- I driver per tutto l'hardware del dispositivo.

Il fatto che Android usa un Kernel di derivazione Linux non si traduce nel fatto che Android sia a tutti gli effetti una distribuzione GNU/Linux, il sistema operativo di Google infatti non possiede un X Window System[12] e non include tutto l'insieme delle librerie standard GNU.

2.4 I dispositivi e le applicazioni

2.4.1 I dispositivi

Anche se Android è stato inizialmente concepito per smartphone (un esempio di smartphone è in figura 2.4), le sue caratteristiche gli permettono di essere usato su svariati tipi di dispositivi come per esempio i Netbook ed i Tablet PC.

"With no licensing fees or proprietary software, the cost to handset manufacturers for providing Android handsets, and potentially other Android-powered devices, is comparatively low. Many people now expect that the advantages of Android as a platform for creating powerful applications will encourage device manufacturers to produce increasingly tailored hardware."[13] Reto Meier. [32]

Il primo dispositivo mobile dotato di Android è stato il T-Mobile G1 noto anche come HTC Dream, rilasciato il 22 ottobre 2008 .

Le sue caratteristiche principali sono:

- processore: Qualcomm MSM7201A ARM11 528 MHz;
- tastiera QWERTY;

[12] X Window System è il gestore grafico standard per tutti i sistemi Unix.

[13] "In assenza di tariffe legate a licenze software ed in assenza di software proprietario, i costruttori di hardware compatibile con Android hanno costi di produzione relativamente bassi. Molte persone si aspettano che ora i vantaggi di Android come piattaforma per lo sviluppo di potenti applicazioni incoraggeranno i produttori a creare dispositivi con hardware sempre più a misura di Android"

- schermo touch-screen 3.2 pollici, risoluzione: 320x480 pixel;
- connettività: 3G UMTS/HSDPA;
- 192 MB di RAM;
- 256 MB di memoria flash interna;

Alla fine del 2009 esistevano circa venti [60] dispositivi basati su Android, numero che durante il 2010 è salito fino a superare i novanta [52] dispositivi, se si considerano le varianti di ogni modello.

Le principali caratteristiche tecnologiche e funzionalità messe attualmente a disposizione da Android sono le seguenti:

Connettività

Supporto a sistemi di connessione come: GSM/EDGE, UMTS, Bluetooth, Wi-Fi, WiMAX ed altri.

Messaggistica

Invio di SMS e MMS.

Browsing

Il browser standard di Android è basato sul motore Web-Kit affiancato dal motore javaScript V8, lo stesso del browser Google Chrome.

Supporto Media

Alcuni dei formati audio/video/immagini supportati sono: mp3, MIDI, Ogg Vorbis, WAV, H.263, H.264 (attraverso container mp4 o 3gp), MPEG-4 SP, AMR, AMR-WB (attraverso container 3gp), AAC, HE-AAC (attraverso container mp4 o 3gp), JPEG, PGN, GIF, BMP.

Supporto Media in Streaming

Al momento i protocolli supportati sono: RTP/RTSP streaming (3GPP PSS, ISMA), download progressivo HTML (per mezzo del tag HTML5 <video>). Grazie al plugin Flash 10.1 sono disponibili anche: Adobe Flash Streaming (RTMP) e HTTP Dynamic Streaming.

Android Market

Il market è un catalogo di applicazioni che possono essere scaricate e installate direttamente sui dispositivi.

Funzionalità che sfruttano il riconoscimento vocale

Dalla ricerca vocale di Google al comando vocale di svariate applicazioni.

Altri Hardware supportati

I dispositivi Android possono essere equipaggiati di fotocamere e videocamere, touch-screen, antenne GPS, accelerometri, giroscopi, magnetometri.

I dispositivi mobili Android possono essere dotati di una certa quantità delle funzionalità tra quelle elencate, oltre ad eventuali capacità aggiuntive inserite dai produttori di hardware.

Limitazioni dei device mobili

Nonostante dal puro piano prestazionale gli smartphone non siano paragonabili a dei veri e propri PC non si può negare che essi stiano diventando dei dispositivi sempre più potenti e versatili, accompagnati da software dall'enorme complessità, basti pensare che le loro capacità prestazionali sono paragonabili a quelle possedute da calcolatori risalenti a circa una decina di anni fa che avevano un ingombro di decine di volte superiore.

Un grave punto debole di tutte le apparecchiature mobili è la batteria, che in un certo senso ne limita le potenzialità, se queste apparecchiature poi sono dispositivi da cui dipendiamo per esempio per effettuare o ricevere chiamate di emergenza, il problema acquista una certa rilevanza.

Figura 2.4: Il Nexus One, prodotto da Google in collaborazione con HTC, è uno dei dispositivi di nuova generazione [9].

Questa limitazione intrinseca dipende dal fatto che l'avanzamento tecnologico delle batterie non riesce a tenere il passo dell'incremento della potenza elaborativa e dell'inarrestabile aumento delle tecnologie inserite all'interno di uno smartphone, quasi sempre molto assetate di risorse.

"Mobile phones today are very powerful handheld computers, but they are still limited. The fundamental limitation of a mobile device is battery capacity. Every clock tick of the processor, every refresh of memory, every backlit pixel on the user's screen takes energy from the battery. Battery size is limited, and users don't like frequent battery charging. As a result, the

computing resources are limited"[14] Rick Rogers, John Lombardo [38]

2.4.2 Le Applicazioni

Uno strumento che permette l'incontro tra la domanda e l'offerta di programmi per Android è il Market, il servizio che da la possibilità agli utenti di reperire applicazioni e agli sviluppatori di renderle disponibili.

Bisogna sottolineare che agli sviluppatori non viene imposto di avvalersi di questa facoltà, essi possono scegliere liberamente il canale attraverso il quale distribuire i propri programmi. Gli utenti di conseguenza hanno a disposizione diverse fonti da cui attingere.

Lo Android Market nasce per limitare i problemi derivanti dalla notevole quantità di dispositivi e versioni del sistema operativo. Infatti la natura aperta del sistema comporta una frammentazione dovuta alle svariate personalizzazioni del prodotto possibili.

Gli sviluppatori sono tenuti a indicare il range di dispositivi con cui le loro applicazioni sono compatibili, specificando per esempio la versione minima del firmware richiesta dall'applicazione piuttosto che le risoluzioni per le quali essa è stata ottimizzata. In questo modo il Market rende accessibili agli utenti soltanto le applicazioni adatte ai loro smartphone, rendendo la ricerca di applicazioni una procedura più semplice e sicura, allo stesso tempo, fa sí che gli sviluppatori non vengano investiti da lamentele causate da incompatibilità.

Il Market opera quindi da punto di riferimento senza però esercitare alcun controllo sulle applicazioni né sulle loro possibili caratteristiche, esso si limita alla gestione delle transazioni economiche tra acquirenti e venditori. Nel momento dell'installazione viene chiesto all'utente di garantire all'applicazione eventuali permessi, come per esempio l'accesso ad internet, allo storage del dispositivo oppure il permesso di effettuare chiamate.

[14] "I cellulari di oggi sono dei palmari molto potenti, ma sono comunque limitati. Il limite fondamentale di un dispositivo mobile e la capacità della batteria. Ogni ciclo di clock del processore, ogni refresh della memoria, ogni pixel retroilluminato traggono energia dalla batteria. La dimensione delle batterie è limitata e gli utenti non amano ricaricarle frequentemente. Di conseguenza la potenza di calcolo è limitata."

L'introduzione di nuove applicazioni nel Market ed il loro aggiornamento non devono passare attraverso nessun processo di approvazione, queste operazioni possono avvenire quindi in maniera immediata e lo sviluppatore ha il pieno controllo su tutte le operazioni che riguardano il software di cui lui è autore.

Dato che non c'è nessuna entità che misura la qualità dei prodotti presenti nel market, è stato introdotto un sistema di rating che permette agli utenti di valutare le applicazioni fornendo informazioni preziose per la comunità. Le ricerche effettuate nel Market vengono ordinate in base alle valutazioni e al numero di download e di conseguenza agli sviluppatori non è lasciata altra scelta che rilasciare software di buona qualità, dato che le risultanti valutazioni negative corredate dagli analoghi commenti possono facilmente compromettere il futuro di un'applicazione.

Il Market inoltre offre un buon sistema di tutela del consumatore, quest'ultimo infatti può chiedere il rimborso della somma pagata entro un certo limite di tempo dal momento dell'acquisto dell'applicazione.

Un recente aggiornamento dell'Android Market consiste nella possibilità di esplorare il catalogo delle applicazioni attraverso una pagina web raggiungibile anche dal proprio PC. Questo facilita notevolmente il processo decisionale dell'utente dato che la pagina web in questione offre una notevole quantità di informazioni utili alla scelta delle applicazioni, tra cui le valutazioni ed i commenti, link esterni, video, immagini. Inoltre è possibile effettuare l'eventuale pagamento e l'installazione delle applicazioni direttamente dal proprio PC, attraverso un sistema che collega il proprio smartphone all'account Google da cui si sta accedendo all'Android Market.

2.5 La piattaforma di sviluppo di Android

Uno dei grandi punti di forza di Android è la sua piattaforma di sviluppo open source a cui vanno attribuiti buona parte dei meriti del successo raggiunto fin'ora.

I membri della Open Handset Alliance sono convinti che il miglior modo per offrire software di qualità ai consumatori è rendere semplice per gli

sviluppatori scriverlo. Come sostiene Nicolas Gramlich: [1]

> "Android-Imagination is the limit"[15]

L'idea su cui è basata la piattaforma di sviluppo di Android è quella di non limitare in nessun modo le potenzialità degli sviluppatori offrendo loro gli stessi mezzi usati dai creatori di Android stesso. Infatti le applicazioni di terze parti vengono considerate allo stesso livello di quelle native:

> "your apps are not second class citizens, they are at the same level as any other app that ships with the phone"..."you got the ability to leverage other people's work to enrich your own app, or to become the source for other people to use so your app can be part of someone else's"[16]
>
> Reto Meier [31]

Inoltre Android permette di utilizzare parti di applicazioni all'interno di altre non solo attraverso il classico riutilizzo di codice ma soprattutto grazie ad un efficace meccanismo basato sugli Intent Filters. Gli Intent Filters sono un metodo per esporre al resto del sistema le azioni che un'applicazione può compiere in modo da poter essere sfruttate da qualunque applicazione, essi offrono numerosi benefici tra cui:

- Massimizzare il riutilizzo e la modularità dei componenti. Le applicazioni possono specializzarsi su servizi singoli e in caso di necessità interagire tra loro per fornire un servizio migliore.
- Adattare al meglio le applicazioni alle esigenze dell'utente. Ad esempio l'applicazione relativa alla fotocamera, che ha una funzione per permettere la condivisione delle immagini, può chiedere all'utente di selezionare l'applicazione da usare per portare a termine tale operazione, il software che richiede il servizio può tranquillamente non essere a

[15] anddev.org Site-Admin. [1]

[16] "Le vostre applicazioni non sono cittadini di seconda classe, esse sono allo stesso livello di qualsiasi altra applicazione che viene fornita con il telefono"..."Avete la possibilità di sfruttare il lavoro di altri per arricchire la vostra applicazione, o di essere una vera e propria fonte, in modo che la vostra applicazione può essere parte di quella di qualcun altro."

conoscenza delle applicazioni correntemente installate sul dispositivo, sarà il meccanismo degli Intent Filters a individuare i programmi che potranno accogliere la richiesta.

2.5.1 Android Software Development Kit (SDK)

Il Software Development Kit[17] di Android mette a disposizione numerosi strumenti considerati nel seguito.

Le Application Programming Interface (API) di Android

Il nucleo della SDK è costituito dalle API[18], che sono librerie messe a servizio degli sviluppatori, sono esattamente le stesse usate da Google per creare le applicazioni native.

Android Virtual Device Manager

Un Android Virtual Device (AVD) consiste in un vero e proprio emulatore di dispositivi Android, utile per poter sviluppare applicazioni anche senza essere in possesso di uno smartphone oppure per simulare una configurazione hardware neutrale. È possibile creare dispositivi virtuali scegliendo la grandezza e densità del display, la versione del sistema operativo installata e altri parametri.

Dalvik Debug Monitor Server (DDMS)

È uno strumento essenziale per il debugging che include una serie di funzionalità per monitorare l'esecuzione delle applicazioni su dispositivi reali o emulatori.

[17]Software Development Kit è un termine che in italiano si può tradurre come "pacchetto di sviluppo per applicazioni" e sta a indicare un insieme di strumenti per lo sviluppo e la documentazione di software. [54]

[18]Le Application Programming Interface, sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito. È un metodo per ottenere un'astrazione, di solito tra l'hardware e il programmatore, o tra software a basso ed alto livello. Le API permettono di evitare ai programmatori di scrivere tutte le funzioni dal nulla. [54]

Documentazione

La documentazione inclusa nella SDK è molto dettagliata, offre informazioni precise su ogni singolo package e classe.

Tutorial

Sono guide per la realizzazione di applicazioni basilari, utili per gli sviluppatori alle prime armi.

Codice di esempio/ dimostrativo

I tutorial sono accompagnati da applicazioni già pronte che possono venire usate come modello per costruirne altre.

2.5.2 Altri strumenti

Oltre alla SDK esistono anche altre risorse preziose per gli sviluppatori:

Ambienti di sviluppo Integrato

Per il momento l'unico Integrated Development Environment (IDE)[19] ufficialmente supportato da Android è Eclipse [27] se equipaggiato con il plugin Android Development Tools (ADT) che permette di rendere molto più semplice e rapido il processo di sviluppo software visto che nativamente tutti i tools della SDK non sono accessibili tramite interfacce grafiche ma soltanto attraverso linea di comando.

[19] Ambiente di sviluppo integrato, software di ausilio alla programmazione.

Parte II

Realizzazione del progetto

Questa seconda parte riguarda la realizzazione del gioco “Catching Cherries”.

Nel capitolo 3 verrà presentata la tecnica di sviluppo [54] adottata (paragrafi 3.1, 3.2), inoltre verrà illustrato il linguaggio di modellazione scelto (paragrafo 3.3) ed infine verrano introdotti gli strumenti usati per la gestione del progetto (paragrafi 3.4, 3.5).

Nei capitoli 4, 5, 6, 7 verranno affrontati in dettaglio tutti i passi del processo di sviluppo del gioco, dal suo concepimento al suo rilascio.

Capitolo 3

Le tecniche di sviluppo adottate

Quando si realizza un sistema software è importante seguire un percorso che aiuti a raggiungere risultati di alta qualità in un tempo prefissato. Questo significa stabilire una serie di passi da effettuare in determinati momenti in modo da ottenere un piano di lavoro.

Ma come disse il Generale Eisenhower, il vero valore dei piani non risiede nella documentazione che li testimonia, ma nell'insieme di ragionamenti che costituiscono il processo stesso di pianificazione:

"'In preparing for battle I have always found that plans are useless, but planning indispensable"[1]

3.1 Tecnica di sviluppo adottata

La tecnica di sviluppo su cui è ricaduta la scelta fa parte della famiglia dei modelli iterativi[2] ed è fortemente ispirata al Rational Unified Process (RUP) [29, 54].

[1] "Nel preparare una battaglia ho sempre trovato che i piani sono inutili, ma la pianificazione è indispensabile". [55]

[2] Nella categoria dei modelli iterativi rientrano tutti i processi di sviluppo che permettono di ritornare ad una fase del procedimento già affrontata in precedenza. Per ogni fase "iterabile" si tende a partire con un abbozzo della soluzione che verrà successivamente raffinata al prossimo passaggio. [54]

Alcuni vantaggi offerti dai modelli iterativi rispetto a quelli a cascata[3] sono:

- Riduzione dei rischi anticipata.

  Nei modelli a cascata i rischi vengono ridotti soltanto quando il sistema è in uno stadio molto avanzato, questo significa che la probabilità di fallimento del progetto è più alta se si adottano quei modelli. I modelli iterativi permettono di eliminare i rischi più critici prima di fare grossi investimenti nel progetto.

- Feedback precoce.

  È possibile ottenere feedback dagli utenti anche prima della realizzazione complessiva del prodotto, in maniera tale da poterlo modificare prima che esso raggiunga stadi troppo avanzati, in cui le modifiche diventano estremamente costose.

- Testing e integrazione vengono effettuati costantemente.

Le figure 3.1, 3.2 evidenziano le differenze sostanziali dei due modelli citati, mentre la figura 3.3 mostra l'entità del rischio in funzione dello stato di avanzamento di un progetto a seconda che venga adottato un modello a cascata o iterativo.

3.2 Descrizione del RUP

Il Rational Unified Process è un modello di processo software iterativo sviluppato da Rational Software. Il RUP non definisce un singolo, specifico

[3]Il modello a cascata o ciclo di vita a cascata (waterfall model o waterfall lifecycle in inglese) è un modello di ciclo di vita del software (ovvero di processo software) secondo cui la realizzazione di un prodotto software consta di una sequenza di fasi strutturata in analisi dei requisiti, progetto, sviluppo, collaudo, integrazione e manutenzione. Ciascuna di queste fasi produce un ben preciso output che viene utilizzato come input per la fase successiva. Benché gran parte delle critiche a questo modello siano oggi universalmente accettate, il ciclo di vita a cascata continua a rimanere un punto di riferimento importante, in sostanza un modello "canonico" rispetto al quale vengono spesso descritte le "variazioni" moderne; ed è spesso il primo modello di sviluppo software che si insegna agli studenti. [54]

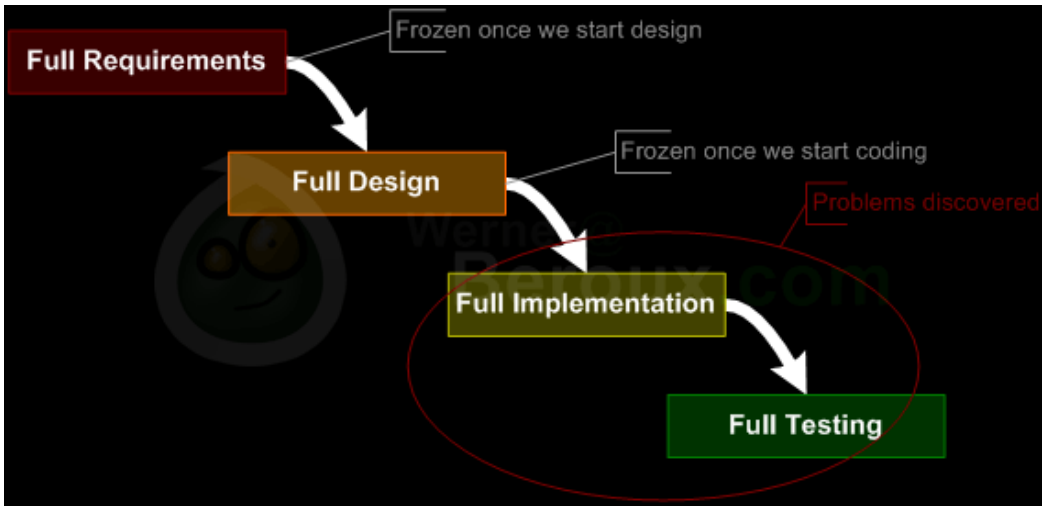


Figura 3.1: Il modello di sviluppo a cascata. [8]

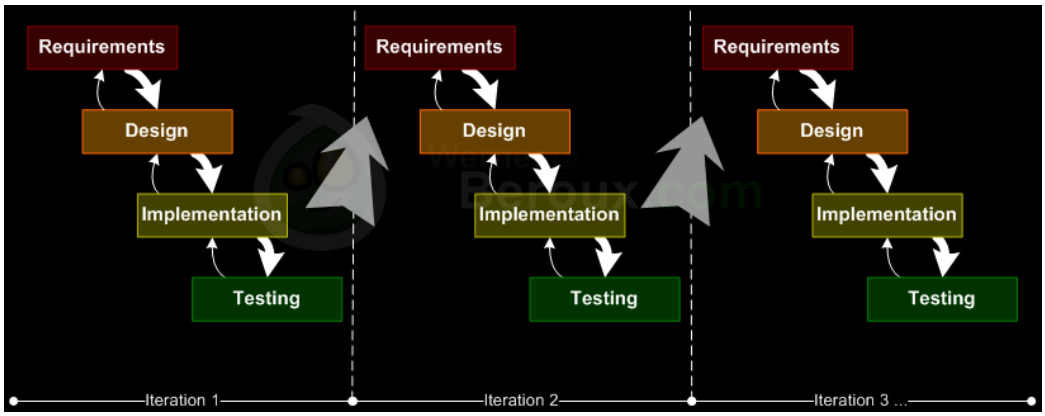


Figura 3.2: Il modello di sviluppo iterativo. [8]

processo, bensì un framework adattabile che può dar luogo a diversi processi in diversi contesti.

I creatori del RUP partirono dalla diagnosi di un campione di progetti software falliti, allo scopo di identificare cause tipiche o generali di fallimento. Quindi confrontarono questa informazione con la struttura dei processi software descritti in letteratura e applicati nella pratica, cercando di identificare le soluzioni proposte precedentemente. L'elenco dei motivi di fallimento identificati comprende per esempio:

- Gestione ad hoc (non standard) dei requisiti;

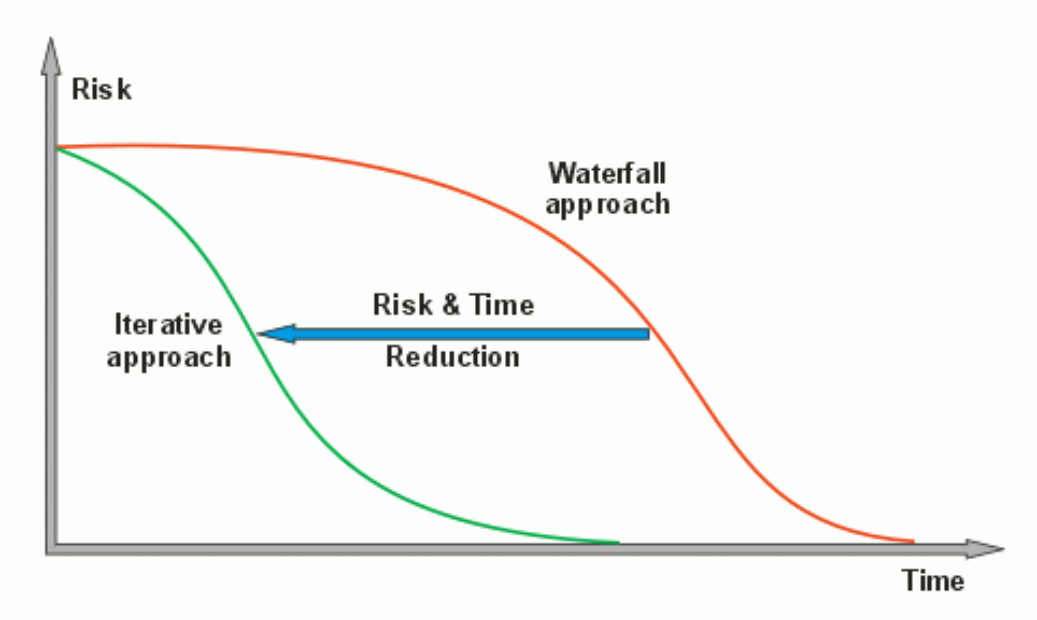


Figura 3.3: Entità del rischio in funzione dello stato di avanzamento di un progetto [41]

- Comunicazione ambigua e non precisa;
- Architettura fragile (incapace di sopportare situazioni di particolare criticità);
- Incapacità di gestire la complessità;
- Inconsistenze nei requisiti, nel progetto o nelle implementazioni;
- Collaudo insufficiente;
- Valutazione soggettiva dello stato del processo;
- Incapacità di affrontare il rischio;
- Propagazione non controllata delle modifiche;
- Insufficiente automazione;

Il RUP si può descrivere come una collezione di best practice mirate a evitare questi e altri problemi, e come un ambiente di gestione dei processi che facilita

l'applicazione di tali pratiche. Il processo fu progettato utilizzando strumenti tipici della progettazione del software; in particolare, esso fu descritto in termini di un metamodello object-oriented, espresso in UML (vedi 3.3).

Nel RUP, il ciclo di vita di un processo software viene suddiviso in cicli di sviluppo, a loro volta scomposti in fasi. Le fasi previste sono:

- Inception Phase (fase iniziale).
- Elaboration Phase (fase di elaborazione).
- Construction Phase (fase di costruzione).
- Transition Phase (fase di transizione).

Ogni fase ha un certo insieme di obiettivi e si conclude con la realizzazione di un deliverable (prodotto) di qualche genere. Le fasi sono ulteriormente scomposte in iterazioni, che sono associate a periodi temporali e hanno deadline precise.

Inception Phase

L'Inception Phase si può considerare come una particolare elaborazione e precisazione del concetto generale di analisi di fattibilità.Lo scopo principale è quello di delineare nel modo più accurato possibile il business case, ovvero comprendere il tipo di mercato al quale il progetto afferisce e identificare gli elementi importanti affinché esso conduca a un successo commerciale. Fra gli strumenti utilizzati ci sono un modello dei casi d'uso, la pianificazione iniziale del progetto, la valutazione dei rischi, una definizione grossolana dei requisiti e così via. Se il progetto non supera questa milestone, detta "Lifecycle Objective Milestone", esso dovrà essere abbandonato o ridefinito.

Elaboration Phase

La fase di elaborazione definisce la struttura complessiva del sistema.

Questa fase comprende l'analisi di dominio e un prima fase di progettazione dell'architettura. Questa fase deve concludersi con il superamento

di una milestone detta "Lifecycle Architecture Milestone". A questo scopo devono essere soddisfatti i seguenti criteri:

- Deve essere stato sviluppato un modello dei casi d'uso completo all'80%.
- Dev'essere fornita la descrizione dell'architettura del sistema.
- Dev'essere stata sviluppata un'"architettura eseguibile" che dimostra il completamento degli use case significativi.
- Dev'essere eseguita una revisione del business case e dei rischi.
- Dev'essere completata una pianificazione del progetto complessivo.

Se il progetto non passa questa milestone, potrebbe ancora essere abbandonato, oppure dovrà essere rivisitato. Al termine di questa fase si transita infatti in una situazione di rischio più elevato, in cui le modifiche all'impostazione del progetto saranno più difficili e dannose.

Construction Phase

In questa fase viene portato a termine il grosso degli sviluppi. Viene prodotta la prima release del sistema.

La milestone di questa fase si chiama "Initial Operational Capability" e rappresenta la prima disponibilità delle funzionalità del sistema in termini di implementazione.

Transition Phase

Nella fase di transizione, il sistema passa dall'ambiente dello sviluppo a quello del cliente finale. Vengono condotte le attività di training degli utenti e il beta testing del sistema a scopo di verifica e validazione. Si deve in particolare verificare che il prodotto sia conforme alle aspettative descritte nella fase di Inception. Se questo non è vero si procede a ripetere l'intero ciclo; altrimenti, si raggiunge la milestone detta "Product Release" e lo sviluppo termina.

Iterazioni

Tipicamente, un progetto gestito usando il RUP viene suddiviso in iterazioni. Questa scomposizione presenta numerosi vantaggi (in particolare rispetto alla valutazione dell'avanzamento del progetto e alla gestione dei fattori di rischio) ma implica un overhead specifico. Il RUP definisce una "Project Management Discipline" (disciplina di gestione dei progetti) a cui il project manager può affidarsi per amministrare le iterazioni.

3.3 Linguaggio di modellazione scelto

Per questo progetto è stato scelto come linguaggio di modellazione lo Unified Modeling Language (UML) [20, 29, 54].

In ingegneria del software, Unified Modeling Language ("linguaggio di modellazione unificato"), è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. Il nucleo del linguaggio fu definito nel 1996 da Grady Booch, Jim Rumbaugh e Ivar Jacobson sotto l'egida dello OMG [4], che tuttora gestisce lo standard di UML. Il linguaggio nacque con l'intento di unificare approcci precedenti, raccogliendo le best practice nel settore e definendo così uno standard industriale unificato.

UML svolge un'importantissima funzione di lingua franca nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa UML per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico.

La modellazione UML avviene principalmente attraverso diagrammi che sono divisi in due categorie, come si vede in figura 3.4:

- diagrammi di struttura: descrivono le strutture statiche dei dati, e si possono ottenere utilizzando per esempio i diagrammi delle classi, i diagrammi di dispiegamento o i diagrammi dei componenti;

[4] L'Object Management Group (OMG) è un consorzio creato nel 1989 con 440 aziende tra cui: Microsoft, Digital, HP, NCR, SUN, OSF con l'obiettivo di creare un sistema di gestione di un'architettura distribuita.

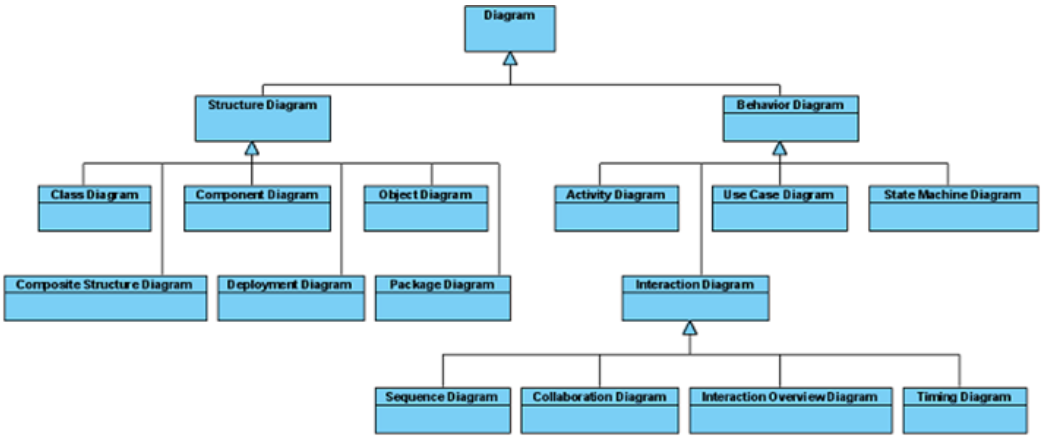


Figura 3.4: Classificazione dei diagrammi UML [44].

- diagrammi di comportamento: descrivono la collaborazione tra oggetti. Ci sono molte tecniche di visualizzazione per la modellazione del comportamento, come il diagramma dei casi d'uso, il diagramma di sequenza, il diagramma di stato, il diagramma di collaborazione e il diagramma di attività.

Di seguito verrà fornita una descrizione per tutti i tipi di diagrammi che verrano utilizzati per la modellazione del gioco.

3.3.1 Introduzione ai diagrammi dei casi d'uso

I diagrammi dei casi d'uso (Use Case Diagram (UCD)) Sono diagrammi dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

Sono impiegati soprattutto nel contesto della Use Case View (vista dei casi d'uso) di un modello, e in tal caso si possono considerare come uno strumento di rappresentazione dei requisiti funzionali di un sistema. Tuttavia, non è impossibile ipotizzare l'uso degli UCD in altri contesti; durante la progettazione, per esempio, potrebbero essere usati per modellare i servizi offerti da un determinato modulo o sottosistema ad altri moduli o sottosistemi. In molti modelli di processo software basati su UML, la Use Case View e gli

Use Case Diagram che essa contiene rappresentano la vista più importante, attorno a cui si sviluppano tutte le altre attività del ciclo di vita del software (processi del genere prendono l'appellativo di processi Use Case Driven).

3.3.2 Introduzione ai diagrammi di attività

Il diagramma di attività (Activity Diagram) definisce le attività da svolgere per realizzare una data funzionalità. Può essere utilizzato durante la progettazione del software per dettagliare un determinato algoritmo. Più in dettaglio, un diagramma di attività definisce una serie di attività o flusso, anche in termini di relazioni tra le attività, i responsabili per le singole attività ed i punti di decisione. L' activity diagram e spesso usato come modello complementare allo Use Case Diagram, per descrivere le dinamiche con cui si sviluppano i diversi use case.

3.3.3 Introduzione ai diagrammi delle classi

I diagrammi delle classi (class diagram) consentono di descrivere tipi di entità, con le loro caratteristiche, e le eventuali relazioni fra questi tipi. Gli strumenti concettuali utilizzati sono il concetto di classe del paradigma object-oriented e altri correlati (per esempio la generalizzazione, che è una relazione concettuale assimilabile al meccanismo object-oriented dell'ereditarietà).

Uno degli assunti fondamentali del paradigma a oggetti è che il concetto di classe, e concetti correlati come l'ereditarietà o il polimorfismo, si prestino a rappresentare in modo diretto e intuitivo la realtà, in qualsiasi ambito (per usare le parole di Grady Booch, "un oggetto è qualsiasi cosa a cui si possa pensare"). I diagrammi delle classi UML sono basati su versioni astratte di tali concetti, e possono essere utilizzati per descrivere sostanzialmente qualsiasi contesto a qualsiasi livello di astrazione (enfatizzandone, però, solo alcuni aspetti). Di conseguenza, UML prevede un loro impiego a livello di analisi e in particolare analisi del dominio (ovvero la descrizione del contesto in cui un sistema software deve operare), ma anche a livello di progettazione (nella descrizione della struttura interna del sistema, dei suoi componenti e delle loro relazioni).

3.3.4 Introduzione ai diagrammi di sequenza

Un diagramma di sequenza (Sequence Diagram) è un diagramma utilizzato per descrivere uno scenario. Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate; in pratica nel diagramma non compaiono scelte, né flussi alternativi.

Normalmente da ogni Activity Diagram sono derivati uno o più Sequence Diagram; se per esempio lo Activity Diagram descrive due flussi di azioni alternativi, se ne potrebbero ricavare due scenari, e quindi due Sequence Diagram alternativi.

Il Sequence Diagram descrive le relazioni che intercorrono, in termini di messaggi, tra Attori, Oggetti di business, Oggetti od Entità del sistema che si sta rappresentando.

3.3.5 Introduzione ai diagrammi di stato

Il diagramma di stato (State Chart Diagram) è un diagramma usato per descrivere il comportamento di entità o di classi in termini di stato (macchina a stati). Il diagramma mostra gli stati che sono assunti dall'entità o dalla classe in risposta ad eventi esterni.

Il concetto di stato è spesso posto in relazione a ciclo di vita; l'insieme completo di stati che un'entità o una classe può assumere, dallo stato iniziale a quello finale, ne rappresenta il ciclo di vita.

Gli elementi rappresentati da uno State Chart Diagram sono lo stato - distinguendo tra iniziale, intermedi e stato finale - l'evento, l'azione e la guardia. Lo stato descrive una qualità dell'entità o classe che si sta rappresentando (pratica aperta, in lavorazione, sospesa, chiusa); l'evento è la descrizione dell'azione che comporta il cambiamento di stato, l'azione è l'evento che ne consegue, la guardia è l'eventuale condizione che si deve verificare perché si possa compiere l'azione.

3.3.6 Introduzione ai diagrammi dei componenti

Il diagramma dei componenti (Component diagram) è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. Per componente si intende una unità software dotata di una precisa identità, nonché responsabilità e interfacce ben definite. Questi diagrammi si accompagnano spesso ai diagrammi di dispiegamento, che mostrano invece la collocazione fisica dei componenti sull'architettura di esecuzione.

3.3.7 Introduzione ai diagrammi di dispiegamento

Il diagramma di dispiegamento (Deployment Diagram) è un diagramma di tipo statico usato per descrivere un sistema in termini di risorse hardware detti nodi, e di relazioni fra di esse. Spesso si utilizza un diagramma che mostra come le componenti software sono distribuite rispetto alle risorse hardware disponibili sul sistema; questo diagramma è costruito unendo il diagramma dei componenti e il diagramma di dispiegamento.

3.4 Descrizione diagramma di Gantt

Uno dei principali strumenti di supporto che sono stati usati è il diagramma di Gantt [54].

Esso è costruito partendo da un asse orizzontale, a rappresentazione dell'arco temporale totale del progetto, suddiviso in fasi incrementali (ad esempio, giorni, settimane, mesi), e da un asse verticale, che rappresenta delle mansioni o attività che costituiscono il progetto. Barre orizzontali di lunghezza variabile rappresentano le sequenze, la durata e l'arco temporale di ogni singola attività del progetto (l'insieme di tutte le attività del progetto ne costituisce la Work Breakdown Structure). Queste barre possono sovrapporsi durante il medesimo arco temporale ad indicare la possibilità dello svolgimento in parallelo di alcune delle attività. Man mano che il progetto progredisce, delle barre secondarie, delle frecce o delle barre colorate possono essere aggiunte al diagramma, per indicare le attività sottostanti completate

o una porzione completata di queste. Una linea verticale è utilizzata per indicare la data di riferimento.

Un diagramma di Gantt permette dunque la rappresentazione grafica di un calendario di attività, utile al fine di pianificare, coordinare e tracciare specifiche attività in un progetto dando una chiara illustrazione dello stato d'avanzamento del progetto rappresentato; di contro, uno degli aspetti non tenuti in considerazione in questo tipo di diagrammazione è l'interdipendenza delle attività, caratteristica invece della programmazione reticolare, cioè del diagramma Pert[5]. Ad ogni attività possono essere in generale associati una serie di attributi: durata (o data di inizio e fine), predecessori, risorsa, costo. Ad ogni attività possono essere associate una o più risorse. Alcuni software disponibili sul mercato permettono di visualizzare il carico di lavoro di ogni risorsa e la sua saturazione, impostando una certa disponibilità per ogni risorsa. Contestualmente, può essere definito il calendario dei giorni lavorativi e festivi, e il numero di ore di lavoro giornaliere.

3.5 Descrizione Cocomo II

Per ottenere una preliminare previsione dei costi di sviluppo del software è stato utilizzato il sistema COCOMO II (COnstructive COst MOdel) [54]. Esso offre la possibilità di stimare la quantità di risorse umane richieste dall'intero processo di sviluppo software quantificandole in termini di sforzo, durata e costo economico. Lo sforzo viene espresso in Person-Months (mesi persona), quantità di lavoro effettuata da una persona in un mese, la durata totale viene misurata in mesi e il costo viene espresso in Dollari.

[5]Pert sta per Program Evaluation and Review Technique, è una tecnica con cui si tengono sotto controllo le attività di un progetto utilizzando una rappresentazione reticolare che tiene conto della interdipendenza tra tutte le attività necessarie al completamento del progetto.

Capitolo 4

Inception Phase

L'obiettivo di questa fase dello sviluppo è di gettare le fondamenta dell'applicazione, in maniera che essa possa crescere sostenuta da una base stabile.

Si inizierà stabilendo le caratteristiche di base che dovrà avere l'applicazione e studiando la sua possibile collocazione nel mercato (paragrafo 4.1). Al termine della fase di Inception bisognerà aver ridotto considerevolmente la probabilità di fallimento del progetto, procedendo innanzitutto all'individuazione dei rischi più rilevanti (paragrafo 4.2), alla valutazione della fattibilità del progetto (paragrafo 4.3) e all'identificazione dei requisiti più critici (paragrafo 4.6).

Contemporaneamente verrà effettuata una pianificazione del lavoro (paragrafo 4.4) ed una stima delle risorse necessarie (paragrafo 4.5). In seguito si potrà procedere con la progettazione (paragrafo 4.7) che consisterà principalmente nel concepimento di un idea di alto livello per l'architettura del sistema, la quale costituirà la pietra miliare della fase di Inception.

4.1 Visione

Il software che si è deciso di creare dovrà essere un gioco a due dimensioni destinato ai dispositivi mobili Android.

Il gioco dovrà consistere nel muovere un'entità (per esempio un personaggio o un oggetto) in direzione orizzontale, cercando di prendere oppure evitare degli oggetti che si muoveranno verso di esso in direzione verticale. Gli oggetti, denominati agenti, dovranno appartenere a diverse tipologie, il contatto con essi quindi avrà effetti diversi a seconda della tipologia.

L'obiettivo sarà il rilascio di un gioco che avrà le caratteristiche generali sopraccitate, sulla base delle quali si è deciso che la categoria del gioco dovrà essere riconducibile a quella dei casual game (si veda [56]). Si ritiene che la categoria dei casual game permetta un buon posizionamento nello specifico modello di mercato a cui afferisce Android (si veda il paragrafo 1.2.1). Lo studio del tema del gioco e di tutte le sue caratteristiche indipendenti dall'architettura verrano effettuati nella fase di Elaboration.

4.2 Analisi dei rischi

L'analisi dei rischi è di vitale importanza dato che permette di individuare le possibili cause di fallimento di un progetto preventivamente alla mobilitazione di tutte le risorse necessarie alla sua realizzazione.

Fortunatamente i rischi legati a questo progetto non sono eccessivi soprattutto perché il contesto in cui verrà inserito (vedi 1.2.1) non prevede tra i requisiti di accesso investimenti di grossa entità.

Inoltre le specifiche (vedi 4.1) non sono troppo vincolanti e lasciano un buon grado di libertà allo sviluppatore che dovrà saper gestire in maniera ottimale le risorse a sua disposizione.

Tuttavia un rischio abbastanza evidente è quello legato all'eventuale scarso apprezzamento del prodotto (a prescindere dalle cause di tale dato di fatto), che attraverso il passa-parola negativo[1] e ad altre caratteristiche proprie dei canali di distribuzione delle applicazioni Android (paragrafo 2.4.2) è in grado di decretare il fallimento del progetto, vanificandone gli sforzi.

[1]Nell'ambito del marketing il passaparola (indicato con l'espressione word of mouth) indica il diffondersi, attraverso una rete sociale, di informazioni e/o consigli tra consumatori. Le esperienze negative vengono trasmesse con maggiore frequenza, ed è per questo che il passa-parola negativo è così temibile. [13, 54]

Per scongiurare questo rischio (incombente) si è deciso che l'applicazione passerà attraverso una fase di "post-beta testing"[2], prima di essere rilasciata al pubblico.

4.3 Studio della fattibilità

Prima di decidere se intraprendere o meno il progetto è stata effettuata un'approfondita documentazione sullo sviluppo di applicazioni per Android [34, 6, 1], con particolare attenzione alle questioni relative allo sviluppo di giochi [21, 22, 11, 18].

Le principali problematiche che sono state individuate durante questa fase documentativa possono essere suddivise in due livelli di dettaglio: il primo riguarda lo sviluppo di applicazioni per dispositivi mobili in generale, il secondo livello è relativo allo sviluppo di giochi per la piattaforma Android.

Per comprendere al meglio la parte 4.3.1 si tenga conto delle considerazioni fatte nei paragrafi 1.2.2, 2.4.1.

4.3.1 Problematiche nello sviluppo per i device mobili

- Prestazioni limitate.

  La legge di Moore[3] si applica anche ai dispositivi mobili, ma in quel campo, gli avanzamenti tecnologici sono prevalentemente volti ad ottenere dispositivi più compatti ed efficienti, piuttosto che ad aumentare in maniera significativa la pura potenza di calcolo.

  Diversa è la situazione nell'ambito dei desktop e dei server: l'aumento della densità dei transistor viene sfruttato per avere effetti più diretti sulle prestazioni.

[2]Il beta testing è una fase di prova e collaudo di un software non ancora pubblicato, con lo scopo di trovare eventuali errori. La distribuzione della versione beta del software può essere ristretta ad un piccolo numero di persone oppure accessibile a tutti, a seconda dei casi [54].

[3]La legge di Moore che prende il nome dal cofondatore di Intel, Gordon Moore, afferma: "le prestazioni dei processori, e il numero di transistor ad essi relativi, raddoppiano ogni 18 mesi". Per altre informazioni vedere [33, 54]

Nelle piattaforme mobili, ottenere applicazioni efficienti è di importanza critica, dato che i difetti di un codice poco ottimizzato sono molto più evidenti, in quanto non possono essere mascherati e compensati da parte del lato hardware.

Le ottimizzazioni devono coinvolgere tutti gli aspetti del programma: da quelli macroscopici e di alto livello, come il costo computazionale degli algoritmi, a quelli di più basso livello, apparentemente irrilevanti, come la scelta del modificatore di una variabile.

- Memoria limitata, trade-off[4] tra spazio e tempo.

  Nell'ambito dei calcolatori tradizionali è possibile sacrificare l'efficienza in termini di spazio a vantaggio del tempo e viceversa. Nei device mobili anche la memoria (sia primaria che secondaria) è soggetta a limitazioni, quindi non sempre è conveniente utilizzare strutture dati più efficienti in termini di tempo a scapito dello spazio occupato. Inoltre non si può sempre contare sulla compressione di risorse per contenere le dimensioni complessive dell'applicazione, dato che l'accesso ad esse potrebbe risultare eccessivamente lento per via delle operazioni di decompressione.

- Display di dimensioni ridotte.

  Un'ulteriore sfida nello sviluppare applicazioni per device mobili sta nel realizzare applicazioni dalla buona usabilità[5], che ha stretti legami con la qualità dell'interfaccia grafica.

  Le piccole dimensioni dei display di certo non agevolano il compito dello sviluppatore, il quale deve calibrare ogni aspetto dell'interfaccia in maniera tale che essa risulti facile da utilizzare e contemporaneamente gradevole alla vista. Ad esempio la presenza di touch-screen ottimizzati

[4] Un trade-off (o trade off) è una situazione che implica una scelta tra due o più alternative, in cui la perdita di valore di una costituisce un aumento di valore in un'altra. [54]

[5] L'usabilità è definita dall'ISO (International Organisation for Standardisation), come l'efficacia, l'efficienza e la soddisfazione con le quali determinati utenti raggiungono determinati obiettivi in determinati contesti. In pratica definisce il grado di facilità e soddisfazione con cui l'interazione uomo-strumento si compie. [54]

per le dita implica che la dimensione dei vari elementi grafici deve essere maggiore rispetto al caso in cui il touch-screen sia concepito per essere utilizzato con un pennino.

- Gli utenti dei dispositivi mobili sono diversi da quelli dei PC.

  Gli utenti dei device mobili sono solitamente più esigenti in termini di reattività delle applicazioni, soprattutto perché non sono seduti comodamente davanti ad un PC, ma potrebbero avere bisogno urgente di una qualche funzionalità, di conseguenza la loro pazienza è limitata. Attese anche soltanto di pochi secondi potrebbero essere sufficienti per un utente a reputare di bassa qualità una determinata applicazione.

4.3.2 Problematiche nello sviluppo di giochi su Android

- Frammentazione della realtà Android.

  Come già detto in precedenza, l'apertura è uno dei principi basilari di Android che permette una completa personalizzazione della piattaforma, ma un effetto collaterale dell'eccessiva personalizzazione consiste nella formazione di tante piccole realtà incompatibili tra di loro.

  Questa frammentazione riguarda vari aspetti della piattaforma, i dispositivi esistenti sono notevoli e sono caratterizzati da diversi sistemi di input, da schermi dalle diverse risoluzioni e da diverse versioni di android installate.

4.3.3 Risultati dello studio di fattibilità

Dopo un'attenta analisi è stato deciso di affrontare lo sviluppo dell'applicazione dato che si è ritenuto che le problematiche sopraelencate non sono eccessivamente vincolanti.

Tuttavia nel momento in cui verrà stilata la lista dei requisiti, tali problematiche influenzeranno pesantemente le decisioni in merito a tutti gli aspetti del gioco.

4.4 Pianificazione delle attività da svolgere

La pianificazione delle attività e stata effettuata attraverso la realizzazione di un diagramma di Gantt (3.4). La figura 4.1 mostra il diagramma di Gantt del progetto, il tool che è stato scelto per realizzare tale diagramma è Gantt Project [28].

Date le moderate dimensioni di questo progetto non è stata fatta una pianificazione eccessivamente granulare che porta i suoi benefici solo nel caso di grandi progetti. In questo modo è stato possibile contenere i costi di overhead[6] derivanti dalla gestione del progetto.

Il progetto è stato diviso in quattro fasi consecutive.

Ogni fase corrisponde ad una macro-attività del diagramma Gantt, ed è costituita da una o più iterazioni. Contemporaneamente alle attività di seguito elencate, è prevista un'attività di documentazione, essa viene svolta in maniera continuativa durante tutta la durata del progetto.

Fase di Inception

Questa fase iniziale verrà svolta in un'unica iterazione e dovrà essere completata in dieci giorni. Le attività di cui è composta sono:

- studio del business case e degli obiettivi di massima del progetto;
- analisi dei rischi;
- studio della fattibilità;
- stima dei costi e pianificazione;
- specifica dei requisiti;
- progettazione.

[6] Risorse spese in attività necessarie alla gestione di una certa operazione.

Fase di Elaboration

Questa fase è costituita da un'unica iterazione e dovrà essere completata in un mese. Le attività di cui è composta sono:

- studio del tema del gioco;
- specifica dei requisiti;
- progettazione;
- implementazione architettura di base;
- testing e debugging.

Per i dettagli della fase di Elaboration si veda il capitolo 5.

Fase di Construction

Questa fase è costituita da N iterazioni e dovrà essere completata in trentotto giorni. Questa fase non è stata pianificata in modo altamente granulare dato che sarebbe stato particolarmente oneroso calcolare preventivamente il numero di iterazioni di raffinamento che la compongono. La fase di Construction quindi corrisponde ad una singola attività nel grafico ma è bene tenere conto che essa nella realtà sarà formata da numerose attività che si susseguono ciclicamente fino al raggiungimento del prodotto finito.

Per i dettagli della fase di Construction si veda il capitolo 6.

Fase di Transition

Questa fase finale è costituita da un'unica iterazione e dovrà essere completata in sette giorni. Le attività di cui è composta sono:

- gestione equilibri di gioco;
- valutazione;
- rilascio.

Per i dettagli della fase di Transition si veda il capitolo 7.

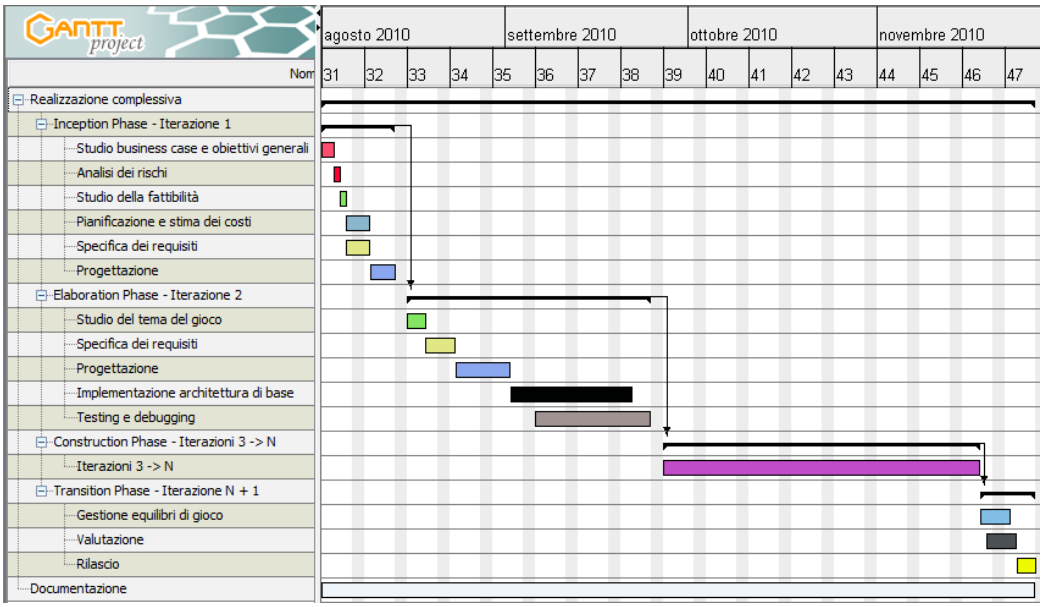


Figura 4.1: Il diagramma di Gantt del progetto.

4.5 Stima dei costi

Il tool COCOMO II si presenta come un interfaccia in cui inserire numerosi parametri necessari a realizzare la stima, come guida alla scelta dei valori di tali parametri si è utilizzato il manuale di Cocomo II [50].

La figura 4.2 mostra i valori che sono stati inseriti per i vari parametri. Il primo dato da inserire riguarda la grandezza del software che dovrà essere sviluppato, misurata in Source Lines Of Code (SLOC), ovvero righe di codice sorgente.

È stato scelto di inserire un numero pari a 2500 per ottenere una stima massimale dello sforzo, questa cifra verrà confrontata con l'effettivo numero di righe che costituiranno il prodotto finito. Non essendoci la previsione di riutilizzare o modificare codice già esistente, sono stati lasciati vuoti i campi relativi al numero di righe modificate e riutilizzate.

Successivamente ci sono vari parametri denominati "cost drivers", ovvero fattori che determinano il costo, sono divisi in categorie. La scelta di alcuni dei parametri è motivata di seguito:

- nella famiglia "Product" per il parametro "Required Software Relia-

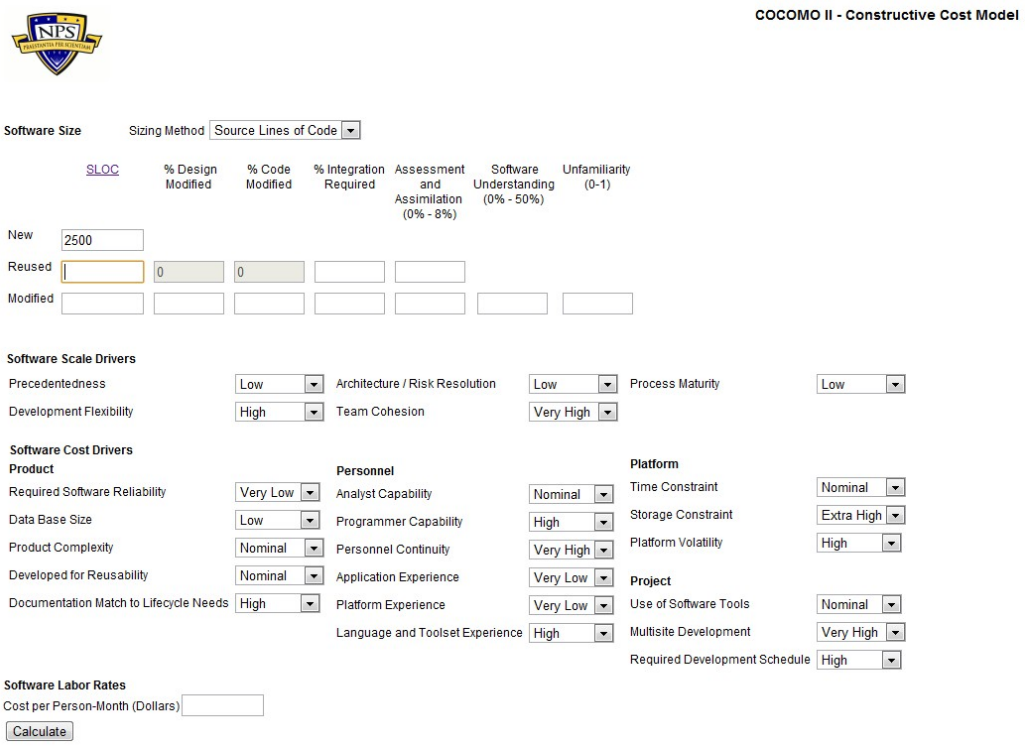


Figura 4.2: Schermata che mostra i valori inseriti nel web-tool COCOMO II

bility" è stato scelto il grado minore, "Very Low" questo perché l'affidabilità richiesta al software da realizzare non è alta, in quanto un eventuale malfunzionamento di esso non può causare danni di alcun tipo, trattandosi di un gioco.

- Nella famiglia "Personnel" per il parametro "Platform Experience" è stato scelto il grado "Very Low", dato che lo sviluppatore non ha esperienze precedenti per quanto riguarda la piattaforma Android. Per il parametro "Language and Toolset Experience" è stato selezionato il grado "High" dato che lo sviluppatore può contare su una buona esperienza con il linguaggio e degli strumenti di sviluppo.

Il campo che fa riferimento al salario è stato lasciato vuoto visto che lo sviluppatore non è alle dipendenze di alcun datore di lavoro.

I risultati ottenuti sono mostrati dalla figura 4.3:

Results

Software Engineering

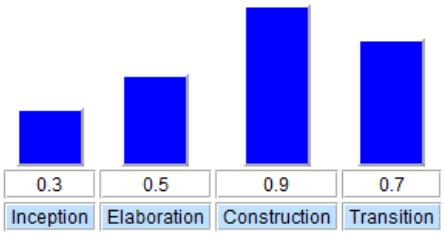
Effort = 6 Person-months
Schedule = 8 Months
Cost = $0

Total Equivalent Size = 2500 SLOC

Phase Distribution

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	0.4	1.1	0.3	$0
Elaboration	1.5	3.3	0.5	$0
Construction	4.7	5.4	0.9	$0
Transition	0.7	1.1	0.7	$0

Staffing by Phase



Software Effort Distribution for RUP/MBASE (Person-Months)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.1	0.2	0.5	0.1
Environment/CM	0.0	0.1	0.2	0.0
Requirements	0.1	0.3	0.4	0.0
Design	0.1	0.5	0.7	0.0
Implementation	0.0	0.2	1.6	0.1
Assessment	0.0	0.1	1.1	0.2
Deployment	0.0	0.0	0.1	0.2

Figura 4.3: Output della stima dei costi attraverso COCOMO II

Lo sforzo risultante è pari a 6 Mesi-Persona distribuiti in un periodo di 8 mesi. Si presume che un costo stimato così consistente è attribuibile alla scarsa esperienza sia nel settore specifico che nella piattaforma.

4.6 Specifica dei requisiti

In questa fase di Inception sono stati individuati parte dei requisiti che dovrà avere l'applicazione, in particolare sono stati identificati quelli più critici.

I requisiti sono stati suddivisi in due gruppi, il primo gruppo è un elenco di requisiti volti a delineare il gioco, il secondo è un elenco di requisiti di natura tecnica.

Caratteristiche essenziali del gioco

- dovrà essere sviluppato un gioco per la piattaforma Android;
- il gioco dovrà essere 2D;
- il gioco dovrà far cadere degli oggetti verticalmente, essi verranno denominati agenti;
- dovranno esserci varie tipologie di agente, ognuna avente comportamenti diversi;
- il giocatore dovrà controllare un "Personaggio" o entità che potrà muovere orizzontalmente per prendere o evitare gli agenti;
- il contatto fra il "Personaggio" e un oggetto dovrà avere effetti diversi a seconda della tipologia dell'oggetto.

Requisiti tecnici

- Il gioco dovrà essere compatibile con tutti i dispositivi android a partire dalla versione 1.5.

  Questo è necessario per non escludere tutti gli utenti ancora legati alla versione 1.5, che al momento costituiscono circa il 25% del mercato (vedi figura 4.4).

- Dovranno essere soddisfatti requisiti minimi di reattività ed efficienza.

  Verranno prese come riferimento le linee guida fornite direttamente da Android [14, 15].

- I sistemi di input supportati dovranno essere touch-screen, tasti direzionali e tastiera fisica, in maniera da garantire compatibilità con il maggior numero possibile di dispositivi.
- Dovranno essere supportate tutte le risoluzioni esistenti attualmente.
- Dovranno essere rispettati anche tutti gli altri canoni di qualità consigliati da Android (vedi [3, 51, 47, 16]).

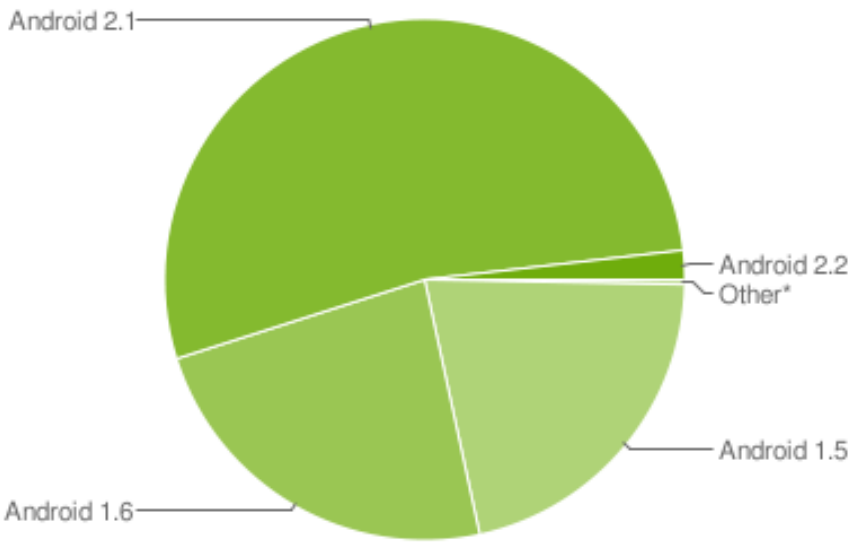


Figura 4.4: La suddivisione del mercato in base alle versioni, dati relativi al mese di luglio 2010 [5]

4.7 Progettazione

In questa prima fase di progettazione sono stati definiti alcuni casi d'uso del gioco ed è stato progettato il sistema che permetterà di ottenere le animazioni, che sono l'essenza del gioco. Oltre a quest'ultima, l'attività più rilevante di questa fase di progettazione è l'ottenimento di un'architettura di alto livello. Partendo dal meccanismo di realizzazione delle animazioni è stata costruita l'ossatura del gioco, prevedendo per il momento solo i componenti più critici.

4.7.1 Diagramma dei casi d'uso

Il diagramma in figura 4.5 è una rappresentazione dei casi d'uso relativi alla partita.

Il giocatore ha la possibilità di muovere il "Personaggio" (per personaggio si intende l'entità da lui controllata). Il personaggio sarà definito nel momento in cui verrà concepito il tema del gioco.

Il gioco farà cadere degli agenti e dovrà rilevare gli eventuali contatti tra questi ultimi e l'entità controllata dal giocatore. Nel momento in cui viene rilevato un contatto, il gioco si occupa di attuarne gli effetti a seconda del tipo di agente con cui è avvenuto il contatto.

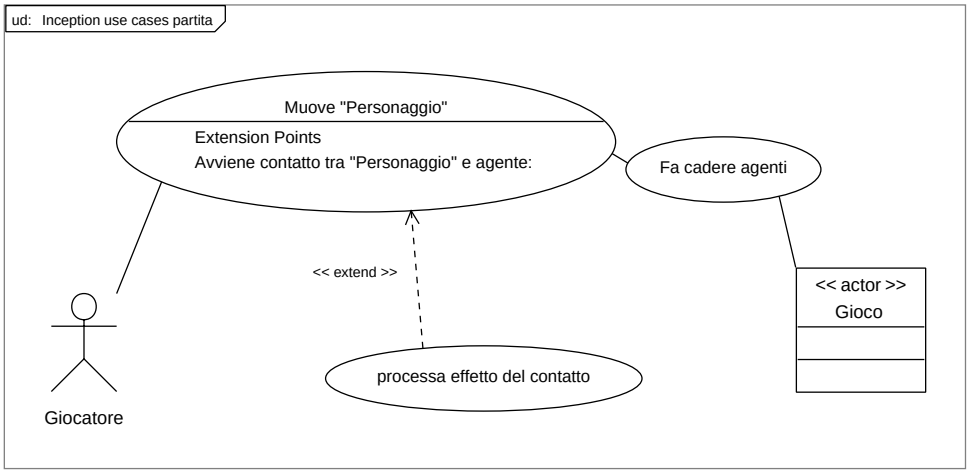


Figura 4.5: Diagramma dei casi d'uso prodotto durante la fase di Inception

4.7.2 Diagramma di sequenza

Attraverso il diagramma UML di sequenza in figura 4.6 è stato possibile modellare la dinamica che sta alla base dell'andamento delle animazioni del gioco.

L'applicazione sarà costituita da due thread, uno che si occuperà dell'intero gioco (User Interface Thread), l'altro (Timer Thread) avrà lo scopo di innescare l'aggiornamento della grafica del gioco.

Il Timer Thread non sarà altro che un thread incapsulato all'interno di un oggetto Timer, il quale fornisce una serie di funzionalità utili agli scopi

dell'applicazione, come la pianificazione di un'operazione ad intervalli regolari di lunghezza p (millisecondi). Verrà pianificata l'esecuzione del Timer in maniera che ad ogni sua esecuzione al tempo T_i (che avverrà diverse volte per ogni secondo) verrà invalidata la schermata corrente, cosicché il componente che si occuperà di gestire il layout dell'applicazione sarà notificato e invocherà il ridisegno della schermata relativo alla situazione della partita al tempo $T_i + X$ (dove X è il tempo di overhead che trascorre dall'invalidazione alla chiamata del ridisegno), dopo che sarà trascorso di nuovo un tempo p, il Timer invaliderà nuovamente la schermata (Siamo quindi al tempo T_i+X+p), la procedura descritta costituisce a tutti gli effetti il ciclo dell'animazione del gioco.

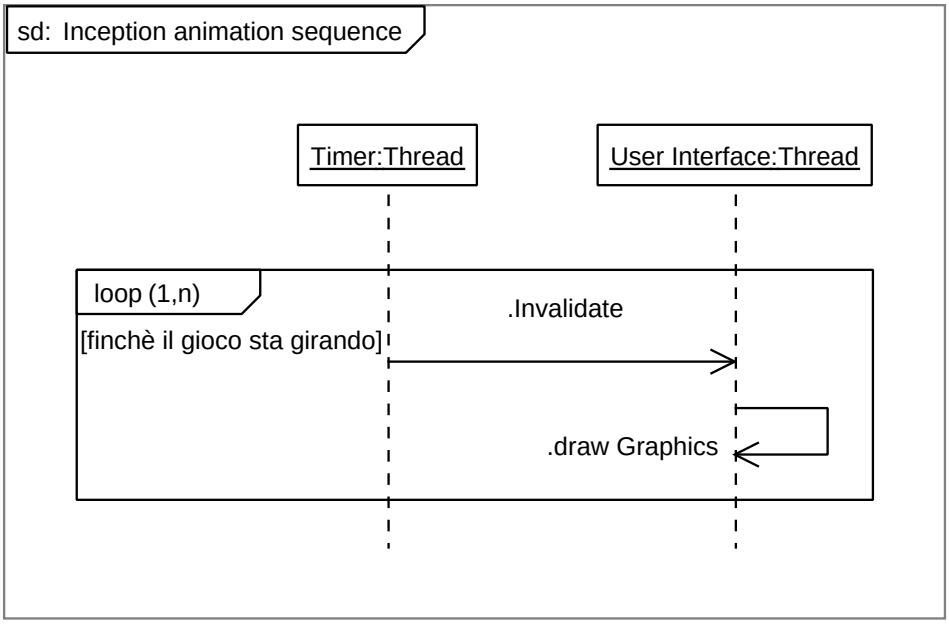


Figura 4.6: Diagramma di sequenza che illustra la sequenza di animazione.

4.7.3 Studio dell'architettura

Uno degli scopi primari di questa fase di Inception è di produrre un'architettura di alto livello che costituisca l'anima dell'intero gioco.

La struttura dell'applicazione che verrà realizzata si basa sull'architettura standard delle applicazioni android, a cui verranno aggiunti degli elementi atti a costituire l'architettura caratteristica di un gioco. Questi elementi riguardano soprattutto la gestione delle animazioni.

Come è possibile vedere dalla figura 4.7, l'applicazione sarà formata dai seguenti componenti principali:

- Una classe di tipo Activity che farà da canale di collegamento tra il gioco e il resto della piattaforma Android, essa dovrà caricare il layout del gioco e gestire in maniera opportuna tutte le sue schermate.
- Un file XML che definirà gli elementi base dell'interfaccia grafica del gioco, esso verrà caricato dalla classe di tipo Activity dell'applicazione.
- Un Layout Manager, che rappresenterà il nucleo del gioco, si dovrà occupare principalmente di disegnare la grafica del gioco attraverso il meccanismo di animazione descritto nel paragrafo 4.7.2 che sta alla base del cosiddetto "main loop"[7], ad ogni iterazione di quest'ultimo corrisponderà un aggiornamento dello stato della partita.
- Un Timer appartenente alla libreria standard Java, verrà usato per invalidare la schermata correntemente visualizzata in modo da generare un ridisegno forzato della grafica del gioco per passare all'iterazione successiva del main loop. Il main loop verrà definito in dettaglio, nel paragrafo 5.3.6 della fase di Elaboration.

ss

[7]Il main game loop è il componente centrale di qualsiasi gioco. Regola l'esecuzione di tutte le sotto-attività che formano un gioco.

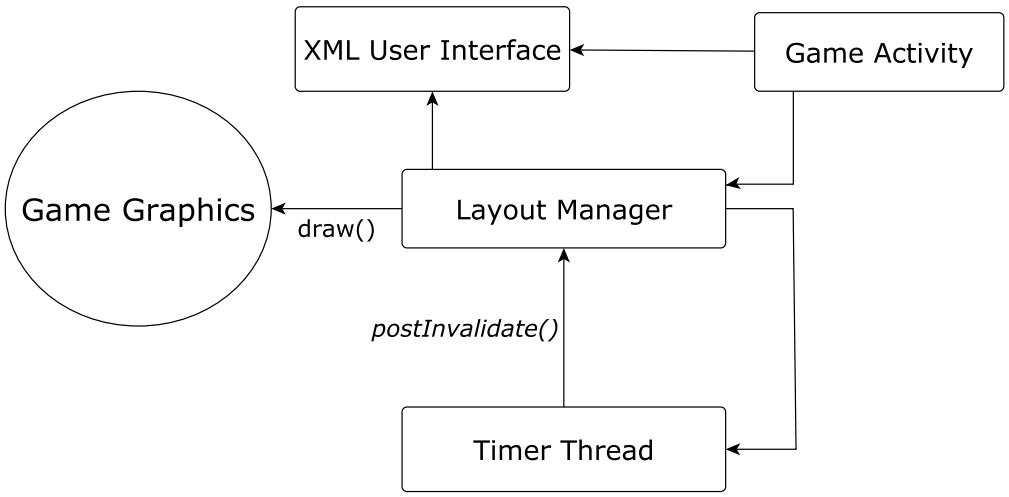


Figura 4.7: Il nucleo dell'architettura dell'applicazione

Capitolo 5

Elaboration Phase

Gli obiettivi principali di questa fase di Elaboration sono due: ideare tutti gli elementi caratterizzanti del gioco e implementare una solida ossatura dell'applicazione basata sui risultati della fase di Inception.

Il tema del gioco verrà studiato nel paragrafo 5.1, successivamente, nel paragrafo 5.2 verranno definiti i requisiti che dovrà avere il gioco in base al tema deciso.

Nel paragrafo 5.3 verrà portata avanti la modellazione UML in maniera da ottenere abbastanza elementi per poter realizzare il primo prototipo dell'applicativo paragrafo 5.4. Nel paragrafo 5.5, infine, si provvederà a verificare il funzionamento del software prodotto e all'illustrazione delle sue principali funzionalità.

5.1 Ricerca del tema del gioco

L'idea del gioco prodotta nella fase di Inception costituisce soltanto un punto di partenza da cui iniziare a plasmare il gioco come forma di intrattenimento.

Quest'attività non è stata svolta precedentemente visto che non avrebbe avuto senso impiegare risorse nella sua realizzazione prima di avere la certezza che il progetto sarebbe stato avviato.

Adesso che sono stati coperti gli aspetti decisivi per la buona riuscita del progetto, è possibile prendere in considerazione anche gli aspetti secondari,

come il tema del gioco.

Lo studio del tema del gioco consiste nel definire tutti i fattori caratterizzanti il gioco.

Innanzitutto sono state scartate tutte le possibilità che avrebbero determinato la complicazione del processo di realizzazione.

L'idea che è stata scelta permette di mantenere un moderato livello di complessità grafica senza troppi compromessi.

Il gioco dovrà consistere nel controllare un carrello della spesa e raccogliere degli oggetti raffiguranti prodotti diversi, che cadono verso il basso.

Gli agenti quindi possono assumere il ruolo di due diverse tipologie di prodotti:

- Frutta, che avrà effetti positivi, come l'aumento del punteggio.
- Junk food[1], che avrà effetti negativi, come la perdita degli Hit Points (HP).

Il giocatore avrà un certo numero di HP al termine dei quali la partita avrà fine.

Utilizzare elementi non animati come i cibi, renderà più semplice lo sviluppo dell'applicazione, senza introdurre una limitazione artificiale, dato che i cibi sono "statici" per natura.

Inoltre è stato deciso che il nome del gioco sarebbe stato "Catching Cherries".

5.2 Specifica dei requisiti

I requisiti individuati nella fase di Inception (vedi 4.6) non costituiscono una lista esaustiva di tutte le funzionalità dell'applicazione e soprattutto non tengono conto del tema del gioco.

Tuttavia i requisiti di questa fase di Elaboration sono soltanto una estensione dei requisiti fondamentali concepiti nella precedente fase.

Di seguito sono riportati i requisiti del gioco Catching Cherries:

[1]La parola junk food è un termine informale che si riferisce ad una determinata categoria di cibi che si ritiene abbiano proprietà nutritive quasi nulle

- Ogni partita dovrà terminare nel momento in cui gli HP del giocatore risulteranno minori o uguali a zero.
- Il gioco dovrà avere una difficoltà crescente in base al punteggio del giocatore.
- La difficoltà del gioco dovrà essere percepita in termini di velocità degli agenti e numero di agenti presenti sullo schermo contemporaneamente.
- L'entità controllata dal giocatore dovrà rappresentare un carrello della spesa.
- Dovranno esserci tre categorie di agenti: quelli che conferiscono punteggio, quelli che conferiscono HP e quelli che detraggono HP.
- Nel momento in cui si verifica un contatto tra il carrello ed un agente dovrà essere riprodotto un suono diverso a seconda della categoria dell'agente.
- Gli agenti che conferiscono punteggio dovranno rappresentare dei frutti di vario tipo, il punteggio conferito dovrà essere diverso a seconda del tipo di frutto.
- Gli agenti che conferiscono HP dovranno rappresentare una ciliegia.
- Gli agenti che detraggono HP dovranno rappresentare varie forme di junk food, il cosiddetto "cibo spazzatura". Gli HP detratti dovranno variare a seconda del tipo di cibo.
- Il gioco dovrà scegliere casualmente gli agenti creati di volta in volta, inoltre la coordinata orizzontale in cui viene collocato ogni nuovo agente dovrà anch'essa essere generata in maniera casuale.

5.3 Progettazione

In questa fase di progettazione verrano modellati i comportamenti e la struttura dell'applicazione. Non si terrà conto di tutti i requisiti, come per esempio

le funzionalità audio, in quanto ritenute aggiunte che posso essere fatte in un'iterazione successiva del processo di sviluppo. In questa fase sono state fatte le scelte progettuali più critiche per l'intero progetto.

5.3.1 Diagramma dei casi d'uso

Il diagramma UML in figura 5.1 rappresenta i casi d'uso relativi alla partita ed è un'ampliazione del diagramma 4.7.1.

Nel momento in cui viene rilevato un contatto tra il carrello controllato dal giocatore ed un agente, il sistema procede ad identificare la tipologia dell'agente in questione e ad effettuare una delle seguenti operazioni a seconda della sua natura:

- (a) Incrementare gli Hit Points del giocatore, se l'agente ha le sembianze di una ciliegia.
- (b) Incrementare il punteggio del giocatore, se l'agente ha le sembianze di uno dei due tipi di frutta al di fuori della ciliegia.
- (c) Decrementare gli Hit Points del giocatore, se l'agente ha le sembianze di uno dei tre tipi di "junk food".

Se si è nei casi (b) o (c), il sistema deve effettuare un'ulteriore controllo. Nel caso (b), il sistema deve verificare se il giocatore ha superato la soglia di punteggio richiesta per entrare nel livello di difficoltà superiore, in caso positivo il livello di difficoltà verrà aumentato. Nel caso (c), il sistema è tenuto a controllare se il giocatore ha esaurito gli Hit Points, in caso positivo verrà lanciato il game over.

5.3.2 Diagramma di attività

Il diagramma di attività in figura 5.2 rappresenta lo stesso scenario descritto dal diagramma dei casi d'uso 5.3.1, da una prospettiva che pone in maggiore risalto l'andamento del flusso di esecuzione.

In particolare è possibile notare che l'unico modo per porre fine alla partita è esaurire completamente gli Hit Points, non esiste quindi una condizione

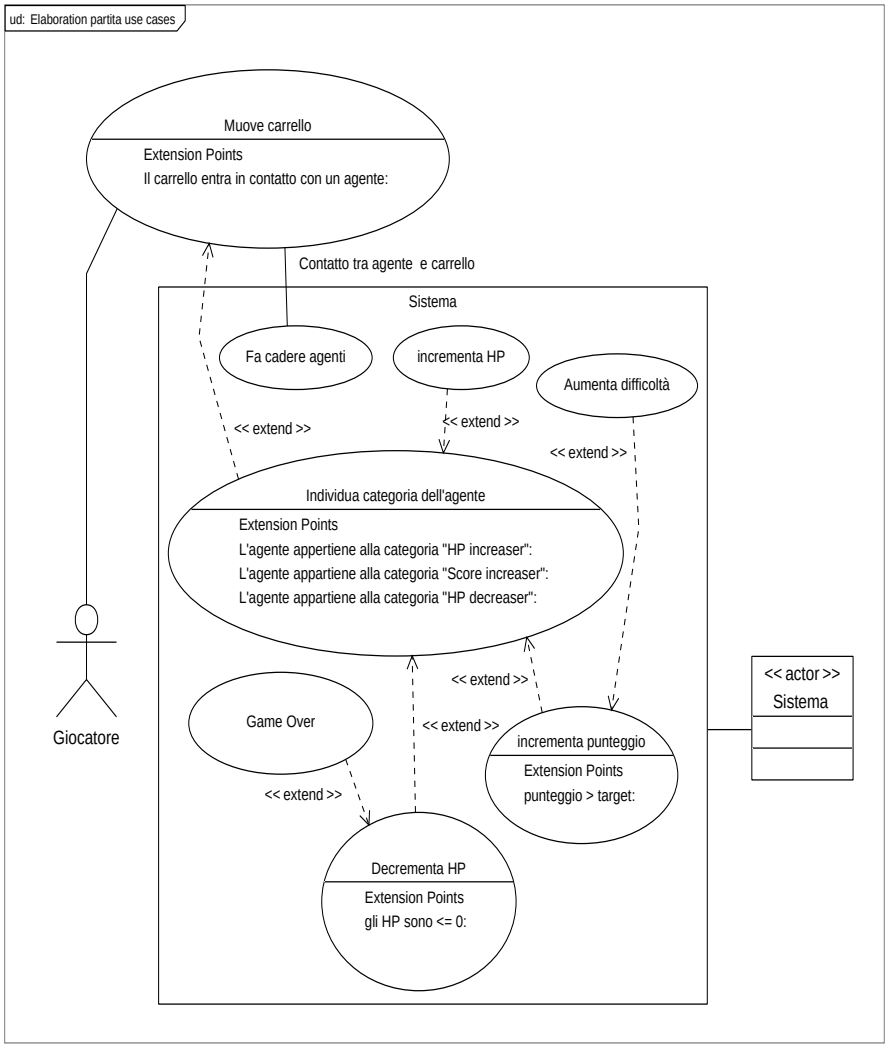


Figura 5.1: Diagramma dei casi d'uso relativi alla partita, prodotto nella fase di Elaboration

di "vittoria assoluta" ma soltanto una di tipo relativo al punteggio ottenuto. In questo modo il gioco è teoricamente infinito ed il giocatore sarà sempre stimolato a migliorare il proprio punteggio.

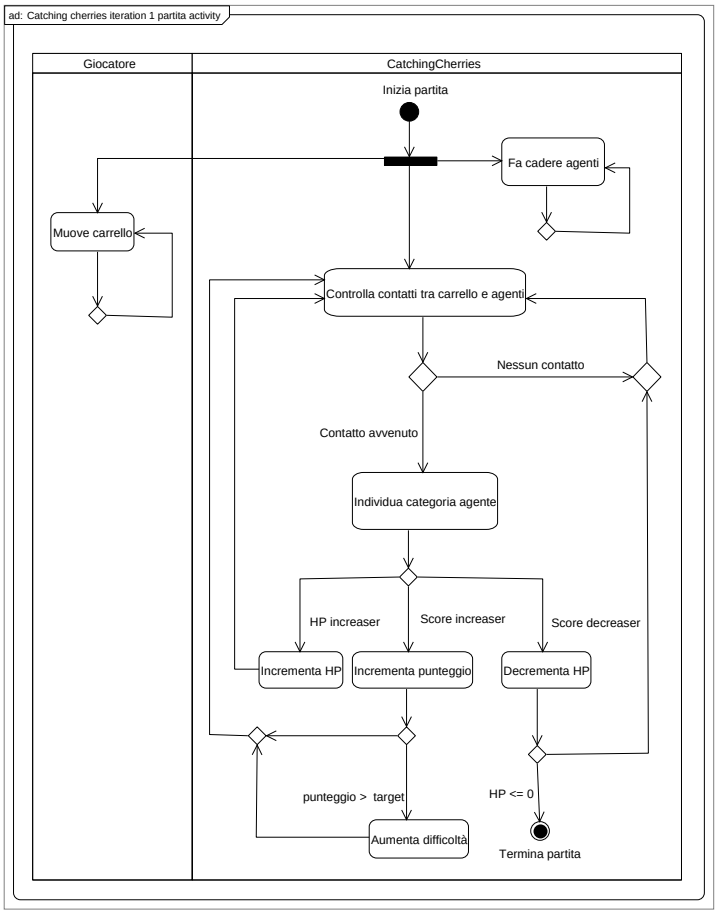


Figura 5.2: Diagramma di attività relativo alla partita, prodotto nella fase di Elaboration

5.3.3 Diagramma delle classi

Il diagramma in figura 5.3 è stato usato per modellare la struttura di base del gioco. La classe GameActivity sarà la Activity (si veda 2.3.2) dell'applicazione, i suoi compiti di base sono descritti nel paragrafo 4.7.3.

La classe astratta AbstractGame dovrà incorporare funzionalità comuni a tutti i giochi basati sulla stessa architettura di Catching Cherries. Essa dovrà contenere una classe interna che non è altro che il Timer descritto nel paragrafo 4.7.3.

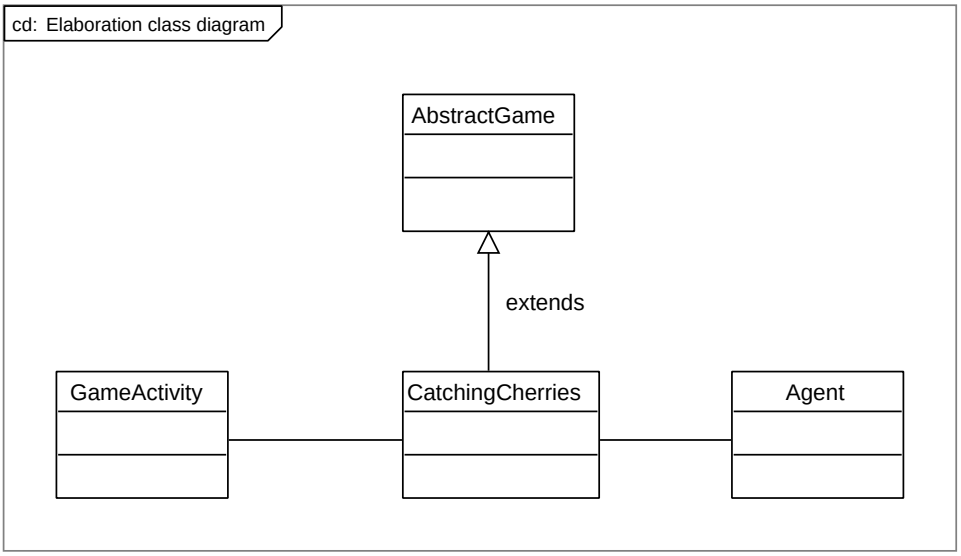


Figura 5.3: Diagramma delle classi prodotto durante la fase di Elaboration

La classe CatchingCherries sarà il vero e proprio nucleo del gioco: oltre a offrire le funzionalità di layout manager descritte nel paragrafo 4.7.3, si occuperà di interpretare i diversi input ricevuti dall'utente. La classe Agent modella gli stati ed i comportamenti dei diversi tipi di agente.

5.3.4 Diagramma di sequenza

A questo punto attraverso il diagramma UML in figura 5.4 è possibile analizzare la sequenza in cui dovranno verificarsi alcune operazioni chiave del gioco. Nel momento in cui l'utente avvia l'applicazione, devono essere inizializzate tutte le risorse necessarie al gioco e deve essere generata l'interfaccia grafica. Soltanto a questo punto l'applicazione sarà pronta a ricevere comandi.

Non appena l'utente effettuerà la pressione del tasto "Start Game", l'Activity del gioco dovrà richiamare il layout manager affinché provveda ad avviare la partita, che dovrà proseguire fino al game over. L'utente dovrà essere

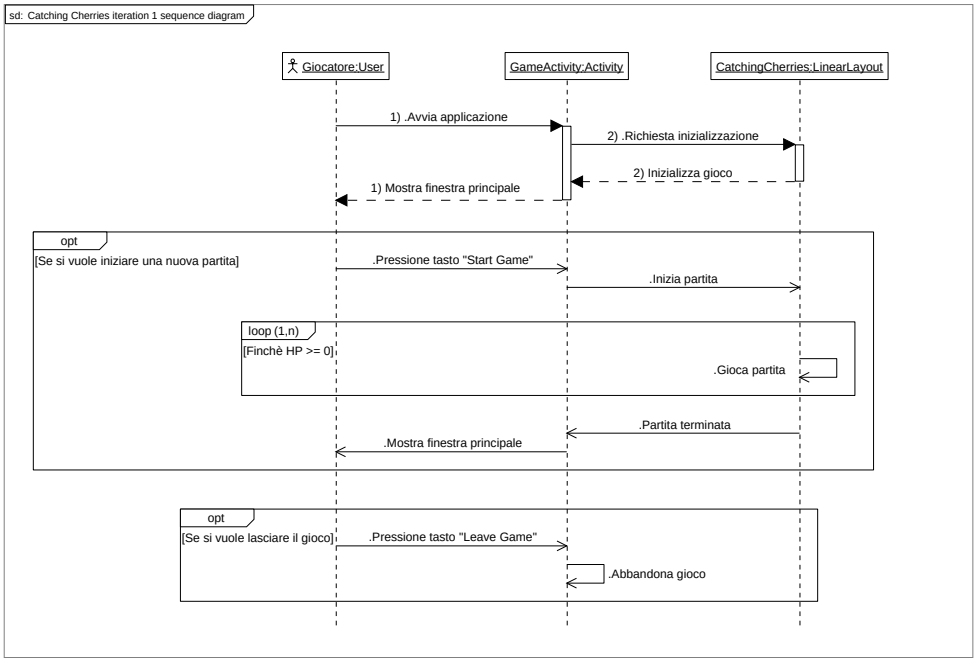


Figura 5.4: Diagramma di sequenza prodotto durante la fase di Elaboration

riportato alla schermata iniziale del gioco da cui potrà scegliere se giocare di nuovo o chiudere l'applicazione con la pressione del tasto "Leave Game".

5.3.5 Diagramma di stato

Il diagramma in figura 5.5, è la macchina a stati finiti relativa all'applicazione. Per il momento gli stati attraverso i quali l'applicazione potrà passare sono tre:

- (T) applicazione terminata;
- (A) applicazione inizializzata ma in attesa;
- (G) applicazione in modalità gioco.

Quando il programma viene lanciato, esso passa dallo stato (T) allo stato (A). Se viene avviata una nuova partita, ci sarà una transizione dallo stato

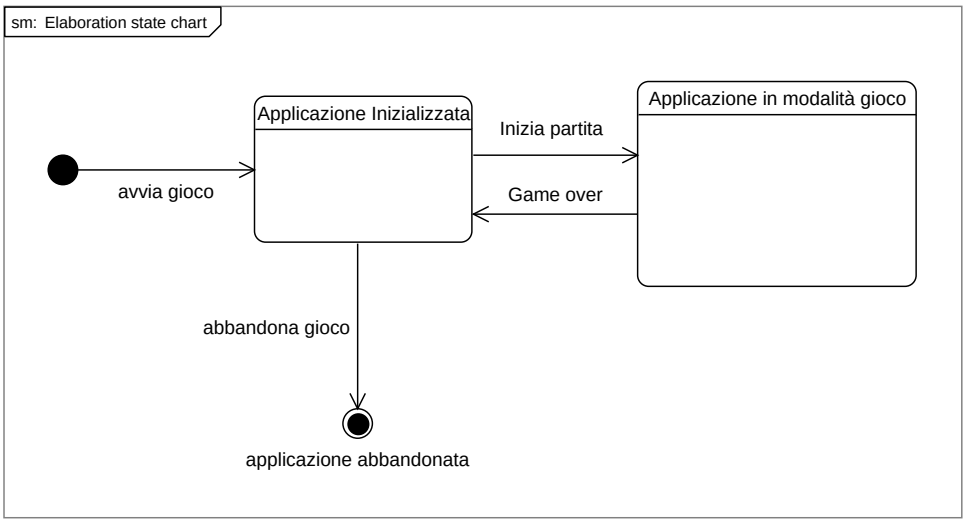


Figura 5.5: Diagramma degli stati prodotto durante la fase di Elaboration

(A) allo stato (G), di conseguenza da quel momento dovrà essere innescata la procedura vista nel paragrafo 4.7.2 che non dovrà mai essere eseguita al di fuori[2] della modalità gioco. Non appena la partita sarà terminata, l'applicazione dovrà passare di nuovo allo stato (A).

Si tenga conto che questo è un diagramma che rappresenta un determinato punto di vista che non considera il ciclo di vita delle Activity, il quale verrà analizzato nella Fase Construction.

5.3.6 Il main loop del gioco

Il main loop regola l'esecuzione della partita, che è il cuore del gioco, esso dovrà continuare ad essere iterato fino al game over.

[2]Eseguire la procedura in questione ha senso solo se sono richieste animazioni, nel caso di schermate statiche la sua esecuzione sarebbe superflua

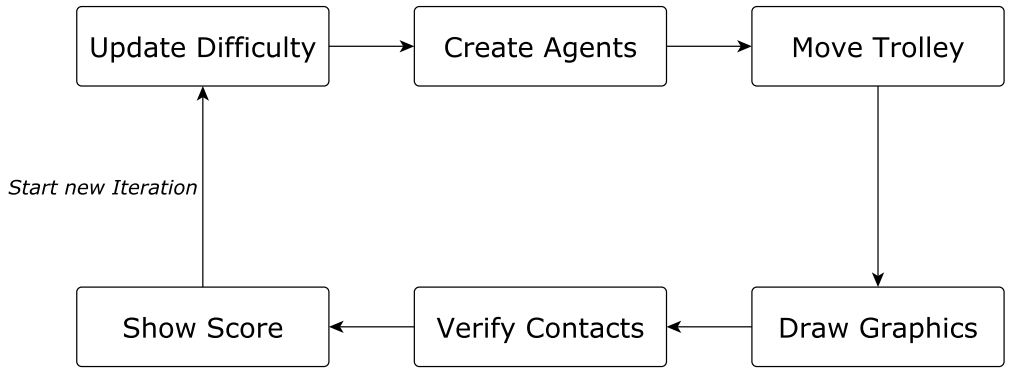


Figura 5.6: Il main loop di Catching Cherries

La decisione di fare uso di un meccanismo come il main loop deriva dalla natura stessa del gioco. Esso è stato introdotto poiché il gioco deve avere le seguenti caratteristiche:

- (A) Elementi grafici che cambiano continuamente anche in assenza di input.
- (B) Non attendere input dal giocatore prima di aggiornare lo stato della partita.

In realtà (B) implica (A), ma essa è stata esplicitata per rendere più chiara la situazione.

Il main loop è costituito dalle seguenti operazioni che dovranno seguire l'ordine di esecuzione indicato:

- (1) - (Update Difficulty) Prima di tutto deve essere aggiornata la difficoltà del gioco, in questo modo la velocità e la quantità di agenti verranno regolate in tempo utile alla loro visualizzazione.

- (1) - (Create Agents) Qui dovrà essere implementato un sistema che gestisca la generazione di nuovi agenti in base alla difficoltà del gioco.
- (3) - (Move Trolley) Viene interpretato l'input del giocatore.
- (4) - (Draw Graphics) Soltanto a questo punto si hanno a disposizione tutti i dati necessari a poter disegnare il carrello e gli agenti.
- (5) - (Show Score) Vengono aggiornati il punteggio e gli HP del giocatore per essere mostrati nella schermata di gioco.
- (6) - (Verify Contacts) Il sistema controlla se sono avvenute collisioni tra il carrello e gli agenti.

Come si può notare la scelta progettuale di adottare il main loop permette di dividere il problema principale "partita" in una serie di sotto-problemi più semplici. I sotto-problemi corrispondono alle varie fasi della partita che potranno così essere implementate indipendentemente l'una dall'altra.

5.4 Implementazione architettura di base

Uno degli obiettivi principali di questa fase di Elaboration è realizzare un prototipo funzionante basato sull'architettura definita nella fase di Inception ed evoluta nella fase corrente. In questo caso la parola prototipo non indica un prodotto usa e getta, ma un software che in futuro sarà parte dell'applicazione finale.

Anche se il prototipo rappresenta una versione primordiale del gioco, che implementa soltanto parte delle funzionalità introdotte con i requisiti di questa fase di Elaboration, la sua importanza non è assolutamente da sottovalutare. Esso costituisce lo scheletro dell'applicazione, da cui dipende la buona riuscita dell'intero progetto.

È stata data la precedenza alle funzionalità più critiche, tralasciando per il momento quelle più superficiali in modo da ottenere il prototipo nel minor tempo possibile.

È stato stabilito che il prototipo includerà le seguenti funzionalità basilari:

- supporto touch screen;
- supporto display con risoluzione HVGA[3];
- avvio della partita e chiusura dell'applicazione;
- generazione random degli agenti;
- livello di difficoltà stabile;
- movimento del carrello;
- tre agenti diversi, uno per categoria (vedi paragrafo 5.2);
- basilare gestione contatti tra agenti e carrello;
- calcolo e visualizzazione punteggio e HP;
- possibilità di game over.

5.4.1 Scelte implementative

Fra tutte le operazioni coinvolte nel progetto la scrittura del codice è una tra le più costose, se non la più costosa in assoluto.

Nel momento in cui il progetto arriva in una fase abbastanza adulta, le modifiche divengono estremamente onerose e quindi non è conveniente cambiare la direzione intrapresa in principio, anche se si scopre che tale direzione non era la migliore.

Nella prima fase di implementazione dell'applicazione si sono dovute effettuare delle scelte che avrebbero condizionato pesantemente il suo futuro. Per questo motivo il prototipo è stato dotato soltanto delle funzionalità indispensabili a verificare il suo funzionamento, ed è stato ovviamente predisposto per successive espansioni.

Considerando lo studio di fattibilità (paragrafo 4.3) ed i requisiti tecnici (paragrafo 4.6) relativi alla fase di Inception, durante la realizzazione del

[3]HVGA = 320 x 480 pixel.

prototipo sono state effettuate numerose scelte implementative, di cui sono state riportate le più cruciali.

Se si vuole sviluppare applicazioni, ed in particolare giochi, di qualità, per Android, si deve essere coscienti fin dall'inizio che i vincoli (vedi paragrafi 1.2.2, 2.4.1, 4.3) posti dal contesto in cui si opera richiedono un costante scendere a compromessi.

Per questo progetto, si è stati spesso obbligati a scegliere tra flessibilità e prestazioni. Produrre codice flessibile significa renderlo manutenibile, quindi facilmente riutilizzabile e predisposto a future modifiche. Ma il prezzo da pagare per una tale resilienza è la scarsa efficienza.

Un codice prestante invece è snello, efficiente, e ottimizzato per una causa specifica. Talmente specializzato che spesso non è utilizzabile al di fuori dal contesto in cui è radicato e tentativi di modifiche hanno alta probabilità di immissione di bug.

Riuscire a scrivere software che sia contemporaneamente "veloce" ed adattabile è indubbiamente un'ardua impresa e talvolta non è possibile ottenere un risultato del genere.

Si è cercato di favorire maggiormente la flessibilità ma molto spesso si è dovuta affrontare una dura realtà: il fine ultimo di un gioco non è certo quello di essere un'enciclopedia di "best practices"[4] della programmazione in versione "slideshow"[5]. Avvolte è necessario infrangere le regole di buona programmazione per non rendere l'intero progetto un'attività fine a se stessa. Il che significa che in alcuni casi è necessario mettere in secondo piano alcuni aspetti anche nobili per non perdere di vista gli obiettivi primari.

In alcune parti del gioco si è dovuto sacrificare maggiormente la flessibilità, soprattutto all'interno del main loop.

In linea di massima si è cercato di ottimizzare le operazioni che vengono eseguite con maggiore frequenza, quindi quasi esclusivamente quelle relative

[4]Per migliore pratica o migliore prassi si intendono in genere le esperienze più significative, o comunque quelle che hanno permesso di ottenere migliori risultati, relativamente a svariati contesti. [54]

[5]Con il termine slideshow (dall'inglese slide, diapositiva e show, presentazione) si definisce una sequenza di immagini. [54]. A questa parola è stato dato un doppio senso che si riferisce ad un gioco nel quale è possibile distinguere tutti i singoli fotogrammi

al main loop, che in alcuni casi vengono effettuate centinaia di volte al secondo. La maggior parte delle accortezze che si sono avute non sarebbero state così rilevanti se il software fosse stato di diversa natura (ad esempio un programma per PC).

Alcuni principi che sono stati seguiti sono descritti di seguito:

- Allocare meno memoria possibile.

  Questa restrizione non è dovuta tanto all'allocazione di memoria in se quanto alle conseguenze che essa provoca.

  Ogni volta che viene allocata della memoria, si da la possibilità al Garbage Collector (GB)[6] di intervenire, provocando al gioco degli evidenti rallentamenti. Interruzioni nelle animazioni della durata di piccole frazioni di secondo possono essere sufficienti a compromettere l'esperienza di gioco.

  Un esempio di ottimizzazione che è stata effettuata è la non adozione di strutture dati che ereditano da Collection: Arraylist, Map, ecc...

  Sono stati usati gli array, molto più efficienti in termini di memoria allocata.

- Ridurre al minimo le invocazioni di metodi non statici, in quanto più costose in termini di tempo.

- Non utilizzare metodi getter e setter, almeno per quanto riguarda le parti di codice che vengono eseguite più spesso.

- Cercare di usare i modificatori *final static* per le variabili, dato che conferiscono ad esse tempi di accesso più brevi.

Per lo sviluppo dell'applicazione è stato usato l'ambiente di sviluppo Eclipse (vedi 2.5.2) integrandolo con gli ADT [4], che permettono di utilizzare tramite tale IDE tutti gli strumenti della SDK (vedi 2.5.1) di Android.

[6]Per garbage collection si intende una modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, liberano le porzioni di memoria che non dovranno più essere successivamente utilizzate dalle applicazioni. [54]

5.4.2 Porzioni di codice del prototipo

Per brevità, in questa parte sono state riportate soltanto le parti del sorgente del prototipo ritenute più rilevanti.

Classe AbstractGame

Questa classe astratta fornisce delle funzionalità generiche che possono risultare utili all'implementazione di una certa tipologia di giochi. Di questa classe verrano descritti i seguenti componenti:

- metodo onLayout, listato 5.1;
- metodo startTimer, listato 5.2;
- classe interna ViewRefresher, listato 5.3.

Metodo onLayout Viene invocato ogni qualvolta è necessario assegnare posizione e dimensioni alla vista principale dell'applicazione, in questo caso questo succede appena viene lanciato il gioco. Quando viene lanciato esso richiama prima il metodo initialize, che viene implementato nella classa figlia CatchingCherries e poi il metodo startTimer.

```
/**
 * Metodo che viene eseguito ogni volta che viene creato
 * il layout dell'applicazione.
 */
@Override
protected void onLayout(boolean changed, int left,int top,
   int right, int bottom)
{
   super.onLayout(changed, left, top, right, bottom);
   try {

      /*
       * Se le dimensioni di questa View sono cambiate,
       * si procede a inizializzare il gioco ed ad
       * avviare il timer.
       */
```

```
        if (changed) {

            initialize();

            startTimer();
        }
            }
    catch (Exception e) {
    e.printStackTrace();
    }
}
```

Listato 5.1: Metodo onLayout

Metodo startTimer Questo metodo si occupa di istanziare e pianificare l'esecuzione del timer.

Il primo parametro del metodo schedule indica il timer da pianificare, il secondo parametro indica il tempo che deve trascorrere prima dell'esecuzione iniziale del timer, il terzo parametro è il periodo, cioè l'intervallo di tempo che deve trascorrere tra due esecuzioni consecutive del timer.

```
/**
 * Metodo che inizializza e pianifica il timer del gioco.
 */
private void startTimer() {

    loopTimer = new Timer();

    /*
     * Viene pianificata l'esecuzione del timer,
     * il parametro refreshDelay indica il periodo,
     * inteso come inverso della frequenza.
     */

    loopTimer.schedule(new ViewRefresher(), 0,
        refreshDelay);
}
```

Listato 5.2: Metodo startTimer

Classe interna ViewRefresher

Questa classe modella il timer che viene pianificato nel metodo startTimer. Questo è possibile grazie al fatto che essa estende TimerTask, che a sua volta implementa l'interfaccia Runnable, di conseguenza gli oggetti ViewRefresher sono a tutti gli effetti dei Thread. Dal momento in cui viene pianificato, un timer di tipo ViewRefresher non fa altro che invalidare la schermata di gioco ad intervalli regolari, questo per far sì che essa venga ridisegnata in una versione più aggiornata.

```
/**
* Timer java usato per invalidare la schermata visualizzata
* in modo da innescarne il ridisegno forzato. L'obiettivo è
* di pianificare l'esecuzione del timer in modo da ottenere
* un'animazione.
*
* @author Daniele D'Adamo
*/
private class ViewRefresher extends TimerTask {

   /**
    * Il run di questo thread rappresenta l'operazione
    * che deve essere eseguita ad ogni pianificazione
    * del timer.
    */
   @Override
   public void run() {

      /*
       * La View viene invalidata in modo da forzare la
       * chiamata del metodo onDraw() da parte del thread
       * responsabile dell'interfaccia grafica.
       * Non è necessario invalidare la View se
       * l'applicazione non è in fase di gioco ma
       * in una schermata priva di animazioni.
       */
         if (playing) {
            postInvalidate();
         }
```

```
    }
}
```

Listato 5.3: Classe ViewRefresher

Classe GameActivity

Questa classe è la Activity dell'applicazione, in quanto tale, si occupa di associare il layout appropriato all'applicazione, in questo caso si tratterà di un layout personalizzato. Inoltre si occupa di gestire la pressione del pulsante "Start Game", inviando alla classe CatchingCherries la richiesta di iniziare la partita e del pulsante "Leave Game", provvedendo ad effettuare la chiusura dell'applicazione.

Di questa classe verrà analizzato il metodo onCreate (listato 5.4).

Metodo onCreate Nel momento in cui il sistema crea un'istanza dell'applicazione, il primo metodo che viene richiamato è onCreate. Esso imposta il layout dell'applicazione, attiva la rilevazione di eventi touch e associa a degli oggetti gli elementi grafici definiti nel layout XML.

```
/**
 * Metodo che viene eseguito al momento della creazione
 * della Activity, cioè all'avvio del gioco.
 * Questo metodo carica il layout personalizzato,
 * definito nel file XML.
 */
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    /*
     * Viene usato un LayoutInflater per definire
     * il layout personalizzato.
     */
    LayoutInflater inflater = LayoutInflater.from(this);

    // Viene creata una reference al layout dell'applicazione.
```

```
    mainView = inflater.inflate(R.layout.main, null);

    setContentView(mainView);

    /* Queste due operazioni sono fondamentali per permettere
     * all'applicazione di ricevere eventi touch.
     */
    mainView.setFocusable(true);
    mainView.setFocusableInTouchMode(true);

    /* Vengono create reference ad oggetti definiti nel layout
     * XML, in modo da poter effettuare su essi
     * delle operazioni.
    /*
    label1 = (TextView)findViewById(R.id.label1);
    layout1 = (LinearLayout)findViewById(R.id.layout1);
    startPauseButton =
       (Button) findViewById(R.id.primaryButton);
    leaveGameButton =
       (Button) findViewById(R.id.secondButton);
}
```

Listato 5.4: Metodo onCreate

Layout XML

È stato riportato il file XML che definisce il layout personalizzato (listato 5.5). In questo file vengono designati alcuni elementi dell'interfaccia grafica, come i tre pulsanti, un area di testo e due sotto-layout. Inoltre viene impostato il colore di sfondo dell'applicazione.

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

<!--Layout Definito dall'Utente   -->
<it.unibo.is.dadamo.thesis.main.CatchingCherries

    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ll_absolute"
     android:orientation="vertical"
     android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
android:background="#CCFF99">

  <LinearLayout
  android:id="@+id/layout1"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content">

     <Button
     android:id = "@+id/secondButton"
     android:layout_width = "wrap_content"
     android:layout_height="wrap_content"
     android:text="Leave Game"
     android:onClick="SecondButtonPressed"/>

     <Button
     android:id = "@+id/primaryButton"
     android:layout_width = "wrap_content"
     android:layout_height="wrap_content"
     android:layout_weight="7"
     android:text="Start game"
     android:onClick="PrimaryButtonPressed"/>

     <Button
     android:id = "@+id/thirdButton"
     android:layout_width = "wrap_content"
     android:layout_height="wrap_content"
     android:text="Button Three"/>

  </LinearLayout>

  <LinearLayout
  android:id="@+id/layout2"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:gravity="center">
```

```
    <TextView
    android:id = "@+id/label1"
    android:layout_width = "fill_parent"
    android:layout_height="wrap_content"
    android:text="welcome to Catching Cherries!!"
    android:textSize="30sp"
    android:textColor="#000000"/>

    </LinearLayout>

</it.unibo.is.dadamo.thesis.main.CatchingCherries >
```

Listato 5.5: Layout XML

Classe CatchingCherries

Classe principale del gioco, estende AbstractGame e di conseguenza LinearLayout, gestisce tutta la logica del gioco.

Si occupa di inizializzare le risorse necessarie al gioco, rilevare gli input del giocatore e soprattutto gestire l'esecuzione della partita.

Verrano descritti i metodi onTouchEvent (listato 5.6) e onDraw (listato 5.7), gli altri componenti principali verranno analizzati nel capitolo 6.

Metodo onTouchEvent Questo metodo gestisce le azioni eseguite dal giocatore sul touch-screen. La sua funzione è quella di individuare le coordinate del punto toccato dal giocatore e stabilire se è stato toccato il carrello, in caso positivo il carrello verrà spostato. Il carrello risulterà toccato soltanto se entrambe le coordinate del punto toccato corrispondono a coordinate appartenenti all'area del carrello.

```
/**
 * Metodo che viene eseguito quando il sistema rileva un
 * evento corrispondendo al tocco dello schermo.
 */
   @Override
   public boolean onTouchEvent(MotionEvent event) {

      /*
```

```
         * Vengono create delle variabili temporanee per
         * memorizzare le coordinate corrispondenti al tocco.
         */
        int horizontalTouchCoordinate = (int)event.getX();
        int verticalTouchCoordinate = (int)event.getY();

        /*
         * Attraverso queste condizioni si verifica se è
         * stato toccato un punto all'interno della sagoma
         * del carrello.
         */
        if ( horizontalTouchCoordinate >= trolleyXCoordinate
           && horizontalTouchCoordinate <= trolleyXCoordinate
           + trolleyWidth
           && verticalTouchCoordinate >= trolleyYCoordinate
           && verticalTouchCoordinate <= trolleyYCoordinate
           + trolleyHeight
        ){

           /*
            * In caso positivo viene memorizzata la nuova
            * posizione in cui dovrà essere disegnato
            * il carrello.
            */
           trolleyXCoordinate = horizontalTouchCoordinate
              - (trolleyWidth/2);

        }
     return true;
}
```

Listato 5.6: Metodo onTouchEvent

Metodo onDraw Questo metodo implementa la procedura descritta nel paragrafo 5.3.6, cioè il main loop del gioco. Ogni esecuzione di questo metodo corrisponde ad un singolo aggiornamento dell'intero stato del gioco e di conseguenza ad un singolo fotogramma (frame) dell'animazione.

```
/**
```

```
 * Metodo che implementa il main loop dell'applicazione.
 */
@Override
protected void onDraw(Canvas canvas) {
super.dispatchDraw(canvas);

   /*
    * Si procede solo se l'applicazione è in modalità
    * gioco.
    */
   if (playing) {

      /*
       * Quando verrà implementato, questo metodo gestirà
       * l'incremento di difficoltà del gioco.
       */
      //updateDifficulty();

      /*
       * Qui vengono generati gli agenti e viene
       * estratta la loro posizione orizzontale.
       */
      createAgent();

      // Viene aggiornata la posizione del carrello.
      moveTrolley();

      // Vengono disegnate tutte le grafiche.
      drawGraphics(canvas);

      /*
       * Si individuano eventuali contatti tra
      * carrello e agenti.
      */
      verifyContacts();

      /*
       * Vengono aggiornati e disegnati
       * il punteggio e gli HP.
```

```
            */
            showScore(canvas);

        }

    }
```

Listato 5.7: Metodo onDraw

5.5 Testing e debugging

A questo punto è stato verificato il funzionamento del prototipo realizzato.

Il testing è stato effettuato tramite lo AVD Manager (vedi 2.5.1) fornito con la SDK ed il prezioso DDMS (vedi 2.5.1), fornito nel medesimo pacchetto ed integrato in Eclipse, che già fornisce un'ottimo tool di debugging.

Le operazioni di emulazione sono particolarmente costose e questo risulta in scarse prestazioni complessive dello AVD.

Ciò significa che in questa fase la varietà di operazioni di testing è limitata, in quanto alcuni dei risultati osservati, soprattutto quelli relativi alle prestazioni del gioco, sono funzione della potenza di calcolo della macchina che ospita l'emulatore.

Tutti i dati relativi alle prestazioni non sono da considerarsi attendibili ma in questo caso sono particolarmente utili in quanto rappresentano lo scenario di esecuzione peggiore (worst-case-scenario).

Infatti l'obiettivo di un buon testing è di verificare il funzionamento del programma soprattutto nei casi limite, meno prevedibili, non soltanto negli scenari di esecuzione migliori.

Il DDMS è munito di efficaci strumenti per il debugging, quelli che sono stati utilizzati maggiormente sono:

- LogCat

  È lo strumento che permette di osservare tutte le operazioni che avvengono all'interno del dispositivo, tramite la generazione di log suddivisi per livello di priorità.

Esso è risultato utile per sondare continuamente lo stato della partita.

Il comando:

```
Log.d("Debug creazione agenti ", "agente estratto");
```

ad esempio crea un rapporto visualizzabile in tempo reale dal quale è possibile verificare se l'estrazione di un certo agente è stata portata a compimento, il primo parametro indica il contesto del rapporto, il secondo è il messaggio.

- Allocation Tracker

  Permette di tenere traccia di tutte le allocazioni di memoria che vengono effettuate in un determinato intervallo di tempo. Esso è utile quindi per scoprire se avvengono delle allocazioni non pianificate, annidate all'interno di operazioni di cui non si conoscono tutti i minimi dettagli.

  Ad esempio l'invocazione del metodo:

  ```
  Arrays.sort(myArray);
  ```

  comporta la duplicazione temporanea dell'array e di conseguenza un' allocazione di memoria.

La funzione debug di Eclipse, arricchita dal pacchetto ADT, permette di scandire l'esecuzione del programma in singole esecuzioni di istruzioni. In questo modo è possibile comprendere la dinamica di tutte le operazioni che compongono il software.

Screenshots

In questa parte sono riportati alcuni screenshots del prototipo in esecuzione, per fornire un idea del suo aspetto grafico.

La figura 5.7 mostra la schermata principale dell'applicazione. Il pulsante "Button Three" non è al momento legato ad alcuna funzionalità.

La figura 5.8 mostra lo stato iniziale della partita.

La figura 5.9 mostra l'applicazione dopo che è avvenuto il game over.

Figura 5.7: Schermata principale dell'applicazione.

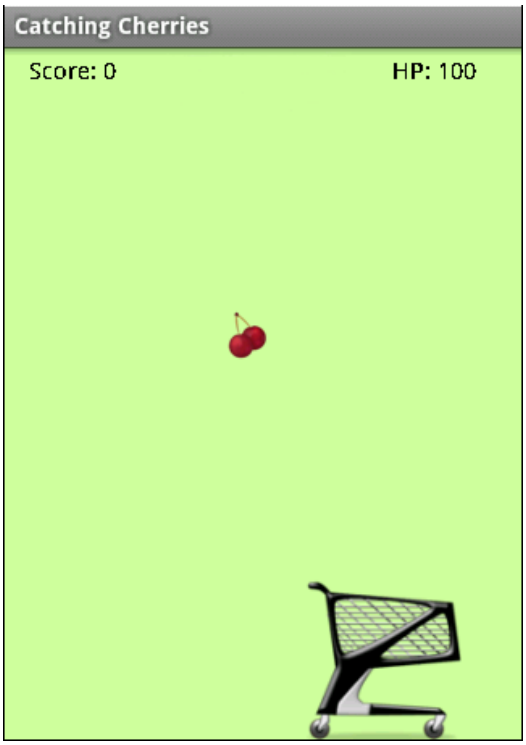


Figura 5.8: Partita appena iniziata.

Figura 5.9: Schermata di game over.

Capitolo 6

Construction Phase

L'obiettivo della fase di Construction è di ottenere un prodotto finito attraverso successivi raffinamenti, la maggior parte dell'implementazione del gioco è avvenuta quindi in questa fase.

Il prototipo proveniente dalla fase di Elaboration è stato utilizzato come nucleo intorno al quale costruire il resto dell'applicazione.

Questa fase è stata affrontata attraverso numerose iterazioni, ognuna corrispondente ad un ciclo di vita dell'applicazione: ogni iterazione di Construction ha contribuito a perfezionare in maniera progressiva il prodotto. Per ogni iterazione c'è stata prima di tutto la scelta degli obiettivi di quel determinato ciclo di vita, seguita dalla ricerca e dalla messa in atto delle strategie per ottenere tali obiettivi. L'iterazione viene conclusa con la valutazione dei risultati ottenuti.

Nel paragrafo 6.1 è riportata la lista dei requisiti introdotti in questa fase, il paragrafo 6.2 contiene il risultato di tutte le iterazioni di progettazione. Nel paragrafo 6.3 viene illustrata una parte del codice sorgente del gioco ed infine nel paragrafo 6.4 vengono descritte le modalità in cui è avvenuto il testing al termine di ogni iterazione.

6.1 Metodologia di sviluppo e requisiti aggiuntivi

Nelle prime iterazioni di Construction sono state implementate tutte le funzionalità previste dai requisiti della fase di Elaboration. Ad ogni iterazione è stato concretizzato un basso numero di requisiti, in questo modo è stato possibile concentrarsi sulla risoluzione di ogni sotto problema in maniera separata, senza fare passi in avanti troppo lunghi.

Dopodiché sono state aggiunte nuove funzionalità da implementare nelle successive iterazioni di Construction, seguendo la dinamica appena descritta.

Alla fine di ogni iterazione, se i requisiti (dell'iterazione in questione) risultavano soddisfatti, si passava all'iterazione successiva, aggiungendo altri requisiti sulla base del feedback ottenuto durante la fase di testing.

Di seguito sono riportati tutti i requisiti che sono stati aggiunti durante la fase di Construction.

- Dovranno essere gestiti in maniera migliore i casi in cui la Activity dell'applicazione passi per gli stati "Paused" e "Stopped".
- Dovrà esserci la possibilità di mettere il gioco in pausa. Nel momento in cui l'applicazione viene messa in pausa dovrà apparire una finestra di dialogo.
- La finestra di dialogo dovrà permette al giocatore di riprendere il gioco, abbandonare la partita o avviare una nuova partita.
- Il gioco dovrà prevedere la gestione dei punteggi nelle seguenti modalità:
  - I punteggi più alti dovranno essere memorizzati in un file.
  - Nel momento in cui il giocatore ottiene un punteggio tra quelli più alti, dovrà essergli data la possibilità di inserire il proprio nome, che dovrà essere registrato nel file insieme al suo punteggio.
  - Dovrà esserci una lista dei punteggi più alti, ottenuta dal file, visualizzabile dall'utente attraverso una apposita schermata.

- Gli agenti dovranno cadere con una certa accelerazione.
- Dovranno essere gestiti gli input corrispondenti alla pressione dei tasti: HOME, MENU, BACK.
- Dovrà essere modificato l'effetto prodotto dal contatto tra il carrello e gli agenti che rappresentano junk food. Oltre a sottrarre HP essi dovranno sottrarre al giocatore anche una certa quantità di punti.
- Dovrà essere aggiunta un'ulteriore categoria di agenti: agente "poison", il contatto con quest'ultimo dovrà determinare una notevole riduzione degli HP.
- Dovranno essere aggiunti altri tipi di frutta alla categoria di agenti che conferiscono punteggio.
- Dovrà essere generata una vibrazione nel momento in cui il giocatore ottiene un punteggio elevato e nei casi in cui il carrello entra in contatto con gli agenti a lui ostili.

6.2 Progettazione

In questo paragrafo sono stati riportati i risultati finali della progettazione.

Ad ogni iterazione di Construction sono state effettuate scelte progettuali di piccola entità, in modo da non appesantire eccessivamente la corrispondente fase implementativa.

6.2.1 Diagramma dei casi d'uso

Il diagramma in figura 6.1 rappresenta i casi d'uso relativi alla partita ed è l'ultima delle ampliazioni del diagramma 5.3.1.

La principale novità riguarda l'aggiunta della possibilità di mettere il gioco in pausa.

Nel momento in cui il giocatore mette il gioco in pausa, il sistema dovrà dargli la possibilità di scegliere tra tre possibilità:

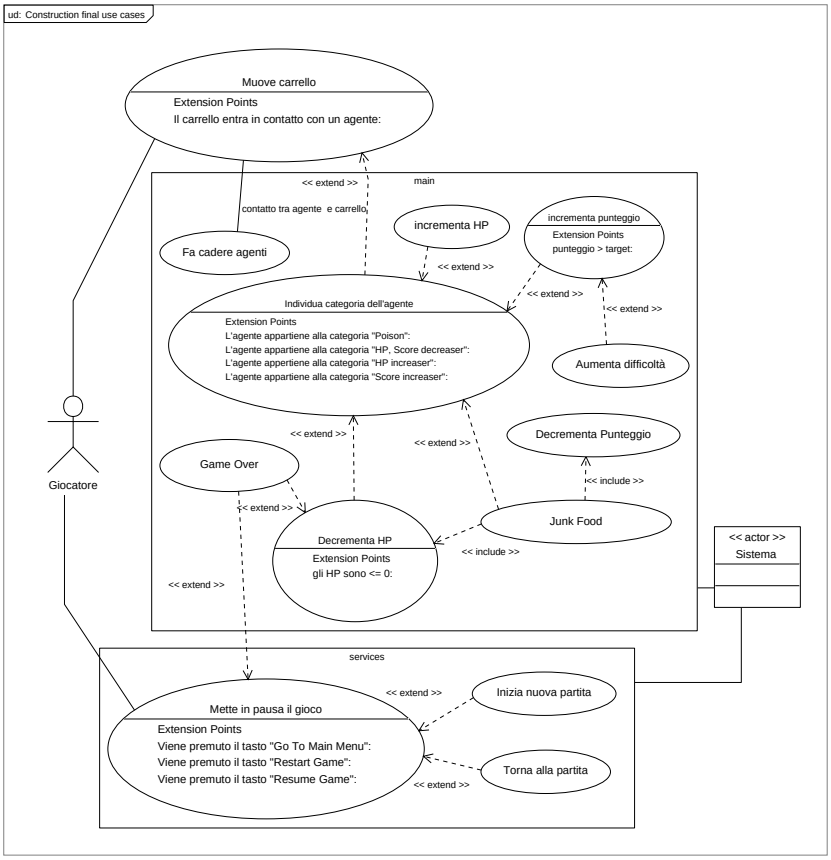


Figura 6.1: Diagramma dei casi d'uso relativi alla partita, riferito all'applicazione nel suo stadio finale di sviluppo

- Lasciare la partita e tornare alla schermata iniziale del gioco. In questo caso verrà lanciato il game over.

- Iniziare una nuova partita annullando quella corrente.

- Riprendere la partita corrente.

Un'altra estensione dei casi d'uso consiste nell'aggiunta di un nuovo tipo di agente ed in una leggera modifica della categoria "junk food".

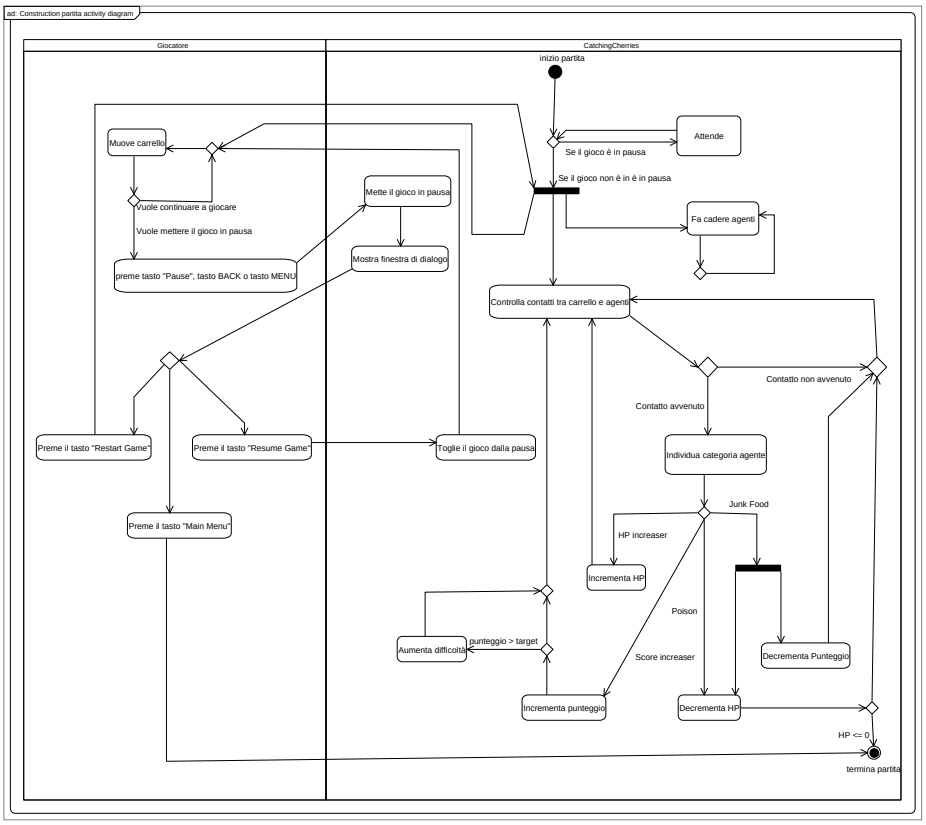


Figura 6.2: Diagramma di attività relativo alla partita, riferito all'applicazione nel suo stadio finale di sviluppo

6.2.2 Diagramma di attività

Il diagramma di attività in figura 6.2 è un'estensione del diagramma 5.2, relativa allo stadio finale dello sviluppo.

Tutti i casi d'uso del diagramma 6.1, qui vengono descritti da un angolazione orientata al flusso di esecuzione della partita. In particolare viene messa in evidenzia l'interazione persona-sistema.

6.2.3 Diagramma delle classi

Il diagramma in figura 6.3 modella la struttura finale dell'applicazione, consiste nella revisione finale del diagramma 5.3. La struttura è stata divisa in

due pacchetti:

- package *main*, che contiene tutte le classi specifiche del gioco Catching Cherries.
- package *services*, che contiene le classi che forniscono servizi ausiliari, sfruttabili da una certa tipologia di giochi.

Le classi del package *main* e la classe AbstractGame sono già state descritte nel paragrafo 5.3.3, le classi del package *services* sono descritte di seguito.

La classe ScoresManager offre funzionalità per la gestione dei punteggi, la classe AudioTrack incapsula funzionalità per l'utilizzo di traccie audio e la classe PauseManager offre un sistema per la creazione di una finestra di dialogo per gestire la pausa.

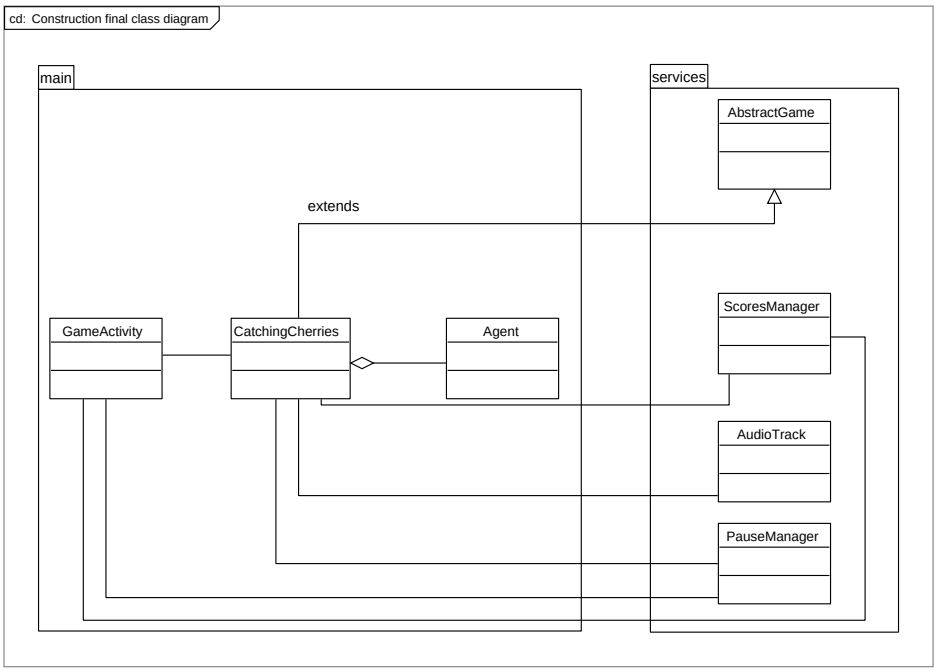


Figura 6.3: Diagramma delle classi riferito all'applicazione nel suo stadio finale di sviluppo

6.2.4 Diagramma di sequenza

Il diagramma di sequenza in figura 6.4 è l'ultima estensione del diagramma 5.4. Esso mostra la sequenza di esecuzione della partita, in maniera più approfondita rispetto al diagramma in figura 6.2. Mette in luce la responsabilità delle varie classi e quindi aumenta il livello di dettaglio della modellazione.

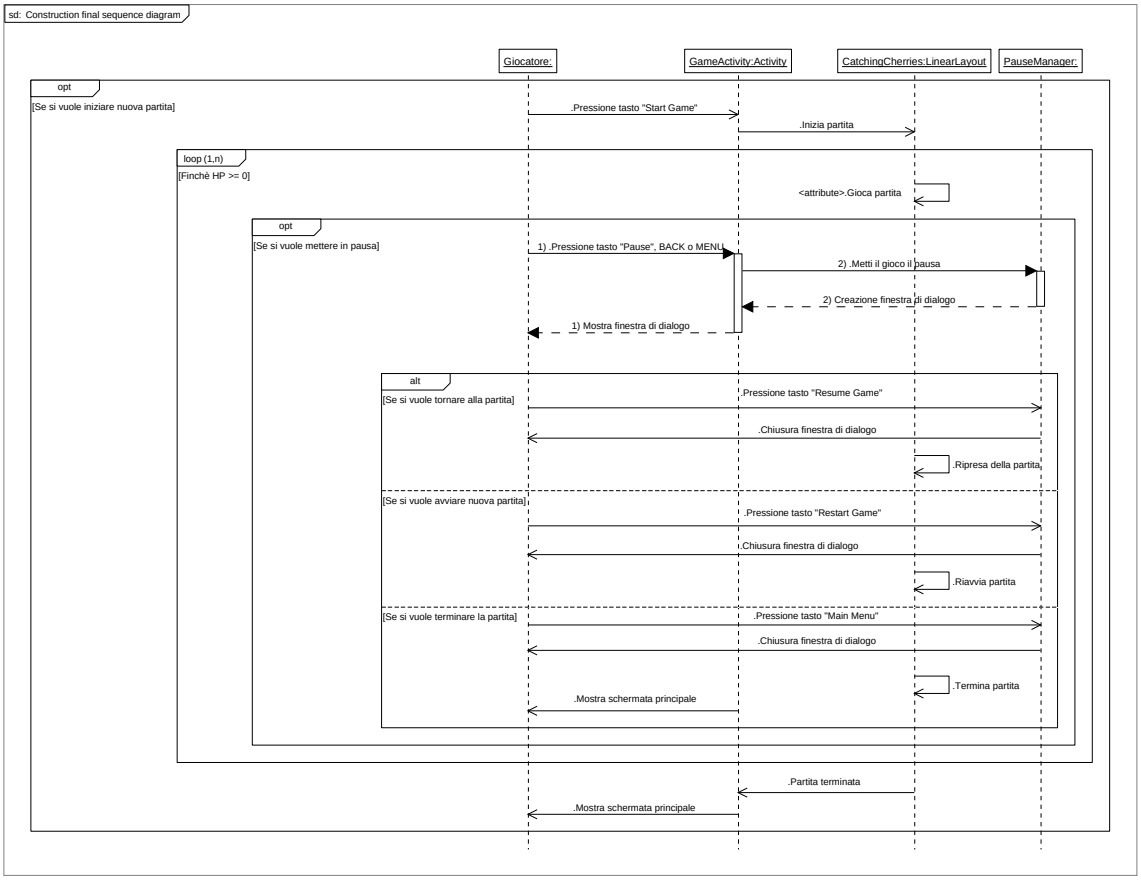


Figura 6.4: Diagramma di sequenza riferito all'applicazione nel suo stadio finale di sviluppo

6.2.5 Diagramma di stato

Il diagramma in figura 6.5 è l'estensione del diagramma 5.5, aggiunge due nuovi stati al gioco:

- (P) Gioco in pausa.
- (S) Applicazione in modalità visualizzazione punteggi.

Nel momento in cui il gioco viene messo in pausa, esso passa dallo stato (G) allo stato (P), di conseguenza la partita deve essere sospesa e il timer deve smettere di invalidare la vista. Se dallo stato (P) il giocatore torna alla partita o ne avvia una nuova, non avverrà alcuna transizione di stato e il timer riprenderà a invalidare la vista. Se invece il giocatore abbandona la partita, ci sarà una transizione allo stato (A) (gioco in attesa) e di conseguenza la schermata non dovrà subire invalidazioni artificiali fino a che la partita non viene avviata di nuovo.

Dalla finestra principale del gioco è ora possibile accedere alla lista dei punteggi più alti, causando una transizione allo stato (A) allo stato (S). Il gioco dovrà interpretare gli input in maniera diversa a seconda dello stato in cui si trova. Ad esempio la pressione del tasto BACK durante la partita farà apparire la finestra della pausa, mentre se lo stesso tasto viene premuto nella schermata principale, verrà abbandonato il gioco.

6.2.6 Diagramma dei componenti

Il diagramma in figura 6.6 mostra l'applicazione dal punto di vista dei componenti che la formano.

Uno dei vantaggi di tale suddivisione è la possibilità di rendere modulare il software e favorire la riusabilità dei componenti.

I componenti del gioco sono:

- Catching Cherries Core.
- Audio Component.
- Game common services.

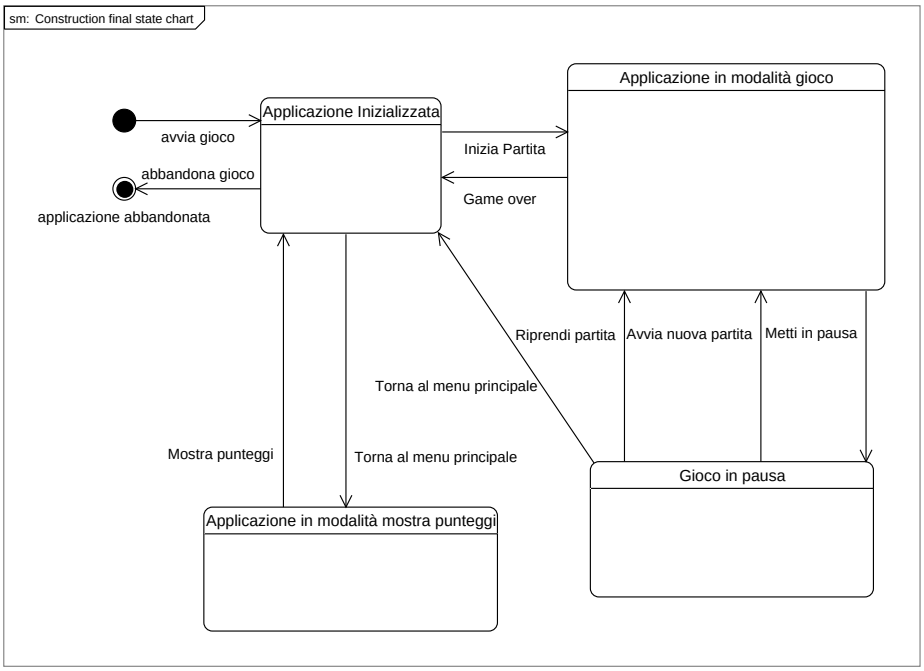


Figura 6.5: Diagramma di stato riferito all'applicazione nel suo stadio finale di sviluppo

- Pause Manager Component.
- Score Manager Component.

Ognuno di essi è specializzato in una particolare funzione ed è completamente intercambiabile e pronto per essere riutilizzato da altri software.

6.2.7 Diagramma di dispiegamento

Il diagramma in figura 6.7 mostra la collocazione del software Catching Cherries all'interno dell'ambiente di esecuzione Dalvik (vedi paragrafo 2.3.4), a sua volta ospitato da un dispositivo Android.

L'artefatto Catching Cherries è diviso in due macro-componenti, il primo è formato da tutti i componenti ausiliari (Audio Component, Game common services, Pause Manager Component, Score Manager Component), il secondo corrisponde al componente Catching Cherries Core.

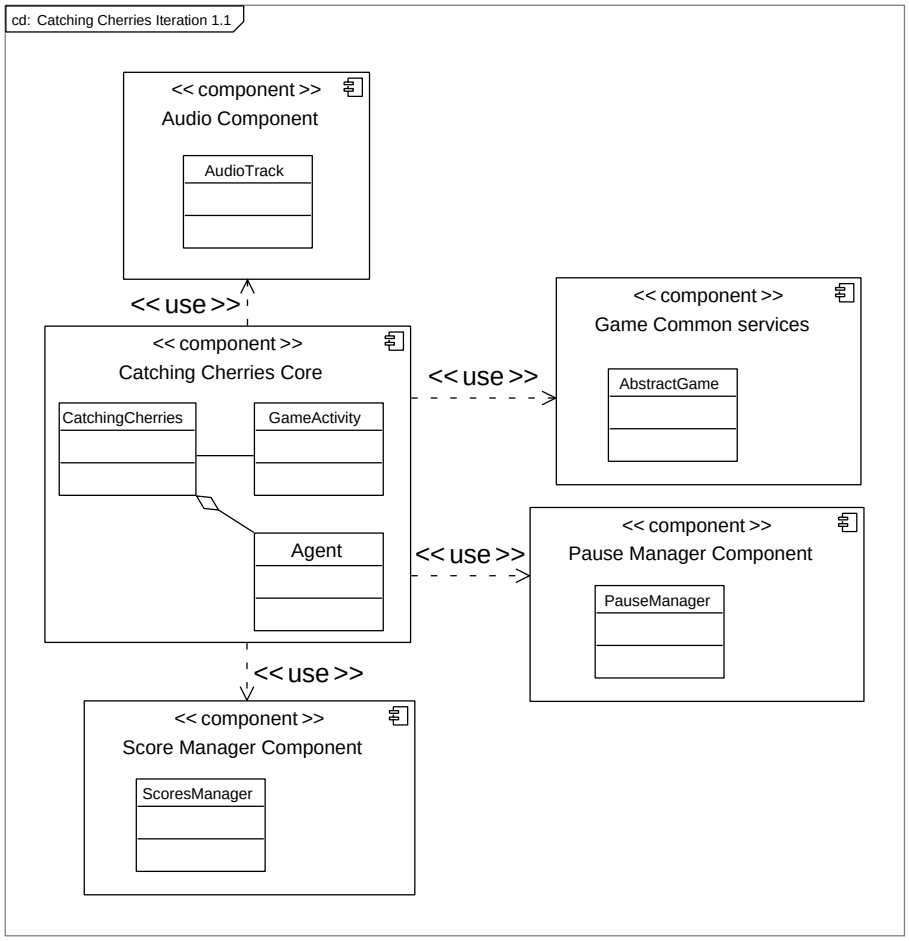


Figura 6.6: Diagramma dei componenti riferito all'applicazione nel suo stadio finale di sviluppo

6.2.8 Gestione eventi

Durante le varie iterazioni della fase Construction è stato deciso come gestire una serie di eventi di diversa natura.

Verrà descritta la gestione degli eventi relativi al cambiamento di stato della Activity del gioco. Prima di entrare nel merito della gestione di tali eventi è bene fare qualche accenno sul ciclo di vita delle Activity Android [34].

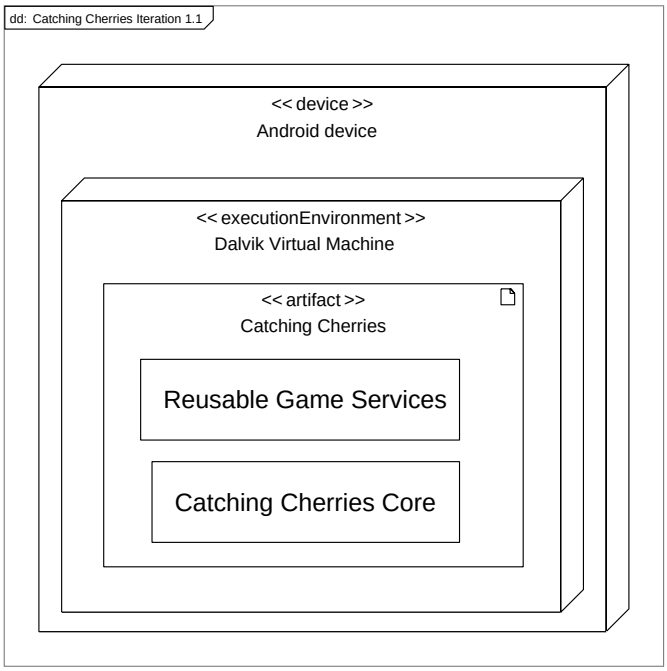


Figura 6.7: Diagramma di dispegamento riferito all'applicazione nel suo stadio finale di sviluppo

Ciclo di vita della Activity Android

In questa descrizione si farà riferimento alla figura 6.8.

Una Activity rappresenta una particolare finestra o schermata, essa può passare attraverso quattro stati:

- Active - La Activity è in esecuzione, è visualizzata ed è in primo piano, questo è lo stato di normale operatività.
- Paused - La Activity è in esecuzione, è visualizzata ma non è in primo piano, in questo stato essa non può ricevere input dall'utente.
- Stopped - La Activity è in esecuzione ma è nascosta da un'altra Activity che è stata lanciata.
- Dead - La Activity è in questo stato se non è ancora stata eseguita oppure se è stata terminata.

Il passaggio tra i vari stati avviene attraverso diversi metodi lanciati dallo Activity Manager non appena è necessario effettuare la transizione. Per gestire l'effetto di ogni transizione in maniera diversa da quella standard prevista da Android, è necessario eseguire degli override dei metodi elencati di seguito.

- onCreate() - Viene invocato quando la Activity viene creata.
- onRestart() - Viene chiamato quando una Activity era nello stato Stopped e sta essendo riavviata. La sua chiamata è sempre seguita da una chiamata al metodo onStart().
- onStart() - Viene invocato quando la Activity sta diventando visibile.
- onResume() - Viene invocato quando la Activity inizia ad interagire con l'utente.
- onPause() - Viene chiamato appena prima che il sistema chiami il metodo onResume() a favore di un'altra Activity.
- onStop() - Viene chiamato quando la Activity non è più visibile dato che c'è n'è un'altra che la copre.
- onDestroy() - Viene chiamato prima che la Activity viene terminata.

Gestione Stati Paused e Stopped

Per un gioco è importante tenere traccia dello stato della partita anche quando la relativa schermata non è visualizzata.

Per poter gestire tale situazione è necessario agire per mezzo dei metodi onPause() e onStop() e di conseguenza anche del metodo onResume(). Dovrà quindi essere eseguito l'override di tali metodi, nelle modalità descritte di seguito.

Se per qualunque motivo la schermata di gioco viene nascosta (per esempio per effetto dell'arrivo di una chiamata oppure perché l'utente ha premuto

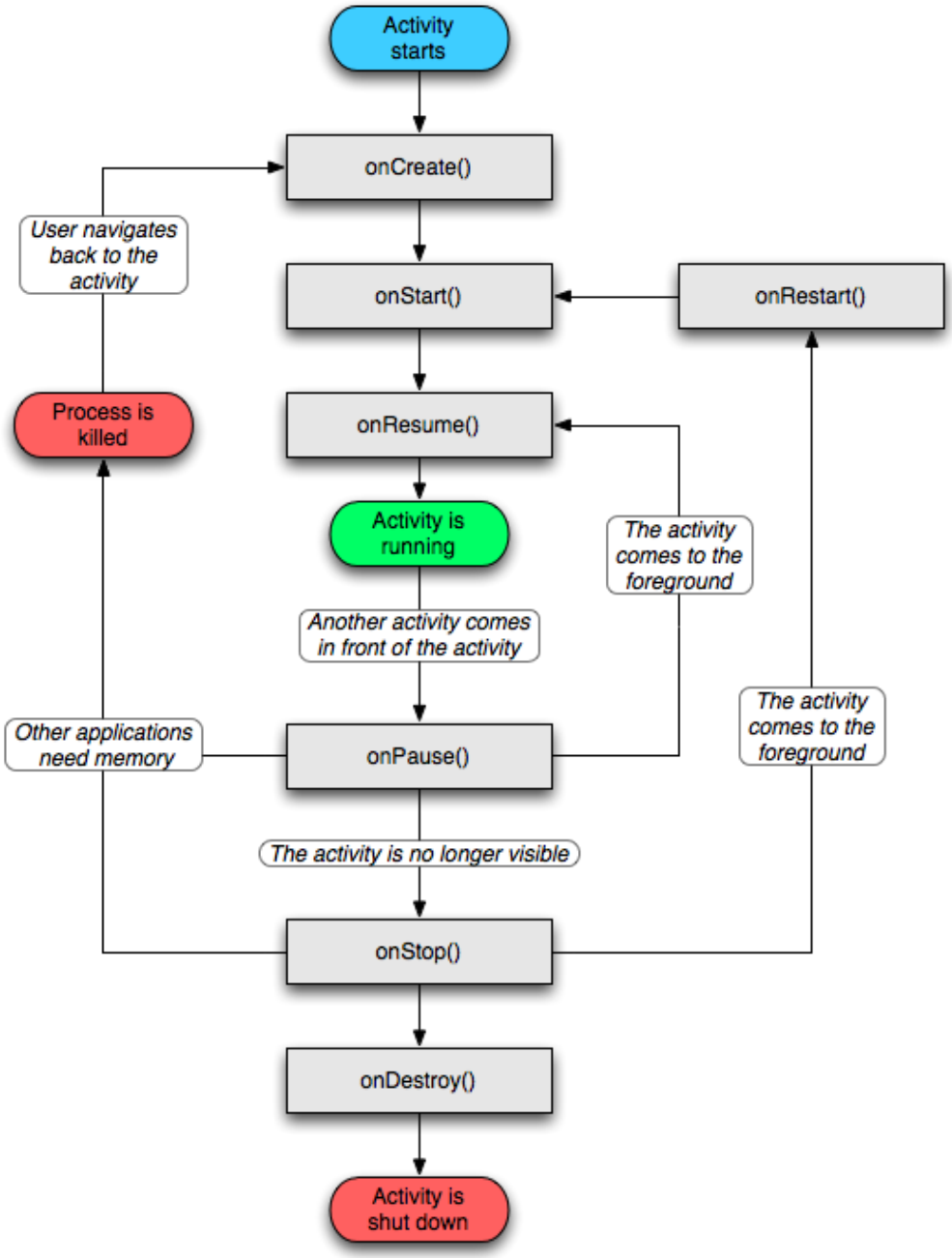


Figura 6.8: Il ciclo di vita delle Activity. [6]

il tasto HOME), la partita deve essere messa in pausa e il suo stato deve essere conservato.

Quindi sia nel metodo onPause() che nel metodo onStop(), dovrà essere richiamata la funzione per sospendere la partita.

Quando la schermata di gioco torna ad essere visualizzata, il metodo onResume() dovrà far apparire la finestra di dialogo della pausa. I metodi appena descritti sono implementati nel paragrafo 6.3.1.

6.3 Implementazione

In questo paragrafo verranno descritte alcune parti del sorgente di Catching Cherries, relativo all'ultima iterazione di Construction. Questo significa che il codice presentato rappresenta l'ultimo raffinamento del prodotto prima di passare alla fase di rilascio.

I principi seguiti durante le varie fasi di implementazione sono descritti nel paragrafo 5.4.1.

La struttura finale del progetto composto da tutte le sue risorse è mostrato nella figura 6.9.

La figura 6.10 mostra gli elementi grafici utilizzati per rappresentare gli agenti ed il carrello.

Alcuni dei metodi che verranno illustrati erano stati implementati già nella fase di Elaboration, ma è stato scelto di includerne soltanto la versione definitiva per limitare la ridondanza di informazioni.

6.3.1 Classe GameActivity

Rispetto alla sua versione iniziale (vedi paragrafo 5.4.2), questa classe è stata aggiornata in modo che venissero gestiti efficacemente anche gli stati Paused e Stopped del ciclo di vita della Activity.

Verrano quindi descritte le implementazioni dei metodi:

- onStop (Listato 6.1)
- onResume (Listato 6.2)

Il metodo onPause non fa altro che invocare il metodo onStop.

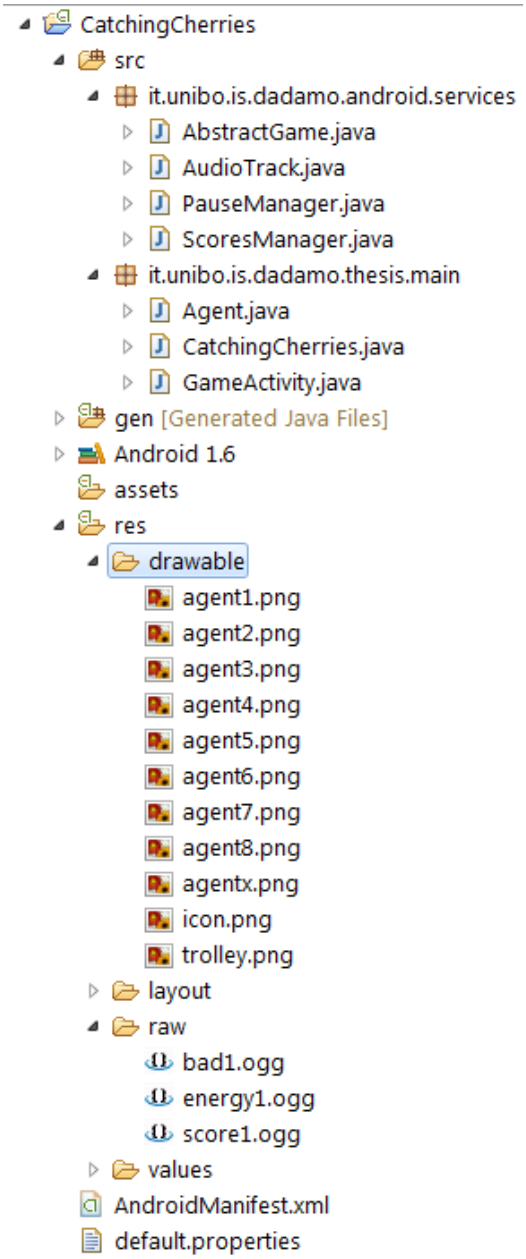


Figura 6.9: La struttura del progetto.

Metodo onStop

Il metodo onStop viene richiamato quando la schermata del gioco viene per qualche motivo coperta da un'altra schermata. L'implementazione di questo

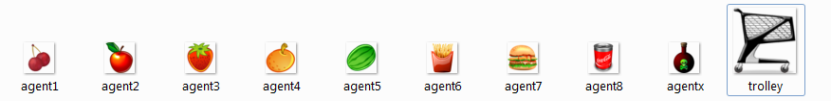


Figura 6.10: Gli elementi grafici di Catching Cherries.

metodo consiste semplicemente nell'invocazione del metodo che sospende la partita se essa è in atto.

Ogni volta che si effettua un override di uno dei metodi descritti nel paragrafo 6.2.8 è necessario per prima cosa fare un'invocazione al corrispondente metodo della superclasse.

```
/**
 * Viene richiamato quando la schermata di gioco viene
 * nascosta per qualche motivo. Se il gioco è in modalità
 * partita, essa viene messa in pausa appena la schermata
 * viene nascosta.
/*
@Override
protected void onStop() {

   super.onStop();

   if (CatchingCherries.playing) {
      CatchingCherries.pauseGame();

   }
}
```

Listato 6.1: Metodo onStop

Metodo onResume

Questo metodo viene richiamato appena la schermata di gioco viene visualizzata. Se la partita era stata messa in pausa, viene creata la relativa finestra di dialogo.

```
/**
 * Viene richiamato appena la schermata di gioco viene messa
 * in primo piano. Se la partita era stata messa in pausa,
```

```
 * viene creata la finestra di dialogo della pausa.
 */
@Override
protected void onResume() {

   if (CatchingCherries.gamePaused ) {
      PauseManager.showPauseDialog();
   }
   super.onResume();
}
```

Listato 6.2: Metodo onResume

6.3.2 Ottimizzazione ViewRefresher

Nel corso della fase di Construction sono state messe in atto numerose ottimizzazioni, c'è n'è una particolarmente rilevante che permette di risparmiare molte risorse.

La classe interna ViewRefresher (descritta nel paragrafo 5.4.2 a pagina 83) si occupa di invalidare la vista dopo un certo intervallo di tempo.

Questa operazione avveniva in maniera incontrollata e non soltanto durante la partita, di conseguenza veniva spesa invano un gran quantità di risorse. Pertanto era urgente apportare una modifica a questo sistema.

La versione iniziale della classe interna ViewRefresher vista nel listato 5.3, non esercitava alcun controllo sull'esecuzione dell'invalidazione della schermata. È bastato aggiungere all'interno del run() un controllo che verifica se la schermata di gioco necessita di essere ridisegnata.

```
// la vista non viene invalidata se il gioco non è in
   modalità partita.
if (playing && !gamePaused) {
   postInvalidate();
   }
}
```

Listato 6.3: Classe ViewRefresher versione ottimizzata

6.3.3 Classe CatchingCherries

La classe CatchingCherries verrà analizzata in maniera più estesa rispetto alle altre classi, dato che la maggior parte della logica del gioco risiede in essa.

Di questa classe verrano descritte le seguenti parti:

- implementazione della pausa (Listato 6.4);
- metodo onKeyDown (Listato 6.5);
- metodo onKeyUp (Listato 6.6);
- metodo createAgent (Listato 6.7);
- implementazione dell'accelerazione degli agenti;
- metodo drawAgents (Listato 6.8);
- metodo verifyContacts (Listato 6.9).

I metodi createAgent, drawAgents (attraverso il metodo drawgraphics), e verifyContacts fanno parte del main loop e vengono invocati ad ogni sua iterazione.

Implementazione della pausa

Il gioco può essere messo in pausa in modo diretto o indiretto:

- (*a*) Modo diretto - l'utente preme il tasto virtuale "Pause Game", il tasto BACK, il tasto MENU oppure il tasto P.
- (*b*) Modo indiretto - La Activity passa attraverso lo stato Paused oppure Stopped, questo può avvenire a causa di un'azione dell'utente (ad esempio mettendo il dispositivo in standby) oppure a causa di eventi come la ricezione di una chiamata.

In entrambi i casi il gioco reagisce sospendendo la partita. Nel primo caso la finestra di dialogo relativa alla pausa appare immediatamente, mentre nel secondo caso appare nel momento in cui la schermata di gioco viene messa di nuovo in primo piano.

Il primo tentativo di implementazione della pausa era più complesso in quanto per congelare lo stato della partita agiva su numerose variabili, esso è risultato poco flessibile dal momento dell'introduzione dell'accelerazione. Dunque è stato trovato un sistema più astratto basato direttamente sulla sospensione del ridisegno di tutti gli elementi grafici (la sospensione del ridisegno è mostrata nel listato 6.3). La partita viene sospesa invocando il metodo pauseGame (listato 6.4), che è composto da tre istruzioni:

```
/**
 * Richiamare questo metodo per mettere il gioco in pausa,
 */
static void pauseGame(){
   if (!gamePaused) {
      gamePaused = true;
      // Momento adatto per far eseguire il GC.
      System.gc();
   }
}
```

Listato 6.4: Metodo pauseGame

L'istruzione `System.gc();` invoca l'esecuzione del Garbage Collector. Questo è uno dei momenti ottimali per farlo visto che il gioco è in una pausa naturale, in questo modo si riduce la probabilità che il GC venga eseguito durante le animazioni.

La finestra di dialogo della pausa è gestita dalla classe PauseManager che offre il metodo pubblico showPauseDialog. L'invocazione di quest'ultimo crea la finestra di dialogo con le tre possibilità. Ogni tasto della finestra è associato ad una particolare funzione, quindi se il giocatore vuole continuare la partita verrà richiamato soltanto il metodo resumeGame, che setta la variabile `gamePaused` a false, questa operazione viene eseguita in tutti e tre i casi. Se invece egli vuole riavviare la partita, verrà richiamato il metodo

restartGame, che non fa altro che azzerare tutti dati della partita corrente. Se il giocatore vuole abbandonare la partita, verrà richiamato il metodo gameOver, che fa passare il gioco in modalità attesa e fa in modo che venga mostrata la schermata principale. In questo caso, se il punteggio del giocatore rientra in classifica, gli verrà chiesto di inserire il proprio nome.

Metodo onKeyDown

Questo metodo è richiamato quanto viene rilevata la pressione di un tasto. Ogni tasto è associato ad un codice che lo identifica.

Per ogni stato del gioco ci sono tasti legati a varie operazioni:

- Durante la partita è possibile usare i tasti direzionali oppure i tasti A e D per muovere il carrello a destra o sinistra. Se viene premuto il tasto $SINISTRA$ oppure il tasto A , il gioco memorizza nella variabile `trolleyDirection` il valore -1 per indicare che il carrello adesso dovrà essere spostato verso sinistra. Per quanto riguarda i tasti $DESTRA$ o D, `trolleyDirection` assumerà il valore 1 che indica che il carrello adesso dovrà essere spostato verso destra. Se viene rilevata la pressione del tasto R, la partita verrà riavviata. Se viene premuto il tasto P, il gioco viene messo in pausa e viene mostrata la relativa finestra di dialogo.

- Se non si è nel corso di una partita e viene premuto il tasto S, la partita viene avviata.

- Se viene premuto il tasto B, il sistema dovrà comportarsi come se fosse stato premuto il tasto BACK.

```
/**
 * Metodo che gestisce gli eventi relativi alla pressione di
 * tasti, associa l'azione appropriata ad ogni tasto
 * previsto dal gioco.
 */
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
```

```
/*
 * Qui vengono gestiti gli input immessi durante la
 * partita.
 */
if (playing) {

   /*
    * Se il tasto premuto è il tasto SINISTRA oppure A, la
    * variabile che indica la direzione del carrello sarà
    * associata al codice -1.
    */
   if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT
      || keyCode == KeyEvent.KEYCODE_A)
   {
      trolleyDirection = -1;
   }
   /*
    * Se il tasto premuto è il tasto DESTRA oppure D, la
    * variabile che indica la direzione del carrello sarà
    * associata al codice 1.
    */
   if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT
      || keyCode == KeyEvent.KEYCODE_D)
   {
      trolleyDirection = 1;
   }
   /*
    * Se viene premuto il tasto R, la partita verrà
    * riavviata.
    */
   if (keyCode ==  KeyEvent.KEYCODE_R) {
         resetGame();
   }
   /*
    * Se viene premuto il tasto P ed il gioco non era già
    * in pausa allora viene messo in pausa.
    */
   if (keyCode ==  KeyEvent.KEYCODE_P) {
      if (!gamePaused) {
```

```
            pauseGame();
            PauseManager.showPauseDialog();
         }
      }
   /*
    * Se il gioco non è in modalità partita, verrano gestiti
    * gli input per le altre funzionalità del gioco.
    */
   }else {
      if (keyCode == KeyEvent.KEYCODE_S) {

         if (!playing && !GameActivity.showingScore
            && !gameAborted )
         {
            startGame();
         }
      }
      if (keyCode == KeyEvent.KEYCODE_B){
         GameActivity.onBackButtonPressed();
      }
   }
   return true;
}
```

Listato 6.5: Metodo onKeyDown

Metodo onKeyUp

Questo metodo viene invocato nel momento in cui è stato rilevato il rilascio della pressione di un tasto.

Il principio di funzionamento è analogo al metodo onKeyDown e si occupa di gestire il rilascio dei tasti che indicano la direzione del carrello e i tasti MENU e BACK. Nel momento in cui l'utente smette di tenere premuto uno dei tasti direzionali (oppure A o D), il gioco interpreta quest'azione dando un valore neutro alla variabile `trolleyDirection`. Se invece l'evento era relativo al rilascio del tasto MENU o BACK, viene invocato un metodo che interpreterà tale pressione in base allo stato corrente del gioco.

```
/**
 * Metodo che gestisce gli eventi relativi al rilascio della
 * pressione dei tasti.
 */
@Override
public boolean onKeyUp(int keyCode, KeyEvent event) {
   /*
    * Se il tasto rilasciato è il tasto SINISTRA, DESTRA, A
    * o B la variabile che indica la direzione del carrello
    * sarà associata al codice 0, per fermare il carrello.
    */
   if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT
      || keyCode == KeyEvent.KEYCODE_DPAD_RIGHT
      || keyCode == KeyEvent.KEYCODE_A
      || keyCode == KeyEvent.KEYCODE_D)
   {
      trolleyDirection = 0;
   }
      /*
       * Se il codice corrisponde al tasto BACK, viene
       * richiamato il metodo che interpreta l'azione
       * da associare al suo rilascio in base allo stato
       * corrente del gioco.
       */
      if (keyCode == KeyEvent.KEYCODE_BACK
         && event.getRepeatCount() == 0 )
         {
         GameActivity.onBackButtonPressed();
      /*
       * Se il codice corrisponde al tasto MENU e il gioco è
       * in modalità partita, il suo rilascio viene
       * interpretato come richiesta di pausa.
       */
      }else if (keyCode == KeyEvent.KEYCODE_MENU
         && event.getRepeatCount() == 0 && playing)
      {
         GameActivity.onBackButtonPressed();
      }
```

```
    return true;
}
```

Listato 6.6: Metodo onKeyUp

Metodo createAgent

Questo metodo regola la generazione degli agenti in base alla difficoltà del gioco.

La lista degli agenti Gli agenti non vengono istanziati, ma "riciclati" da una lista che contiene tutti gli agenti, marcando quelli che non sono al momento visualizzati. In questo modo si è evitato di eseguire un'allocazione di memoria per ogni agente presente nel gioco.

Tra le molteplici operazioni eseguite durante l'inizializzazione del gioco, viene creato un array di oggetti Agent "generici" che ha un numero di elementi pari al valore della variabile `maxVerticalAgents`. Questa variabile è calcolata dividendo l'altezza dello schermo per l'altezza massima che può avere un agente, quindi indica il massimo numero di agenti che possono essere presenti contemporaneamente in una schermata.

Sistema di generazione degli agenti Viene usato un sistema che fa in modo che con l'aumento della difficoltà, gli agenti vengano generati con maggiore frequenza.

La variabile `agentLimiter` viene usata per contare il numero di fotogrammi (o iterazioni del main loop) visualizzati dall'ultima volta che è stato generato un agente. Dopodiché viene calcolato un coefficiente di "difficoltà" D = (`linearDifficulty`[1] / `difficultyStep`[2]), che verrà confrontato con

[1] `linearDifficulty` viene decrementata ad ogni aumento di livello del gioco, ogni suo decremento corrisponde ad un lieve aumento del tasso di generazione degli agenti dato che nel rapporto D essa è al numeratore ed il suo valore iniziale è decine di volte più grande di `difficultyStep`.

[2] `difficultyStep` viene incrementata ogni N aumenti di livello, ogni suo incremento corrisponde ad un aumento considerevole del tasso di generazione degli agenti dato che nel rapporto D essa è posizionata al denominatore ed il suo valore iniziale è decine di volte più piccolo di `linearDifficulty`.

`agentLimiter`. Se `agentLimiter` risulta superiore a D, la prima condizione per poter generare un nuovo agente è soddisfatta.

La seconda condizione è soddisfatta in tutti i casi in cui lo schermo non è già saturo di agenti.

Viene generata in maniera casuale la tipologia dell'agente, il tipo ciliegia ha una probabilità di essere estratto di $1/K$, dove K è il numero delle categorie di agenti. Gli altri tipi di agente hanno una probabilità di essere estratti pari al numero degli agenti che compongono la categoria dell'agente in questione diviso K.

A questo punto viene assegnato all'agente il suo passo[3], viene estratta in modo casuale la coordinata orizzontale in cui verrà posizionato, ed infine gli vengono assegnate le dimensioni (in base al suo tipo), ed infine gli viene assegnata la coordinata verticale iniziale.

```
/**
 * Regola il tasso di creazione degli agenti, in rapporto
 * alla difficoltà del gioco.
 */
private static void createAgent() {
   /*
    * Numero di frame visualizzati dall'ultima
    * creazione di agente.
    */
   agentLimiter++;
   /*
    * Questa è la prima condizione da soddisfare per poter
    * generare un nuovo agente.
    */
   if(agentLimiter > linearDifficulty / difficultyStep) {
      agentLimiter = 0;
      int agentIterator = 0;
      /*
       * Un indice dell'array degli agenti che corrisponde ad
       * un agente che non è correntemente visualizzato.
```

[3]Il passo nel contesto di questo gioco è il numero di pixel che un agente percorre ad ogni frame.

```
 */
int freeIndex = -1;

int randomType;
int randomFamily;
int randomPosition;
/*
 * Si resta nel while finché non è stato superato il
 * limite degli agenti visualizzabili
 * contemporaneamente e l'agente correntemente
 * analizzato è già visualizzato.
 */
while (agentIterator < maxVerticalAgents & agents[
   agentIterator].xCoordinate >= 0){
   agentIterator++;
}
/*
 * Se la prima condizione del while non risulta
 * soddisfatta vuol dire che è stato superato il limite
 * degli  agenti visualizzabili  contemporaneamente
 * e di conseguenza non bisogna generare un agente.
 */
if (agentIterator < maxVerticalAgents){
freeIndex = agentIterator;
}
if (freeIndex >= 0) {
   /*
    * Si estrae a caso un numero compreso tra 0 e
    * il numero delle categorie di agenti.
    */
   randomFamily = (int) (Math.random()
      * AGENT_FAMILIES);
   /*
    * Se il numero estratto è pari al codice
    * del tipo ciliegia, l'agente
    * creato sarà di tipo ciliegia.
    */
   if (randomFamily == Agent.CHERRY) {
      randomType = Agent.CHERRY;
```

```
            /*
             * altrimenti si estrae a caso la tipologia
             * dell'agente.
             */
            }else {
               randomType = (int) (Math.random()
                  * AGENT_TYPES - 1) + 1 ;
            }
            agents[freeIndex].type = randomType;

            // Viene impostato il passo dell'agente.
            agents[freeIndex].pace = AGENT_STARTING_PACE +
               paceIncreaser;

            /*
             * Si estrae a caso la coordinata orizzontale
             * che avrà l'agente.
             */
            randomPosition = (int) (Math.random() * (ScreenWidth
                - agents[freeIndex].width) + 1 );

            /*
             * si impostano le dimensioni e le coordinate
             * dell'agente.
             */
            agents[freeIndex].xCoordinate = randomPosition;
            agents[freeIndex].height =
               agentsHeights[randomType];
            agents[freeIndex].width = agentsWidths[randomType];
            agents[freeIndex].yCoordinate =
               menuHeight - agentsHeights[randomType];

         }
      }
   }
```

Listato 6.7: Metodo createAgent

Implementazione dell'accelerazione degli agenti

L'accelerazione è l'aumento della velocità nel tempo, si calcola in m/s^2, dove m indica i metri ed s i secondi. Ma nella programmazione grandezze fisiche reali come metri e secondi non sempre hanno senso, ed è più conveniente ragionare in pixel e fotogrammi.

Per Catching Cherries è stato adottato un semplice sistema in grado di simulare una pseudo-accelerazione di gravità.

Prima di introdurre l'accelerazione, gli agenti avevano una velocità costante di n pixel (nell'ordine delle unità) per fotogramma per tutta la loro caduta. La velocità è stata pensata più come un passo, dato che i fotogrammi sono una grandezza discreta e non continua come il tempo. La variabile utilizzata per memorizzare il passo degli agenti è

```
int pace;
```

che è un campo della classe Agent.

Gli agenti si muovono lungo delle coordinate verticali y, mentre la loro coordinata orizzontale x è fissata dal momento della loro creazione. Nel punto più in "alto a sinistra" dello schermo le coordinate sono entrambe pari a 0 e nel punto più in "basso a destra" esse sono pari al massimo numero contenibile di pixel orizzontali e verticali, rispettivamente (la risoluzione corrente del display).

Ad ogni fotogramma il valore della coordinata y degli agenti viene incrementata di un valore pari a `pace`.

Questa è l'istruzione che implementava il movimento degli agenti:

```
/*
 * La coordinata originale dell'agente viene incrementata
 * di una certa quantità di pixel.
 */
agents[agentIndex].yCoordinate += agents[agentIndex].pace;
```

Per simulare l'effetto dell'accelerazione è necessario aumentare questo passo ad ogni fotogramma, in modo che gli agenti "percorrano" più pixel per fotogramma.

L'idea è stata quella di moltiplicare la coordinata y corrente dell'agente per un coefficiente che rappresenta l'accelerazione, dopodiché sommare questo prodotto al passo ed infine assegnare il risultato alla variabile che contiene la coordinata dell'agente. Questa procedura viene implementata nell'istruzione seguente:

```
agents[agentIndex].yCoordinate =
 (int) (agents[agentIndex].pace +
 (agents[agentIndex].yCoordinate * agentAcceleration));
```

Dove `agentAcceleration` è un campo della classe CatchingCherries:

```
private static float agentAcceleration;
```

In questo modo si può regolare l'accelerazione degli agenti semplicemente modificando una variabile. Calibrare in maniera ottimale l'accelerazione è un'attività meno immediata della sua implementazione, pertanto verrà svolta nel momento in cui verrà messa a punto la difficoltà del gioco.

Metodo drawAgents

Questo metodo viene richiamato dal metodo drawGraphics del main loop, si occupa di disegnare ad ogni frame gli agenti nella loro posizione corretta. Inoltre implementa il sistema di accelerazione appena descritto e fa in modo di nascondere gli agenti che sono arrivati in fondo allo schermo.

```
/**
 * Disegna tutti gli agenti nella posizione appropriata.
 * Implementa un sistema per simulare l'accelerazione
 * degli agenti.
 */
private static void drawAgents(Canvas canvas) {
   // Viene iterata la lista degli agenti.
   for (int agentIndex = 0; agentIndex < maxVerticalAgents;
       agentIndex++) {
       /*
        * Questa condizione è soddisfatta soltanto se l'agente
        * correntemente analizzato è marcato come
```

```
         * visualizzabile.
         */
        if (agents[agentIndex].xCoordinate >= 0) {
           /*
            * Qui è implementata l'accelerazione. Con questo
            * assegnamento si ottiene la coordinata y in cui
            * disegnare l'agente corrente.
            */
           agents[agentIndex].yCoordinate =
           (int) (agents[agentIndex].pace +
           (agents[agentIndex].yCoordinate * agentAcceleration)
              );
              /*
               * Se l'agente ha raggiunto la fine del display,
               * esso viene etichettato come nascosto.
               */
              if (agents[agentIndex].yCoordinate
                 > screenHeight)
              {
                 agents[agentIndex].xCoordinate = -1;
              } else {
              /*
               * Qui viene disegnato l'agente fornendo al
               * metodo drawBitmap l'immagine che raffigura
               * l'agente, le sue coordinate e l'oggetto
               * paint utilizzato per il disegno.
               */
              canvas.drawBitmap(agentImages[agents[agentIndex].
                 type],
                 agents[agentIndex].xCoordinate,
                 agents[agentIndex].yCoordinate,
                 IMAGE_PAINTER);
           }
        }
     }
  }
```

Listato 6.8: Metodo drawAgents

Metodo verifyContacts

Questo metodo verifica se è avvenuto un contatto tra il carrello ed un agente, in caso positivo viene invocato il metodo che dovrà individuare la tipologia dell'agente per attuare l'effetto appropriato. Se è stato rilevato un contatto, l'agente dovrà essere catalogato tra quelli che non devono essere visualizzati.

```
/**
 * Controlla se è avvenuto un contatto tra il carrello ed un
 * agente ed in caso positivo lancia il metodo che si occupa
 * di attuare le conseguenze del contatto.
 */
private static void verifyContacts() {
   // vengono iterati tutti gli agenti
   for (int n = 0; n < maxVerticalAgents; n++) {
      /*
       * I controlli verranno effettuati solo se l'agente è
       * correntemente visualizzato.
       */
      if (agents[n].xCoordinate >= 0) {
         /*
          * Il contatto è avvenuto se la superficie
          * dell'agente in questione interseca
          * quella del carrello.
          */
         if (trolleyXCoordinate + trolleyWidth >
            agents[n].xCoordinate &&
            trolleyXCoordinate < agents[n].xCoordinate +
            agents[n].width &&
            trolleyYCoordinate + trolleyHeight >
            agents[n].yCoordinate &&
            trolleyYCoordinate < agents[n].yCoordinate +
            agents[n].height)
         {
            // Viene attuato l'effetto del contatto.
            Agent.processEffect(agents[n].type);
            /*
             * L'agente coinvolto viene etichettato
```

```
                * come non visualizzabile.
                */
               agents[n].xCoordinate = -1;
            }
         }
      }
   }
```

Listato 6.9: Metodo verifyContacts

6.3.4 Classe ScoresManager

Questa classe fornisce tutte le funzionalità per la gestione dei punteggi degli utenti di un gioco, esse sono:

- Creare la lista dei punteggi più alti in base ai parametri del gioco specifico e memorizzarla in maniera persistente.
- Verificare se un punteggio rientra nella lista, ed in caso positivo individuare la posizione in cui va inserito.
- Permettere l'inserimento del nome del giocatore ed associarlo al relativo punteggio.
- Permettere l'accesso alla lista in qualunque momento, tramite un'interfaccia grafica.

Di questa classe verrà analizzato il metodo checkIfHighScore (Listato 6.10).

Metodo checkIfHighScore

Questo metodo si occupa di calcolare la posizione in classifica di un determinato punteggio, se il punteggio è fuori classifica, il metodo restituisce il valore -1, altrimenti restituisce la posizione nella classifica. Prende in input il valore del punteggio, il numero dei punteggi che compongono la classifica ed il Context dell'applicazione (racchiude una serie di informazioni su di essa) che sta richiedendo il servizio.

Per prima cosa viene verificato se la lista dei punteggi è già presente in memoria, in caso negativo essa viene cercata nel file opportuno. Se il file non esiste, esso viene creato da una lista vuota e le operazioni successive non verranno eseguite.

Nel caso in cui la lista sia stata trovata, i punteggi che essa contiene vengono confrontati con il punteggio immesso in input finché non viene individuata la posizione che spetta al punteggio inserito. Dopodiché se il punteggio rientra nella lista dei punteggi alti, esso viene inserito nella posizione giusta, scalando i punteggi inferiori. Infine viene ritornata la posizione calcolata.

```
/**
 * Calcola la posizione di un punteggio inserito in input,
 * all'interno di una classifica.
 */
public static int checkIfHighScore(int score, int
    numberOfHighScores, Context applicationContext){

    /*
     * La variabile che conterrà la posizione viene
     * inizializzata  con un valore pari all'ultima
     * posizione della classifica
     */
    int rank = numberOfHighScores+1;
    String [][] theScores = highScores;

    // Se la lista è vuota si procede a caricare il file.
    if (theScores[0][PLAYER_NAME_INDEX].equals(EMPTY)) {
        try {
            //Si cerca il file contenente la lista serializzata.
            theScores = (String[][])
                readObjectFromFile("highscores.txt",
                applicationContext);

        } catch (Exception e) {
            /*
             * Se il file non esiste, ne viene creato uno
             * contenente una lista vuota.
             */
            ScoresManager.writeObjectToFile("highscores.txt",
```

```
            applicationContext, theScores);
         theScores = highScores;
      }
   }
   boolean rankFound = false;
   /*
    * Si continua ad iterare la lista finché non
    * viene trovata la posizione.
    */
   while (!rankFound && rank != 1) {
      /*
       * Se il punteggio inserito in input è maggiore di
       * quello nella posizione immediatamente superiore,
       * si continua a confrontarlo con i punteggi più alti,
       * altrimenti si esce dal ciclo.
       */
      if (score >
         Integer.parseInt(theScores[rank-2][SCORE_INDEX]))
      {
         rank--;
      }else {
         rankFound = true;
      }
   }
   /*
    * Se il punteggio rientra nella lista bisogna inserirlo
    * e scalare tutti i punteggi minori.
    */
   if (rank <= numberOfHighScores) {
      for (int i = numberOfHighScores-1; i >= rank ; i--) {
         theScores[i][SCORE_INDEX] =
         theScores[i-1][SCORE_INDEX];
         theScores[i][PLAYER_NAME_INDEX] =
         theScores[i-1][PLAYER_NAME_INDEX];
      }
      theScores[rank-1][SCORE_INDEX] = String.valueOf(score);
      /*
       * viene aggiornata la lista dei punteggi
       * da mantenere in memoria.
```

```
        */
       highScores = theScores;
       }else {
          rank = -1;
       }
   return rank;
}
```

Listato 6.10: Metodo checkIfHighScore

6.4 Testing e debugging

Nella fase di Construction il testing è stato effettuato con gli strumenti descritti nel paragrafo 5.5.

Rispetto alla fase di Elaboration, in questa fase il testing è avvenuto in maniera più approfondita, in quanto è stato necessario considerare un maggior numero i fattori.

Attraverso lo AVD manager sono state simulate tutte le possibili combinazioni di versioni di Android e risoluzioni dei display con il quale si era prefissato di rendere compatibile il gioco.

Ma per verificare tutti gli altri aspetti del software è stato necessario effettuare il testing su un dispositivo fisico, in modo da ottenere dati provenienti da una piattaforma Android reale che presenta enormi differenze da quelle virtuali. Il dispositivo utilizzato è un HTC Legend, le caratteristiche più rilevanti ai fini del testing del gioco sono le seguenti:

- Input/Output - Touch-screen con una risoluzione HVGA di 320 X 480 pixel, è quella adottata da circa il 50 % dei dispotivi [45].
- CPU - 600 mhz, relativamente lenta rispetto ai dispositivi di ultima generazione, rappresenta un ottimo scenario di testing.
- Versione di Android 2.2 (Froyo).

Ottimizzare il gioco per tale dispositivo significa quindi renderlo compatibile con buona parte degli smartphone dotati di Android. Durante le fasi di

testing della fase di Construction ci si è concentrati maggiormente sui aspetti prettamente tecnici, rimandando aspetti come l'equilibratura della difficoltà del gioco alla fase di Transition.

I risultati del testing sono stati utilizzati per determinare i requisiti dell'iterazione di Construction successiva.

6.4.1 Screenshots

In questa parte sono stati inclusi alcuni screenshots del gioco relativi al suo stadio finale di sviluppo.

Screenshots provenienti da dispositivi virtuali

La figura 6.11 mostra il gioco in esecuzione in un dispositivo virtuale con risoluzione QVGA (240x320).

Figura 6.11: Il gioco eseguito in un dispositivo virtuale con display QVGA.

La figura 6.12 mostra una schermata della partita, in un dispositivo virtuale con risoluzione WVGA800 (480x800).

La figura 6.13 mostra come appare in un dispositivo virtuale Android 2.3 con risoluzione HVGA, la schermata di inserimento del nome.

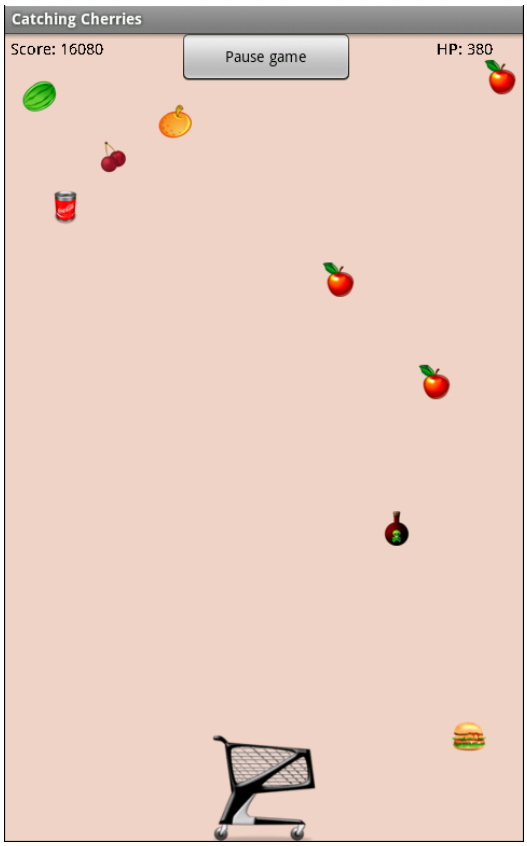


Figura 6.12: Una schermata della partita, in un dispositivo virtuale con display WVGA800.

Screenshots provenienti da HTC Legend

La figura 6.14 mostra il gioco durante una partita, come appare nello HTC Legend.

Se il giocatore mette il gioco in pausa, viene mostrata la finestra di dialogo della pausa, come in figura 6.15.

Nel momento in cui avviene un game over, nel caso in cui il giocatore ottiene un punteggio elevato, appare la schermata in figura 6.16. Se non viene inserito alcun nome, il punteggio verrà associato al nominativo “unknown”.

Dalla schermata iniziale del gioco, è possibile richiedere la visualizzazione della classifica dei punteggi più alti, nel caso in cui essa non sia vuota apparirà come nella figura (6.17). Se invece non sono stati ancora mai inseriti dei

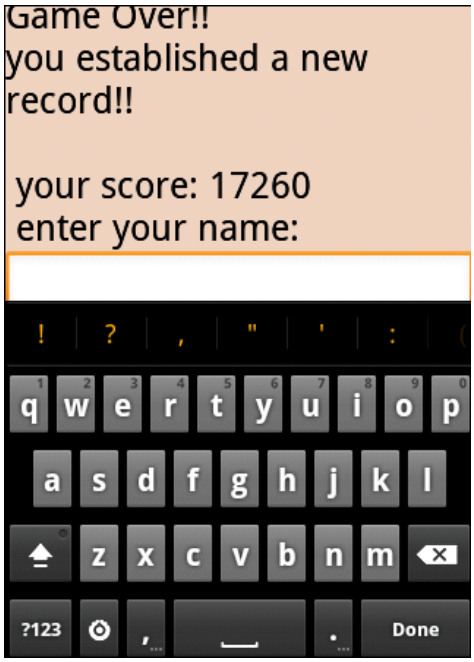


Figura 6.13: La schermata di inserimento del nome, in un dispositivo virtuale Android 2.3 GingerBread.

punteggi, verrà mostrato un messaggio che avverte l'utente di tale situazione. La lista può avere una lunghezza arbitraria e può essere fatta scorrere verso il basso o verso l'alto per visualizzare tutti i punteggi.

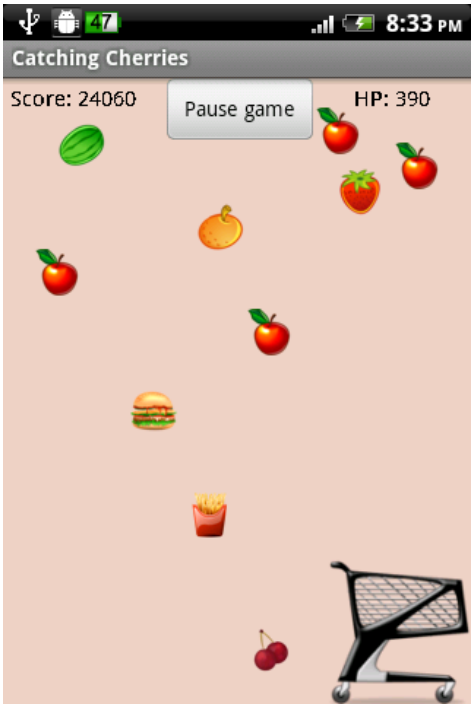


Figura 6.14: Una schermata della partita, in un HTC Legend

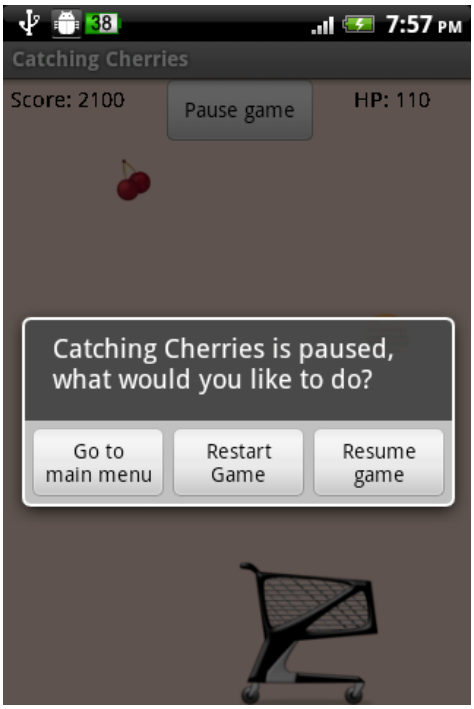


Figura 6.15: La finestra di dialogo della pausa, in un HTC Legend

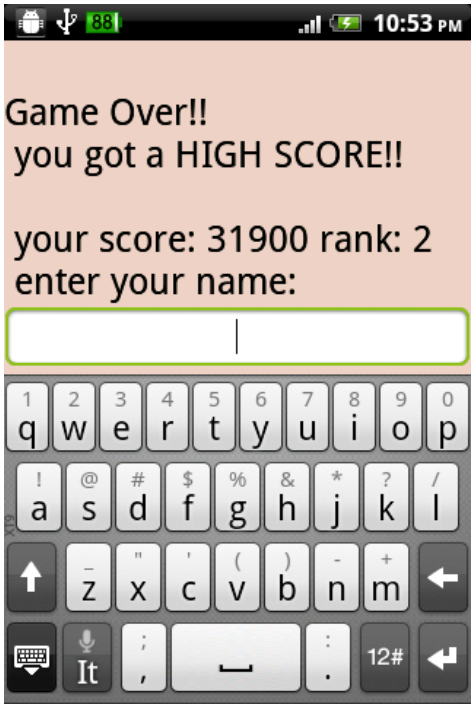


Figura 6.16: La schermata di inserimento del nome, in un HTC Legend

Figura 6.17: La classifica dei punteggi, in un HTC Legend

Capitolo 7

Transition Phase

L'obiettivo della fase di Transition è di realizzare gli scopi per cui il progetto è stato inizialmente concepito.

Durante questa fase si è fatto in modo di far evolvere il software proveniente dalla fase di Construction per farlo diventare un gioco a tutti gli effetti. Per far avvenire questa trasformazione ci si è dovuti focalizzare su coloro per i quali il gioco ha motivo di esistere, cioè gli utenti finali.

Lo sviluppatore è alla stregua degli utenti, il progetto infatti può considerarsi riuscito solamente se soddisfa le loro esigenze.

Anche se il software realizzato nella fase di Construction era completo da un punto di vista tecnico, esso era uno strumento che doveva essere ancora modellato per poter essere definito "gioco". Il passo che doveva ancora essere compiuto riguardava qualità difficilmente misurabili come la capacità di intrattenere.

Nelle attività di testing svolte nella fase di Construction, aspetti del genere sono stati considerati soltanto in maniera indiretta, visto che tali attività sono servite prevalentemente a verificare il gioco da punti di vista prevalentemente funzionali.

In questa fase le operazioni di testing hanno coinvolto anche un ristretto gruppo di utenti, che hanno fornito il feedback necessario all'ottimizzazione finale del software.

Uno degli aspetti che hanno tratto il maggiore giovamento dal feedback

ottenuto è l'equilibratura della difficoltà del gioco.

7.1 Gestione equilibri di gioco

La gestione degli equilibri di gioco è consistita nel trovare un giusto compromesso tra due facce della stessa medaglia:

- Immediatezza del gioco, per catturare rapidamente l'interesse del giocatore. Viviamo in un mondo real-time, tutti vogliono tutto e subito e spesso possono permetterselo, se il gioco pecca di intuitività, viene istantaneamente dimenticato. Se è troppo banale, l'interesse svanisce con la stessa velocità con cui si era creato.

- Sensazione di sfida, per mantenere alto il livello d'interesse nel gioco, stimolando l'utente a non smettere di giocare. Se il gioco è troppo difficile e non esiste un efficace sistema di premiazione, i giocatori non sono motivati abbastanza e scartano il gioco in favore di qualcosa che sia alla loro portata. Avvolte può succedere che un senso di sfida troppo alto possa essere percepito dal giocatore come un bug. [11].

  Allo stesso tempo se il senso di sfida non è abbastanza forte, il giocatore opta per un gioco più stimolante.

Se questi parametri fossero stati oggettivi, non sarebbe nemmeno stato necessario citarli, ma come in molte altre situazioni, non esiste un'unica soluzione.

Il problema riguarda l'enorme varietà di potenziali utenti, dallo hard-core gamer più incallito al giocatore casual per eccellenza.

Il sistema usato per conciliare le diverse esigenze è stato analizzare le reazioni dei diversi utenti a cui è stato sottoposto il gioco e calibrare in funzione di esse i numerosi fattori responsabili dell'impatto con il giocatore.

Premettendo che nella fase di Transition l'implementazione ha avuto un ruolo marginale, si è comunque dovuto agire direttamente sul codice. i fattori che influenzano gli equilibri del gioco sono riportati di seguito.

7.1.1 Frequenza dei fotogrammi

`CatchingCherries.REFRESH_DELAY`

La "velocità" del gioco dipende primariamente da questa costante che regola la frequenza di esecuzione del main loop. Detto in altri termini questa costante rappresenta il periodo[1] p di aggiornamento della schermata espresso in millisecondi, l'intervallo di tempo che deve passare tra un'invalidazione e l'altra della schermata (per la descrizione del sistema di aggiornamento della schermata si veda il paragrafo 4.7.2, per la sua implementazione si veda il listato 5.3). Da essa dipende il frame rate[2] teorico (che coincide con quello massimo) del gioco:

$$fmax = 1/(p \cdot 10^{-3}).$$

Misurato in fps (fotogrammi al secondo).

Il frame rate effettivo, che chiameremo f, nel migliore dei casi è uguale a $fmax$. Esso dipende dal tempo impiegato dal sistema per disegnare una schermata. Questo tempo è funzione della potenza di calcolo e del numero di oggetti da disegnare e per questi motivi va tenuto sotto controllo. Se si imposta un $fmax$ troppo alto, il dispositivo non sarà in grado di erogare un frame rate abbastanza stabile quindi la differenza tra f ed $fmax$ risulterà elevata ed il giocatore avvertirà una palese fluttuazione nel frame rate. Si è scelto quindi di non usare p per regolare la velocità (e quindi la difficoltà) del gioco e fissarlo ad un valore che garantisca animazioni fluide. Il valore scelto è $p = 28$, esso permette di ottenere un $fmax$ di circa 35 fps, ed il testing sul dispositivo (vedi paragrafo 6.4) ha confermato che il gioco riesce a mantenere un frame rate ragionevolmente stabile e soprattutto che scende raramente al di sotto della soglia di percezione dei singoli fotogrammi.[3].

[1] inteso come inverso della frequenza.

[2] Frequenza dei fotogrammi, la frequenza di riproduzione di un'animazione, solitamente misurata in Frames Per Second (fps), fotogrammi al secondo oppure Hertz (Hz), 1 fps = 2 Hz

[3] Il frame rate minimo di un filmato, affinché l'apparato visivo umano non percepisca sfarfallii e artefatti, si attesta sui 30 fps (60 Hz), ma può anche essere superiore per gli oggetti che si muovono molto velocemente sullo schermo. Sebbene un filmato sembri mostrare delle immagini in movimento, infatti, in realtà ciò che viene visualizzato è una sequenza di immagini fisse. Una prima spiegazione del fenomeno, fu ipotizzata dal fisico belga Joseph Plateau, nel XIX secolo, secondo il quale la percezione del movimento continuo era dovuta

7.1.2 Messa a punto velocità di movimento e di creazione degli agenti

Metodo CatchingCherries.updateDifficulty

Il metodo updateDifficulty (Listato 7.1) è il primo che viene richiamato all'interno del main loop. Verifica se è stato raggiunto il punteggio minimo per determinare un aumento del livello di difficoltà. Ad ogni aumento di livello aumenta la percezione della difficoltà agendo su diverse variabili.

L'aumento di difficoltà può avvenire in 3 modi:

1. Attraverso l'aumento del tasso di creazione degli agenti, tramite le variabili `linearDifficulty` e `difficultyStep`.
2. Tramite l'aumento del passo degli agenti, con la variabile `paceIncreaser`.
3. Attraverso l'aumento dell'accelerazione degli agenti, tramite le variabili `agentAcceleration` e `ACCELERATION_STEP`.

Ogni combinazione dei valori di questi parametri determina una diversa esperienza di gioco. In questa fase si è cercato di equilibrare questi valori in modo da ottenere un graduale ma costante aumento della sensazione di difficoltà.

```
/**
 * Gestisce l'aumento della difficoltà, aumentando il tasso
 * di creazione degli agenti, il loro passo, oppure la loro
 * accelerazione.
 */
private static void updateDifficulty(){
   /*
    * Si procede soltanto se è stata raggiunta una quantità
    * di punti superiore al traguardo corrente.
    */
   if (score >= currentLevelingGoal) {
```

alla persistenza delle immagini sulla retina, ma questa teoria fu successivamente smentita da Max Wertheimer: è il cervello che, secondo meccanismi non ancora del tutto chiariti, esegue un'operazione di "assemblaggio" dei singoli fotogrammi, interpretandolo come un movimento. [36]

```
      // Il traguardo viene reimpostato.
      currentLevelingGoal = (int)((currentLevelingGoal +
         LEVELING_DEFAULT_GAP) * gapMultiplier);
      int k = 5;
      /*
       * L'aumento di difficoltà più frequente consiste in un
       * leggero aumento del tasso di creazione degli agenti
       */
      if (levelingCounter < k - 1) {
         levelingCounter++;
         linearDifficulty -= 1;
      /*
       * Nei casi meno frequenti viene aumentato in maniera
       * più percepibile il tasso di creazione degli agenti,
       * viene aumentato il loro passo o la loro
       * accelerazione.
       */
      }else if(levelingCounter < k){
         difficultyStep++;
         linearDifficulty += k;

      }else if (levelingCounter < k + 1){
         difficultyStep++;
         paceIncreaser++;
      }
      else {
         agentAcceleration += ACCELERATION_STEP;
         levelingCounter = 0;
      }
   }
}
```

Listato 7.1: Metodo updateDifficulty

7.1.3 Configurazione effetti del contatto con gli agenti

Nel momento in cui avviene un contatto con un agente, viene richiamato il metodo processEffect (Listato 7.2), della classe Agent. Tale metodo si occupa di individuare il tipo di agente con cui il carrello è entrato in contatto e

attuarne gli effetti appropriati. Durante la fase di Transition sono stati messi a punto i seguenti valori che regolano l'effetto del contatto:

- `Agent.TAKEN_ENERGY` - La quantità base di HP che vengono detratti dal giocatore, viene moltiplicata per un fattore per ottenere il valore specifico al tipo di agente con cui c'è stato il contatto.
- `Agent.TAKEN_SCORE` - La quantità base di punti che vengono detratti dal giocatore, viene divisa per un fattore per ottenere il valore specifico al tipo di agente con cui c'è stato il contatto.
- `Agent.SCORE_UNIT` - La quantità base di punti base che viene assegnata al giocatore per ogni tipo di agente frutta.
- `Agent.GIVEN_ENERGY` - Gli HP che vengono assegnati quando il carrello entra in contatto con le ciliegie.
- `Agent.SCORE_UNIT` - Il numero iniziale degli HP.

Metodo Agent.processEffect

```
/**
 * Metodo che individua la tipologia dell'agente con cui è
 * avvenuto il contatto e applica l'effetto appropriato.
 */
static void processEffect(int type) {

   /*
    * Nel caso più frequente, che è quello della ciliegia,
    * viene assegnata una certa quantità di punti e viene
    * riprodotto l'effetto sonoro appropriato.
    */
   if(type == CHERRY){
   CatchingCherries.hitPoints += GIVEN_ENERGY;
   CatchingCherries.hp1.play();
   }
   /*
    * Nel caso degli altri tipi di frutta viene calcolata la
```

```
 * quantità di punti da assegnare e viene riprodotto
 * l'effetto sonoro appropriato.
 */
else if (type == FRUIT_A || type == FRUIT_B|| type ==
   FRUIT_C|| type == FRUIT_D ) {
   CatchingCherries.score += SCORE\_UNIT + (type * 100);
   CatchingCherries.score1.play();

/*
 * Nel caso l'agente sia di tipo junk food, viene detratta
 * una certa quantità di punti ed HP, viene riprodotto
 * il suono appropriato e viene emessa una
 * lieve vibrazione.
 */
}else if (type == JUNK_A ||type == JUNK_B ||type == JUNK_C
   ) {


   CatchingCherries.score -= TAKEN_SCORE * (1 + type);
   GameActivity.vibrator.vibrate(ms);

   CatchingCherries.hurt1.play();

   CatchingCherries.hitPoints -= TAKEN_ENERGY / (type / 2)
      ;


   if (CatchingCherries.hitPoints <= 0) {
      CatchingCherries.gameOver();
   }

/*
 * Nel caso l'agente sia di tipo veleno, viene detratta
 * una certa quantità di HP, viene riprodotto il suono
 * appropriato e viene emessa una vibrazione per un
 * tempo doppio rispetto al caso junk food.
 */
}else if (type == POISON) {
```

```
        CatchingCherries.hitPoints -= TAKEN_ENERGY;


        GameActivity.vibrator.vibrate( 2 * ms);

        CatchingCherries.hurt1.play();


        if (CatchingCherries.hitPoints <= 0) {
           CatchingCherries.gameOver();
        }
    }
}
```

Listato 7.2: Metodo processEffect

7.2 Rilascio graduale nella user community

Dopo il primo confronto con dei veri utilizzatori, il gioco è pronto per un iniziale rilascio nella user community sotto forma di versione beta. In questo modo sarà possibile ottenere il feedback di cui c'è bisogno per poter apportare continui miglioramenti, senza i rischi legati ad un rilascio prematuro sullo Android market.

Conclusioni

Catching Cherries ha sicuramente raggiunto due obiettivi: quello di poter essere definito gioco e quindi motivo di intrattenimento e quello di costituire un importante esperienza per il suo creatore.

Realizzare un gioco per la piattaforma Android è un'attività in cui vanno conciliati aspetti che non sempre sono compatibili tra loro e che in altri contesti non verrebbero neppure considerati. Ci sono regole da rispettare se non si vuole perdere di vista l'obiettivo primario, che è misurato in fps.

Il prossimo passo sarà quello di approdare sul mercato specifico e gli altri canali di distribuzione.

Gli sviluppi futuri di Catching Cherries prevedono un avvicinamento agli utenti, un gioco che sia in grado di adattarsi ad essi ed alle loro strategie di gioco. Un esempio potrebbe essere la realizzazione di un sistema in cui la difficoltà viene regolata automaticamente in base alle capacità degli utenti, in modo che un giocatore alle prime armi non abbia la consapevolezza di star giocando al minimo livello di difficoltà. Contemporaneamente si dovrebbe cercare di dare ai giocatori più esigenti nuove forme di stimoli, nuove sfide personalizzate.

Un'altra direzione in cui sarebbe utile far crescere il gioco realizzato è l'aspetto grafico, uno dei tanti aspetti non prettamente tecnici che rientrano nelle attività di sviluppo dei giochi.

In Conclusione, Android è indubbiamente una valida piattaforma di sviluppo ma la sua frammentazione sta crescendo al fianco della sua popolarità e rischia di rendere lo sviluppo di applicazioni una procedura insidiosa.

Elenco delle figure

Lista dei Sorgenti

Bibliografia

[1] Android development community | android tutorials. http://http://www.anddev.org/ (ultimo accesso: febbraio 2011).

[2] Android market. https://market.android.com/ (ultimo accesso: febbraio 2011).

[3] Android compatibility, android developers. http://developer.android.com/guide/practices/compatibility.html (ultimo accesso: febbraio 2011).

[4] Android developers, developing in ecplise with adt. http://developer.android.com/guide/developing/eclipse-adt.html (ultimo accesso: febbraio 2011).

[5] Android developers, platform versions. http://developer.android.com/resources/dashboard/platform-versions.html (ultimo accesso: febbraio 2011).

[6] Android developers. http://developer.android.com (ultimo accesso: febbraio 2011).

[7] Apple app-store. http://app-store.appspot.com (ultimo accesso: febbraio 2011).

[8] Beroux.com, object-oriented analysis and design with uml and rup. http://www.beroux.com/english/articles/oop_uml_and_rup.php?part=2 (ultimo accesso: febbraio 2011).

[9] Bongizmo.com, nexus one. `http://www.bongizmo.com/blog/why-nexus-one-is-still-the-best-android-phone/` (ultimo accesso: febbraio 2011).

[10] Chillingo - the premier games publisher. `http://www.chillingo.com/` (ultimo accesso: febbraio 2011).

[11] Chris pruett, google i/o 2010 - writing real-time games for android redux. `http://www.youtube.com/watch?v=7-62tRHLcHk` (ultimo accesso: febbraio 2011).

[12] Center for systems and software engineering, cocomo 2. `http://csse.usc.edu/csse/research/COCOMOII/` (ultimo accesso: febbraio 2011).

[13] Corso di crm di andrea de marco. `corsodicrm.wordpress.com` (ultimo accesso: febbraio 2011).

[14] Designing for performance, android developers. `http://developer.android.com/guide/practices/design/performance.html` (ultimo accesso: febbraio 2011).

[15] Designing for responsiveness, android developers. `http://developer.android.com/guide/practices/design/responsiveness.html` (ultimo accesso: febbraio 2011).

[16] Designing for seamlessness, android developers. `http://developer.android.com/guide/practices/design/seamlessness.html` (ultimo accesso: febbraio 2011).

[17] Distimo publications january 2011. `http://www.distimo.com/` (ultimo accesso: febbraio 2011).

[18] Getting started in android game development. `http://www.rbgrn.net/content/54-getting-started-android-game-development` (ultimo accesso: febbraio 2011).

[19] Eric schmidt's home page. `http://ericschmidt.com/` (ultimo accesso: febbraio 2011).

[20] Martin Folwer. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition).* Addison Wesley, 2003.

[21] Game development for android: A quick primer. `http://android-developers.blogspot.com/2010/06/game-development-for-android-quick.html` (ultimo accesso: febbraio 2011).

[22] Chris pruett, google i/o 2009 - writing real-time games for android. `http://www.youtube.com/watch?v=U4Bk5rmIpic` (ultimo accesso: febbraio 2011).

[23] Hardcore gaming news, angry birds. `http://hcgamingnews.wordpress.com/2010/11/15/angry-birds-coming-to-psn-xbla/` (ultimo accesso: febbraio 2011).

[24] Hardware max, android honeycomb. `http://www.hardwaremax.it/news/telefonia/2420-google-presenta-honeycomb-ed-il-web-store-android-market.html` (ultimo accesso: febbraio 2011).

[25] Home page di android. `http://www.android.com/` (ultimo accesso: febbraio 2011).

[26] Home page di zynga. `http://www.zynga.com/` (ultimo accesso: febbraio 2011).

[27] Il sito ufficiale di eclipse. `http://www.eclipse.org/` (ultimo accesso: febbraio 2011).

[28] Il sito ufficiale di gantt project. `http://www.ganttproject.biz/` (ultimo accesso: febbraio 2011).

[29] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition.* Addison Wesley, 2004.

[30] Liang Wu Mary Meeker, Scott Devitt. Internet trends april 12, 2010. 2010.

[31] Profilo di reto meier su google profiles. `http://www.google.com/profiles/reto.meier` (ultimo accesso: febbraio 2011).

[32] Reto Meier. *Professional Android 2 Development*. Wiley Publishing, Inc., 2010.

[33] Moore's law. `http://www.intel.com/technology/mooreslaw/` (ultimo accesso: febbraio 2011).

[34] Mark L. Murphy. *Beginning Android 2*. Apress, 2010.

[35] Nokia ovi store. `http://store.ovi.com/` (ultimo accesso: febbraio 2011).

[36] Persistenza della visione, wikipedia. `http://it.wikipedia.org/wiki/Persistenza_della_visione`(ultimo accesso: febbraio 2011).

[37] Reuters, google topples nokia from smartphones top spot. `http://uk.reuters.com/article/2011/01/31/oukin-uk-google-nokia-idUKTRE70U1YT20110131` (ultimo accesso: febbraio 2011).

[38] Zigurd Mednieks G. Blake Meike Rick Rogers, John Lombardo. *Android Application Development: Programming with the Google SDK*. O'Reilly Media, Inc., 2009.

[39] Blackberry appworld. `http://us.blackberry.com/apps-software/appworld/` (ultimo accesso: febbraio 2011).

[40] Rovio mobile. `http://www.rovio.com/` (ultimo accesso: febbraio 2011).

[41] rts-systems.com, development process. `http://rts-systems.com/EN/dev_process.html` (ultimo accesso: febbraio 2011).

[42] Profilo di andy rubin. `http://www.crunchbase.com/person/andy-rubin` (ultimo accesso: febbraio 2011).

[43] Rumors sul gphone su cnn money. `http://money.cnn.com/2007/10/09/technology/google_phone.fortune/index.htm?source=yahoo_quote` (ultimo accesso: febbraio 2011).

[44] Sa depot, uml diagram types. `http://www.sa-depot.com/?page_id=662` (ultimo accesso: febbraio 2011).

[45] Screen sizes and densities, android developers. `http://developer.android.com/resources/dashboard/screens.html` (ultimo accesso: febbraio 2011).

[46] Stackoverflow, questions and answers for professional and enthusiast programmers. `http://stackoverflow.com/` (ultimo accesso: febbraio 2011).

[47] Supporting multiple screens, android developers. `http://developer.android.com/guide/practices/screens_support.html` (ultimo accesso: febbraio 2011).

[48] La dichiarazione di andy rubin su android e l'open handset alliance, google official blog. `http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html` (ultimo accesso: febbraio 2011).

[49] The open handset alliance. `http://www.openhandsetalliance.com/` (ultimo accesso: febbraio 2011).

[50] University of Southern California. *COCOMO II Model Definition Manual*.

[51] User interface guidelines, android developers. `http://developer.android.com/guide/practices/ui_guidelines/index.html` (ultimo accesso:, febbraio 2011).

[52] Wikipedia, comparison of android devices. `http://en.wikipedia.org/wiki/Comparison_of_Android_devices` (ultimo accesso: febbraio 2011).

[53] Wikipedia, game programming. `http://en.wikipedia.org/wiki/Game_programming` (ultimo accesso: febbraio 2011).

[54] Wikipedia, l'enciclopedia libera. `http://it.wikipedia.com` (ultimo accesso: febbraio 2011.

[55] Wikiquote, dwight d. eisenhower. `http://en.wikiquote.org/wiki/Dwight_D._Eisenhower` (ultimo accesso: febbraio 2011).

[56] Nick Wingfield. Why can't we stop playing. *The Wall Street Journal*. `http://http://online.wsj.com/article/SB10001424052748703945904575644940111605862.html` (ultimo accesso: febbraio 2011).

[57] World summit award, angry birds. `http://www.wsa-mobile.org/winner/angry-birds-50820101104` (ultimo accesso: febbraio 2011).

[58] Windows phone marketplace. `http://marketplace.windowsphone.com` (ultimo accesso: febbraio 2011).

[59] Chris Wright. *A Brief History of Mobile Games*. Steel Media, 2008.

[60] Youtube.com, development for android. `http://www.youtube.com/watch?v=DSMriibR99E` (ultimo accesso: febbraio 2011).