Davide Aguiari Matricola 0000411115

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

APRILE 2013 – BOLOGNA

Corso di Laboratorio Applicazioni Mobili

Anno Accademico 2012/2013

Monkey Island Swordfighting

1. Scopo

Come progetto per il corso di LAM è stato realizzato il porting Android delle sfide a duello a *insulti pirateschi*, tratto dell'avventura grafica della LucasArts "Monkey Island 1 – The Secret of Monkey Island".
Il risultato è quindi un "casual-game" destinato principalmente agli amanti della saga, sebbene l'introduzione renda il gioco accessibile a qualunque tipo di utenza.
Per la sua realizzazione, sono state utilizzate le API 10 di Android, al fine di permettere la compatibilità con tutti i device con Android 2.3.3 o superiore, tablet compresi.

2. Implementazione

Non pochi sono stati i problemi implementativi, dovuti anche alla decidisione di scegliere un gioco come progetto, senza aver mai avuto esperienze in quel settore, né nella programmazione Android.
Il Thread che si occupa del GameLoop (**GameThread.java**) è il cuore dell'applicazione:
come viene spiegato molto bene in questo topic, http://www.koonsolo.com/news/dewitters-gameloop/, il GameLoop è il motore di tutto.
L'intero loop è semplicemente una sequenza di chiamate:

```
bool game_is_running = true;
while( game_is_running ) {
    update_game();
    display_game();
}
```

Le chiamate sono opportunatamente regolate per mantenere costante il ciclo d'esecuzione, indipendentemente dalla velocità dell'hardware che, nel caso di Android, è decisamente variabile nella sua vastità di device.
Questa separazione netta tra update() e display() ha come comune denominatore il pattern MVC (Model-View-Controller) con la netta divisione dei ruoli nell'intera applicazione.
Il metodo update() sarà quindi chiamato da un controller generale (**GameEngine.java**) con lo scopo di aggiornare tutti i model e la logica dei turni, mentre display() è il metodo della View (del pattern) che visualizzerà a video il tutto.

- L'Activity **Pirates**

A dare il via all'intera applicazione è l'acitivity **Pirates.java** che inizializza il Layout, la SurfaceView che contiene la Canvas, il GameLoop e il GameEngine.
Nell'Activity troviamo anche tutti i metodi utili al salvataggio e ripristino dello stato del gioco in caso di Standby/background, il Listener della ListView, utile al giocatore per selezionare eventuali discorsi/insulti/controInsulti durante il gioco, i metodi di creazione/distruzione della SurfaceView e l'handler inizializzato per scambiare messaggi dal GameLoop Thread all'UI-Thread e viceversa.

- SurfaceView **MainGamePanel**

Dovendo controllare il flusso delle draw() del GameLoop Thread, l'utilizzo di SurfaceView è risultato ottimale: tramite CallBack fornisce l'accesso alla canvas e ai suoi metodi di disegno grafico (come i testi o le drawBitmap).
Inoltre, è la SurfaceView stessa ad avere il compito di avviare o terminare il Thread alla creazione o distruzione della View.

Il suo layout è molto semplice, definito in activity_pirates.xml. E' sufficiente specificare gli attributi minimi e non c'è bisogno di configurare nient'altro della View.

Nella classe MainGamePanel troviamo quindi tutti i metodi necessari al calcolo delle proporzioni delle Bitmap per il touch sullo screen lato utente. Le coordinate x,y vengono infatti definite in proporzione alla larghezza/altezza della canvas, calcolata a tempo di esecuzione.

- Controller **GameEngine**

Il GameEngine è la classe che si occupa di istanziare tutti i protagonisti del pattern MVC.
Inizializza la classe **Dialogs** contenente i discorsi tra pirati e gli insulti del gioco, il custom Adapter per la Listview, i controller e i renderer.

- Controller **Guybrush, Enemy, World**

Questi tre controller sono le sole tre classi capaci di modificare i 3 modelli.
Dentro si trovano, quindi, dai setter/getter più semplici a metodi più complessi che gestiscono i dialoghi dei personaggi e i loro movimenti, le Sprite Animation e le Bitmap.

- Controller **Turns**

Se il GameLoop era il cuore dell'applicazione…, Turns ne è la mente.
Nella classe vi è tutta la divisione della storyline del gioco. Divisa per turni, è possibile identificare 3 macro sezioni:

1. Introduzione [Turni -6,-1]
   In questa fase c'è una sorta di Tutorial grazie al quale vengono spiegate al giocatore le meccaniche del gioco, sotto forma di racconto tra pirati, una sorta di background di Monkey Island e infine il giocatore è chiamato a scegliere la difficoltà della partita: il giocatore può decidere se utilizzare gli insulti originali di Monkey Island 1, Monkey Island 3, o entrambi i set di insulti.
2. Mappa e Duelli [Turni 0-19]
   Le sfide contro i pirati costituiscono l'essenza del gioco.
   Finita l'introduzione, al giocatore viene proposta la mappa di Melee Island. A lui il compito di scegliere, ogni volta, uno dei quattro pirati a disposizione.
   Una volta cliccato il pirata, la sequenza del duello viene interamente inizializzata: il dialogo iniziale, le Sprite di Guybrush e del pirata, lo sfondo e gli insulti/controInsulti imparati dal giocatore (inizialmente 2 per tipo).
   Lo scopo del giocatore è quindi imparare più insulti/controInsulti possibile, al fine di sconfiggere i pirati che, se raggiunto un buon progresso di apprendimento degli insulti, propongo la sfida finale contro il Maestro di Spada, Carla.
   A quel punto il '?' della mappa viene sostituito dalla faccia di Carla.
   Il duello contro il maestro di Spada è simile strutturalmente a quello contro i pirati, sebbene sia unidirezionale: Sarà infatti solo Carla a proporre i suoi insulti e il giocatore a rispondere.

Carla però utilizza degli insulti nuovi, mai usati nei precedenti scontri: il gioco prevede che ci sia un senso logico tra gli insulti di Carla e i controInsulti già appresi. Sta quindi al giocatore capire questo nesso e sconfiggere Carla.

3. Outro [Turno 20]
   Sconfitta Carla il gioco finisce. Viene proposto al giocatore un finale di chiusura e i titoli di coda.

L'introduzione e il finale con Carla e l'Outro sono stati implementati in sotto-cicli in metodi separati per non appesantire ulteriormente il ciclo principale, che -ricordo- viene sempre chiamato ogni 20ms circa.

Il duello contro il pirata consta di 13 step (esclusi i primi di dialogo introduttivo).
Alcuni step sono dedicati al controllo della temporizzazione dell'animazione, altri servono a gestire l'interazione lato utente, altri ancora per il calcolo casuale della risposta del pirata o dell'insulto da proporre a Guybrush.
Il primo che raggiunge 3 punti vince l'intero round e si torna alla mappa.
Lo scopo del giocatore non è battere il pirata, bensì imparare più insulti/controInsulti possibili.

- Model **Guybrush, Enemy, World**

I modelli contengono tutte le variabili utili alla caratterizzazione di Guybrush, dello sfidante e di ciò che gravita attorno (Sfondo, modalità ecc.).
Dentro vi sono quindi tutti i metodi get/set chiamabili dai rispettivi controller.

- **Dialogs**

Questa classe ha il compito di inizializzare tutte le scritte che compaiono durante il gioco.
Si è deciso di salvare nel file /res/value/string.xml tutte le stringhe dell'applicazione.
Per gestire insulti/controInsulti è stato utilizzato lo SparseArray<String>, una sorta di HashMap ottimizzato per Android che tiene in memoria coppie <Chiave,Stringa> ognuna col proprio indice.
Ogni insulto e rispettivo controInsulto ha la propria chiave univoca, che viene controllata al momento della verifica del controInsulto. Quindi se il pirata dice l'insulto 4 a Guybrush, il giocatore dovrà conoscere il controInsulto 4 ed eventualmente selezionarlo, per rispondere correttamente.

- **Music** e **SoundPlayer**

Per gestire la musica mp3 del gioco è stato utilizzato l'oggetto MediaPlayer di Android.media, che viene istanziato all'inizio del gioco, salvato il riferimento in World, ed eventualmente cambiato con la chiamata statica changeMusic() durante le varie fasi.

Per i suoni, invece, si è utilizzato l'oggetto SoundPool che carica in memoria i suoni del gioco e li richiama con l'apposita chiamata durante l'update() del turno.
Tutte le musiche e i suoni sono salvati in /res/raw

- Render **Guybrush, Enemy, World**

Queste sono le classi della View del MVC che, in base alle informazioni fornite dai rispettivi controller, disegnano a video animazioni e testi.
Le classi vengono costruite istanziando la classe **Text** e la **SpriteTile** Animation.
Le chiamate render() non sono altro che le chiamate *display_game()* menzionate in precedenza: inizialmente si occupano di calcolare il rettangolo di destinazione per la draw() delle sprite, per poi passare al disegno vero e proprio.

- SpriteTile

Questa è la classe che si occupa della preparazione delle animazioni.
Quando viene istanziata, viene passato come argomento il Reference ID della SpriteSheet (Bitmap contenente tutte le animazioni possibili) e l'xml associato contenente le coordinate x,y dei quattro vertici del frame.
L'xml viene quindi parsato creando *un'animationSequence* contenente tutti i dati utili per individuare il frame esatto nell'intera SpriteSheet.
La chiamata draw() recupera quindi *l'animationSequence* voluta (*rclip*) e la disegna nel rettangolo di destinazione (*destRect*). Al suo interno vi è anche una chiamata a funzione *update()* che ha il compito di iterare il disegno per ogni frame della *animationSequence.*
Per le Sprite ho fatto uso di una classe già implementata e si può trovare qui:
http://warriormill.com/2009/10/adroid-game-development-part-1-gameloop-sprites/

- Text
Classe adibita alla creazione del *DynamicLayout* contenente i testi.
Attraverso un timer interno, la chiamata *draw*() disegna il layout, precedentemente configurato con il testo da visualizzare e un *TextPaint* per lo stile.
E' stato utilizzato il *DynamicLayout* perché, in combinazione con l'oggetto *Editable* di Android, è possibile modificare il contenuto del Layout direttamente dal nostro GameLoop Thread a tempo di esecuzione.

3. Problematiche e Soluzioni

Realizzare un gioco su dispositivi Android che preveda animazioni "complesse" comporta un pesante uso della memoria Heap del dispositivo, soprattutto se si caricano Bitmap molto grosse come le SpriteSheet. Queste vengono decompresse in memoria e una sprite di 1024x1024x8 occupa già 8MB di Heap.
Android mette a disposizione soluzioni come le BitmapFactory, ma queste risultano utili soprattutto nelle applicazioni che devono far uso di Bitmap grosse ridimensionate ad anteprima.
Purtroppo le Sprite usate in questa applicazione, se rimpicciolite a misure come 1024x1024, perdevano molto in qualità; ho quindi preferito dividere un'iniziale SpriteSheet grande in tre più piccole (Talk, Fight e Lose) che vengono caricate nel turno appropriato.
Per favorire ulteriormente il riciclo della memoria, Android fornisce la chiamata al GC *recycle*() che provvede, appena ne ha la possibilità, a liberare l'oggetto dalla memoria.
Ogniqualvolta che viene caricata una nuova Sprite si tende a riciclare quella precedentemente usata.
Utilizzando questi accorgimenti l'applicazione risulta stabile anche su dispositivi più datati come, per esempio, l'HTC Tattoo (2009, 256 MB RAM, Android 2.3.3).
Consiglio quindi l'eventuale utilizzo di un framework come LibGdx che, attraverso l'uso di OpenGL ES/WebGL, estrae l'implementazione a basso livello della singola visualizzazione di un frame.

Problema interessante è stato visualizzare le immagini in proporzione al device in uso.
E' stato risolto calcolando con software di Image Editing le proporzioni tra i vari vertici del rettangolo di disegno e la dimensione della Canvas in uso.
E' quindi visibile nei Render o nella creazione dei testi, dimensioni relative a *canvas.getWidth*() o *canvas.getHeight().*

Infine ho cercato di risolvere tutti i problemi legati alla diversa gestione di Android nel caso l'applicazione vada in background.
Su device più performanti l'app rimane in memoria in stato di Pausa e questo permette di ripristinare l'effettivo stato di gioco lasciato in sospeso; se però Android richiede memoria per altri usi l'applicazione viene messa in stato di Stop. E' stato verificato che in alcuni dispositivi l'applicazione viene automaticamente messa in stato di Stop se messa in background.
Il caso di stop, più delicato, è quindi stato studiato con maggiore attenzione:
se il gioco è ancora nell'introduzione (non si è ancora arrivati alla mappa dell'isola) il gioco viene "resettato" e si ricomincia dall'inizio; se il giocatore ha già sfidato pirati e appreso insulti, vengono salvati tutti quei parametri utili a un ripristino adeguato dello stato, come il TurnDialog globale, il numero di vittorie conseguite e gli SparseArray<String> degli insulti/ControInsulti.
Purtroppo non è possibile inserire l'intero SparseArray<String> nel Bundle, che accetta solo SparseArray<T extends Parcelable> e String non è Parcelable.
Si è così risolto estrapolando l'array delle chiavi Key degli insulti, che, passato come int[] al Bundle, viene successivamente usato da appositi metodi per ricostruire l'intero SparseArray<String>.

Ultimo problema ostico è stato trovare un modo poco oneroso di risorse per visualizzare i discorsi dei pirati. Inizialmente si era pensato di usare delle semplici TextView, ma queste risultavano difficilmente modificabili a runTime da un Thread fuori dal UI-Thread e si è quindi optato per utilizzare i TextLayout e la loro chiamata draw(Canvas canvas) che, preso un TextLayout, un TextPaint e una canvas, ne disegnano il contenuto testuale.

4. Idee per Estensioni

Il gioco in sé non lascia molte aperture a sviluppi di funzionalità, anche perché si andrebbe a modificare il meccanismo del gioco storico della LucasArts.

E’ quindi preferibile rivederne l’implementazione con librerie grafiche più complete come OpenGL o Libgdx per estenderne la compatibilità su altre piattaforme mobile (LibGdx promette compatibilità iOS, WP e HTML5) e per eliminare eventuali problemi di performance utilizzando un livello più alto di progettazione.

Purtroppo l’utilizzo di animazioni, fondali e meccanismi originali di Monkey Island non mi permettono l’inserimento dell’app nello Store di Google. E’ quindi immaginabile una modifica al comparto grafico inserendo SpriteSheet con licenza libera o disegni fatti “a mano” per evitare il conflitto con le licenze.

5. Conclusioni

Realizzare questo gioco su Android mi ha dato la possibilità di esplorare contemporaneamente due ambiti molto interessanti: quello mobile con Android e quello videoludico.
Anche se questa scelta si è poi configurata in un maggiore sforzo di progettazione e implementazione, realizzare un videogioco, seppur in dimensione più ristretta, mi ha fornito l’occasione per cominciare a studiare meccanismi e tecniche che esulano dal piano didattico della triennale di Informatica di Bologna, che non prevede corsi di studio sul Game Design o sulla Grafica.
Contemporaneamente studiare la piattaforma Android è stata un’ottima opportunità per comprenderne modalità e potenza di linguaggio, per quella che sembra divenire anno dopo anno, la piattaforma mobile del futuro.