Progetto Mobile Programming and Mulimedia

Casagrande Marco - Matricola 1049532

18 gennaio 2018

1 Premessa ed info utili per gli studenti (non originariamente nella relazione, ovviamente)

Potete trovare il codice ampiamente commentato nel file scenes/s1d11.lua
Mi dispiace per il codice disordinato, ma quando ho capito come migliorarlo ormai era già quasi finito lo sviluppo.
Usate il cosiddetto ultimate config.lua (quello che uso pure io), per garantire una risoluzione corretta per ogni device.
Io ho usato loadsave.lua per ottenere salvataggi permanenti nello smartphone, altrimenti ad ogni avvio si azzerava tutto.
Cosa vuole sopra ogni cosa la professoressa Gaggi: rispetto delle best practice studiate a lezione ed analisi approfondita del framework.
Cosa non le è piaciuto di questo progetto: ha accennato alla scarsa difficoltà, ha contestato la frecca per scorrere a destra nella selezione dello scenario (in quanto non porta da alcuna parte), ha contestato la barra della vita che non mostra la vita massima originaria mentre si accorcia, ha contestato il mio non scrivere nella relazione di queste ed altre scelte volontarie che ha dovuto chiedermi poi a voce.
Comunque restano solo piccole osservazioni, il progetto è stato valutato molto positivamente quindi non c'è da preoccuparsi.
Consiglio mio: perdere meno tempo con le menate della grafica e del sonoro che ho fatto da solo oppure cercato con le opportune licenze. Anche la complessità del gioco cambia poco la valutazione.

2 Introduzione

Il progetto consiste nello sviluppo di un'applicazione per Android tramite il framework cross-platform Corona SDK. L'applicazione consiste in un gioco single-player dal nome Walpurgisnacht. La build consiste in un file con estensione apk per Android.

3 Relazione sul progetto

3.1 Panoramica del gioco

Walpurgisnacht è un gioco 2D strategico a turni, basato sulla fisica. La trama parla della notte di Valspurga, nella quale le streghe rinnovano il patto col Diavolo e cercano vittime da offrire.
Lo scopo del giocatore è difendere da streghe e famigli le vittime addormentate. Il giocatore dovrà affrontare svariati livelli con difficoltà crescente, sbloccando man mano nuove meccaniche. Sono presenti dei tutorial per introdurre il giocatore alle meccaniche, lasciando però spazio alla scoperta personale. I menù permettono di navigare per gli scenari, che coinvolgono personaggi differenti con caratteristiche differenti, ed i singoli livelli (sogni). Alcune impostazioni come musica ed effetti sonori possono essere modificate e restano salvate negli avvii successivi.
Lo scopo dei livelli è l'eliminazione dei nemici che appariranno secondo logiche predeterminate, impedendo loro di esaurire la barra della vita del giocatore. I nemici possiedono semplici comportamenti predefiniti. Il giocatore avrà a sua disposizione sei differenti attacchi, che proseguendo nel gioco potranno essere personalizzati secondo le proprie preferenze e le circostanze attuali del combattimento. Il livello terminerà o con la sconfitta di tutti i nemici, o con la sconfitta del giocatore. Ogni livello superato permetterà permanentemente l'accesso al livello successivo. Al termine dei

livelli di uno scenario, si potrà passare allo scenario successivo.
Attualmente è disponibile solo 1 scenario composto da 15 livelli, l'ultimo dei quali vuoto e privo di boss finale.

3.2 Panoramica degli strumenti utilizzati

La scelta riguardo al framework è ricaduta su Corona SDK. Oltre ad essere specializzato nello sviluppo di giochi, fornendo una propria libreria per la fisica, è un software leggero ed intuitivo. La documentazione e community sono punti forti rispetto ad altri framework più di nicchia. Il simulatore real-time fornito con Corona SDK è senz'altro l'attrattiva principale del software, nonchè la componente che più ho apprezzato fin da subito. Il tempo per iniziare a sviluppare a partire dalla visita al sito è stato quasi nullo (2 minuti netti) e senza ulteriori complicazioni. L'idea precedente era stata Cocos2D ma la frustrazione nel peso del software e nella sua complessità di installazione e configurazione mi hanno portato ad abbandonarlo (dopo un paio d'ore).
Come editor di testo per la programmazione con il linguaggio Lua ho utilizzato Atom, con tanto di autocomplete plugin per Corona SDK.
Come editor di file audio ho utilizzato Audacity, genericamente per effettuare tagli ed apportare piccole modifiche ai file originali (solo nel caso la licenza lo permettesse).
Come editor grafico ho utilizzato Gimp. Per creare spritesheet invece ho utilizzato TexturePacker-GUI, comodo per risparmiare tempo nel posizionamento ordinato delle singole immagini.

3.3 Componenti del gioco

3.3.1 Sonoro

Il comparto sonoro è composto da audio recuperato da appositi archivi online, rispettando le policy di attribuzione richieste dall'autore. Volume totale ed effetti sonori possono essere mutati/attivati sia dalla schermata iniziale che all'interno dei livelli. Le preferenze del giocatore vengono salvate per gli avvii successivi.

3.3.2 Sonoro e Corona SDK

Il comparto sonoro è stato integrato nell'applicazione tramite la libreria *audio* di Corona SDK, distinguendo tra *audio.loadStream()* per le musiche (in loop) ed *audio.loadSound()* per gli effetti sonori "monouso". Il channel 1 è stato riservato alle musiche, mentre gli altri sono stati lasciati liberi per gli effetti. Questa suddivisione ha permesso un facile sistema di azzeramento del volume delle sole musiche o dell'audio complessivo del gioco.
La libreria *audio* si è mostrata all'altezza di ogni mia necessità, tranne in due ambiti:

- riproduzione audio di musica del giocatore;
- caricamento dell'audio da parte di Corona Simulator.

Il primo problema è stato l'impossibilità di permettere al giocatore la scelta e l'ascolto di uno dei suoi brani musicali durante il gioco. L'unica soluzione che può adottare è far partire la riproduzione del suo audio e poi avviare il gioco. Se ciò non funzionasse, dovrebbe aprire il gioco, relegarlo a finestra far partire il proprio brano musicale, e poi ritornare al gioco. Il codice per implementare tale funzionalità è stato:

```
if audio.supportsSessionProperty then
  audio.setSessionProperty( audio.MixMode, audio.AmbientMixMode )
end
```

Il secondo problema si è rivelato infinitamente più fastidioso, anche se affligge solo i developer. Molto spesso infatti i caricamenti dei file audio mi hanno causato warning su Corona Simulator e non vengono conseguentemente emessi durante le simulazioni. Considerato il caricamento di cinque file audio su praticamente ogni pagina, ad ogni manciata di refresh dell'applicazione in seguito ad aggiornamenti compariva almeno uno warning. Inoltre, dopo altri refresh aumentava la probabilità che il software smettesse di funzionare e dovesse essere manualmente chiuso e riavviato. Questo comportamento è avvenuto sia prima dell'aggiornamento di fine dicembre, che dopo. Per evitare di dover riavviare Corona Simulator ogni 5-10 minuti, ho dovuto sviluppare il gioco interamente senza suoni ed abilitarli solo alla fine (il problema avviene proprio nel caricamento dell'audio, quindi tutto ciò accade anche con volume zero).

3.3.3 Grafica

La grafica del gioco è composta per la quasi totalità da immagini create da me (si può notarne la "qualità"). Alcune sono organizzate in spritesheet poichè appartengono allo stesso gruppo logico (tutti gli attacchi, tutti i potenziamenti) mentre altre poichè fanno parte di una stessa animazione (la freccia rossa nei tutorial). Non ho provveduto a fornire più versioni delle stesse immagini a diverse risoluzioni per migliorarne dinamicamente la qualità dato che avrebbe richiesto troppo lavoro (già ci ho messo intere ore a realizzare questa grafica passabile).

3.3.4 Grafica e Corona SDK

Anche riguardo alla libreria *graphics* ho qualche piccola annotazione di enorme frustrazione.
Il primo problema è stato il constatare che alcuni oggetti non vengono cancellati automaticamente quando si cambia scena. In particolare oggetti geometrici e testo, dunque il contatore dei nemici rimanenti e dei danni devono sempre essere cancellati manualmente nelle funzioni per chiudere il livello. Forse mi sfugge qualche scenario dove ciò sia vitale, ma visto che tutto il resto viene cancellato automaticamente, non vedo perchè *display.newText()*, *display.newRect()* e simili non dovrebbero subire lo stesso trattamento.
Il secondo problema si è presentato nella rilevazione della dimensione degli spritesheet. E' capitato solo con la freccia rossa nei tutorial e non con altre animazioni (non presenti nel gioco ma che ho testato). Lo spritesheet era stato correttamente diviso in 6 immagini delle stesse dimensioni (generato più volte e calcolato anche a mano per sicurezza), tuttavia nell'animazione la parte sinistra della freccia veniva mangiata dal frame precedente, mentre la punta finiva nel frame successivo, come se fosse presente un offset iniziale a spostare tutto a destra. Non pretendo di essere infallibile, comunque dopo mezzora di tentativi ho rinunciato ed ho semplicemente aggiunto spazio vuoto. In questo modo, errore di calcolo di Corona SDK o no, l'area mangiata è semplicemente spazio vuoto. Continuo a non capire quale sia stato il problema visto che ho agito esattamente come con altri spritesheet (stesso identico codice, funzionante) e con molteplici approcci (sia dando solo le dimensioni della prima immagine lasciando al software il compito di ripetere la suddivisione altre 5 volte, sia definendo nella dichiarazione i pixel iniziali e finali di ogni immagine).
Il terzo problema è stato la difficoltà del posizionare correttamente gli oggetti grafici nello spazio perchè non si hanno coordinate di riferimento. L'unica soluzione è andare a tentativi finchè il posizionamento dell'oggetto piace (stessa cosa per le shape di corpi fisici). Ovviamente il sistema di coordinate è corretto e consistente, ma avrei preferito non dover refreshare 10 volte per azzeccare il pixel desiderato oppure calcolarmi la distanza su un foglio a parte. La semplice aggiunta di una modalità per la suddivisione in quadrati e la visualizzazione delle coordinate sotto al mouse risparmierebbe al developer non poco tempo.
Il quarto problema penso fosse uno strano bug di Atom, ma lo includo dato che tecnicamente riguarda un problema con la grafica. Per la visualizzazione della trama nella scena di introduzione, volevo creare uno scrolling box usando l'oggetto *native.newTextBox()*. Copiando il codice fornito nella documentazione ed inserendo i cambiamenti consigliati per la parte "scrolling" ho provato a testare il risultato. Corona Simulator ha immediatamente smesso di funzionare mentre Atom si è bloccato per 30 secondi. Nel task manager ho notato l'uso di Disco e Memoria al 100% e solo all'effettiva chiusura del processo di Atom tali valori si sono abbassati. Ho riprovato una seconda volta, modificando i parametri del *native.newTextBox()* che ritenevo presumibilmente erronei, ottenendo però lo stesso risultato. Incuriosito e preoccupato dagli eventi, non ho purtroppo trovato corrispondenze precise ed aggiornate con questo comportamento di Atom. Dunque mi sono appuntato mentalmente di non usare mai più un oggetto *native.newTextBox()*.

3.3.5 Menù

I menù sono composti meramente da pulsanti ed immagini. Nella selezione dello scenario si può fare uso della gesture di swipe per passare tra scenari 1-2-3 e 4-5-6.

3.3.6 Menù e Corona SDK

Riguardo alla libreria *widget*, sono rimasto abbastanza deluso. Le funzioni per la creazione di pulsanti sono inutili: non offrono alcuna impostazione di default e necessitano di troppi parametri. Ho risparmiato tempo creandomi una personale funzione di creazione del pulsante al posto di utilizzare *widget.newButton()* nonostante io abbia consultato la documentazione a riguardo. Creare

a modo proprio i pulsanti è l'approccio più istruttivo, veloce e personalizzabile. Questo è un banale esempio tratto dal gioco:

```
local function createButton( text, xpos, ypos, xsize, ysize, fontsize )
      local button = display.newImageRect(
       t_groups.superGroup, "images/graphics/menu_button.png", xsize, ysize
      )
      button.x = xpos
      button.y = ypos
      local buttontext = display.newText(
       t_groups.superGroup, text, button.x, button.y, "GosmickSans.ttf", fontsize
      )
      buttontext:setFillColor( 0, 0, 0 )
      return button, buttontext
    end
```

3.3.7 Fisica

La fisica interviene raramente nel gioco: solo per attacchi che devono compiere una traiettoria ad arco (mele, toast) e per le collisioni. Almeno metà dei corpi fisici sono statici oppure non affetti da gravità.

3.3.8 Fisica e Corona SDK

Riguardo alla libreria *physics* non ho alcuna critica. Direi sia un po' triste l'obbligatorietà di sfruttare la fisica per ottenere traiettorie curve, perchè l'alternativa è dannarsi con avanzate formule matematiche e/o il "Bezier method". La fisica si conferma dunque uno dei punti di forza di Corona SDK.

3.3.9 Scene

Ogni schermata è trattata come una scena separata dalle altre. La loro gestione è semplice: se la schermata può variare (schermata iniziale dove viene aggiunto il pulsante intro, selezione livello dove ne vengono sbloccati di nuovi) essa viene cancellata prima che vi si acceda.

3.3.10 Scene e Corona SDK

Il composer offerto da Corona SDK è un metodo molto intuitivo e si è dimostrato efficace nella gestione delle scene. La difficoltà principale risiedeva nel distinguere quali riferimenti ad oggetti fossero effettivamente indicizzati in determinato momento della scena (create, show, hide, destroy) e fase (will, did). Non ho apprezzato il tutorial all'indirizzo https://docs.coronalabs.com/guide/programming/01/index.html che prima mostra come creare il gioco e poi come convertirlo tramite composer. Seguendo tale ordine ed indicazioni, mi sono ritrovato con un codice complesso da convertire. Nel linguaggio Lua non è possibile dichiarare più di 200 variabili esterne a funzioni. Per aggirare questa restrizione dunque ho dovuto convertire ogni dichiarazione di variabile:

```
local backGroup
local charactersGroup
local bulletsGroup
local choicesGroup
local uiGroup
local superGroup
```

racchiudendola in una table che specificasse un gruppo logico di appartenenza:

```
local t_groups = {
  backGroup, charactersGroup, bulletsGroup, choicesGroup, uiGroup, superGroup
}
```

Purtroppo visto l'alto numero di variabili necessarie alla gestione del combattimento, non ho trovato alternative a questo metodo.

3.3.11 Codice

Il linguaggio di programmazione Lua si è rivelato estremamente semplice da capire ed intuitivo. Nonostante avrei apprezzato un linguaggio con qualche feature aggiuntiva, le applicazioni difficilmente raggiungono livelli di complessità avanzati. L'adozione del linguaggio Lua è stata una scelta migliore rispetto a C#, usato da altri framework, vista l'enfasi posta sulla semplicità da parte di Corona SDK.

3.3.12 Codice e Corona SDK

Manca un sistema per il salvataggio permanente di dati. Gli utenti hanno sviluppato diverse modalità e librerie per ovviare a questo problema. Io ho optato per la soluzione proposta all'indirizzo https://github.com/robmiracle/Simple-Table-Load-Save-Functions-for-Corona-SDK, che effettua caricamenti e salvataggi di dati tramite file con estensione json.
Una problematica del linguaggio Lua non sufficientemente esposta (di cui mi sono accorto per caso) è la mancanza di randomizzazione dei primi 1-2 risultati della funzione *math.random()*. Nel mio caso il primo numero era sempre lo stesso (nonostante l'uso di *math.randomseed(os.time())*). La soluzione consigliata da altri utenti è quella di "svuotare" i primi numeri generati all'avvio dell'applicazione, senza utilizzarli. I valori successivi saranno correttamente casuali.
Non ho apprezzato molto il plugin dell'autocomplete di Atom per Corona SDK. Genericamente la finestra compariva sopra a zone che avrei voluto selezionare, comportando movimenti aggiuntivi fastidiosi. Inoltre i suggerimenti erano o troppo vaghi, costringendomi a cercare nella documentazione online, o superflui. Ho apprezzato il loro sforzo ma avrei fatto meglio a disabilitarlo subito e non pensarci più.

3.4 Valutazione finale su Corona SDK

Corona SDK si riconferma un framework eccezionale per sotto certi aspetti:

- leggero;
- intuitivo;
- ben strutturato;
- essenzialmente free con poche limitazioni;
- ben documentato;
- buona community;
- simulatore compreso nel pacchetto;
- fisica immediata ed eccellente;
- linguaggio Lua estremamente semplice;
- frequentemente aggiornato.

Ma presenta lacune che oscillano tra enormi e minime (in ordine di importanza):

- audio che fa crashare Corona Simulator ogni 10 minuti;
- difficoltà nel collocare oggetti alle giuste coordinate;
- documentazione precisa solo per le conoscenze di base, ma deludente in profondità;
- salvataggio di dati permanenti non integrato direttamente da Corona SDK ma tramite soluzioni utente;
- linguaggio Lua che manca di funzionalità presenti in altri linguaggi più avanzati;
- limitazioni tecnologiche (derivate dall'approccio non nativo).

4 Conclusioni

Corona SDK mi ha sorpreso, sia nel bene che nel male. Sicuramente la possibilità di utilizzare Corona Simulator lo eleva sopra a tanti altri framework, specialmente nell'ambito dei giochi 2D. Le pecche che presenta non sono imperdonabili, quindi il mio giudizio è complessivamente positivo. Ritengo Corona SDK un framework indirizzato ad un target un po' differente dalle mie competenze. In più occasioni avrei preferito maggior complessità in cambio di maggior flessibilità ed autorità. Lo consiglierei caldamente a principianti nel settore, mentre per gli esperti risulterà utile solo per i loro primi giochi. Applicazioni più complesse richiederanno troppi espedienti per compensare alle mancanze del linguaggio Lua e di Corona SDK. In particolare il desiderio di modalità multiplayer e/o grafica 3D.
Corona SDK è il framework ideale per sviluppare giochi dai concetti semplici ed in poco tempo, senza rinunciare all'immediata gratificazione del visualizzare i proprio progressi in tempo reale tramite Corona Simulator.