Progetto di Mobile Programming

Chicken Invader

Anno 2017-2018

Relazione di Matteo Depascale

1) Introduzione

Questa applicazione riguarda principalmente una navicella che spara a dei polli, questi ultimi sparano uova che devono essere schivate dalla navicella. La navicella inoltre deve schivare o distruggere degli asteroidi. L'idea è stata presa da un vecchio gioco per il computer, esso è a sua volta un remake del famoso *Space Invader.*

2) Funzionalità

Nel gioco sono presenti le seguenti funzionalità:

- La navicella può essere trascinata in orizzontale
- La navicella spara dei laser dopo ogni certo intervallo
- I polli possono sparare uova
- La navicella può sparare un razzo che colpisce molteplici bersagli, sia polli che asteroidi
- Quando un pollo muore, c'è una probabilità di 1/27 che scenda un power up, questo consente di aumentare il rateo di fuoco di 50 (inizialmente settato a 700) ed è possibile raccogliere fino a 3 power up
- Shakerando il cellullare è possibile eliminare la prima e l'ultima colonna di polli
- Sia lo shaker che il razzo possono essere utilizzati una volta ciascuno
- Possono avvenire collisioni con: navicella, laser, polli, asteroidi, pacchetti regalo, razzi
- Data la banalità del gioco, il tutorial si tratta semplicemente di alcuni suggerimenti random sotto il livello raggiunto
- L'azione di *Back* che ritorna indietro al menu mentre si sta giocando

3) Schema codice

Il codice prodotto è in forma procedurale, pertanto non vengono create classi o scene diverse per ogni livello.

All'inizio ci troviamo nel menu, cliccando su *Gioca* o su *Punteggi Migliori* si cambia alla rispettiva scena.

Al livello iniziale la chiamata a *createChicken* e successivamente a *createEgg* consentono di iniziare il gioco, il livello proseguirà finche i polli inseriti nella propria tabella non saranno finiti e verrà invocata *newLevelText* che a sua volta, dopo aver mostrato la scritta del livello successivo e del suggerimento con *hintText*, chiamerà *nextLevel* che si occuperà di passare al livello successivo.

Dopo aver completato il livello 2 verrà invocata la funzione *gameLoop* che si occupa di creare il terzo livello degli asteroidi e poi, richiamata dalla stessa, passerà al livello 4. Per trovare il giusto tempo sono state fatte diverse prove e ho dovuto creare delle funzioni locali interne come *gameLoopTimerAsteroid* il cui unico scopo fosse quello di ritardare l'esecuzione della funzione per permettere di arrivare alla fine del gioco con i tempi corretti.

Utilizzando una struttura procedurale spesso ho dovuto ricorrere a delle booleane come *done* che servono a far capire quando abbiamo finito di giocare e non c'è più bisogno di invocare nuovi oggetti (polli o asteroidi). Sono state create anche funzioni come *pauseLaser* o *resumeLaser* con lo scopo di ritardare il timer dell'azione di pausa o ripresa dell'evento.

4) Interfaccia grafica

L'interfaccia sfrutta la suddivisione di Corona SDK, perciò viene suddivisa in 3 gruppi: *backGroup*, *mainGroup*, *uiGroup*. Questi tre gruppi poi vengono inseriti nel gruppo della scena in modo da essere invocati e poi eliminati più facilmente.

Le immagini non vengono ritagliate alla perfezione perciò si può notare come la navicella che, a contatto con l'uovo, non viene colpita fino a che l'uovo non raggiunge una certa posizione (*radius=30*).

5) Audio

L'audio viene offerto per la maggior parte da Eric Matyas attraverso il proprio sito http://soundimage.org. Viene creato l'audio per il background, durante il gioco e all'esterno del gioco mentre si è nel menu e nei punteggi migliori, un audio per il laser, power up, il razzo e la propria esplosione, lo shake e per quando la navicella muore. Non sono riuscito a trovare un audio non fastidioso per le collisioni con i polli o con gli asteroidi perciò ho preferito non metterlo.

Si vuole far notare l'utilizzo di *audio.loadStream* per la musica di background e *audio.loadSound* per gli effetti audio.

6) Punteggi Migliori

Per salvarle i punteggi migliori fatti dall'utente viene utilizzata la libreria di *json* salvando i risultati sul file *score.json*.

Con la funzione *loadScore* apro il file in modalità "r" ovvero read only, mentre con la funzione *saveScore* apro il file in modalità "w" ovvero write only.

7) Commenti su Corona SDK

Vantaggi:

- ✓ Il linguaggio *Lua* è molto semplice
- ✓ La documentazione è ottima
- ✓ Libreria *physics* è facile da utilizzare
- ✓ L'interfaccia è davvero semplice da creare
- ✓ Gestire gli eventi è estremamente intuibile
- ✓ Community sempre a disposizione
- ✓ Le scene sono una grossa semplificazione

Svantaggi:

- × In molte occasioni ho trovato errori che il simulatore non riusciva a trovare
- × La documentazione non entra nei dettagli di alcune funzionalità che a mio parere richiederebbero più spiegazioni
- × Pecca di alcune funzionalità che in alcuni linguaggi sono essenziali, come i prototipi di funzione
- × Alcune volte gli eventi non vengono rilevati nel corretto modo, come ad esempio le collisioni con un oggetto troppo veloce
- × Non è possibile creare oggetti durante una collisione

Conclusione:

Tutto sommato Corona SDK è stata una buona scelta per il tipo di applicazione creata, se invece dovrò creare un'applicazione più complessa probabilmente preferirò un altro framework invece di Corona SDK. Lo consiglio per creare applicazioni semplici in quanto è estremamente intuitivo.