PLAStiblob

Bernardi Lorenzo  
Bortolamiol Alessandro  
Dal Colle Gabriele  
Scognamiglio Simone

# Introduzione

Per questo progetto ci siamo affidati ad un’idea che avevamo avuto durante l’anno: quella di riuscire a coniugare un gioco dall’aspetto animato e divertente con un tema attuale e molto importante come la natura. Plastiblob è anche questo: è un progetto di cui andiamo molto fieri, soprattutto perché perché contiene molta passione.   
Speriamo che possa piacerle almeno un quarto di quanto piace a noi e speriamo riesca ad arrivare al livello finale, per gustare una piccola sorpresa.

Il progetto è stato creato e modificato online grazie alla piattaforma GitHub, ognuno di noi infatti poteva eseguire la sua modifica al progetto e renderla disponibile a tutti gli altri membri del gruppo. Tale scelta è stata presa per aver una maggiore flessibilità in termini di modifica del progetto ma anche di tempo.

Tutto il progetto è stato pensato per uno schermo **1280 x 720**, infatti le immagini di background sono di questa misura. All’interno del file di config del progetto, abbiamo però inserito tale riga:

scale = "zoomEven",

che andrà a ridimensionare gli elementi nello schermo in base alla grandezza del display del cellulare di chi sta utilizzando il gioco.

# Files del progetto

## menu.lua

Menu è la prima pagina su cui l’utente approda subito dopo aver cliccato sull’avvia applicazione. Questa semplice pagina presenta:

* Il logo del nostro gioco
* Un bottone collegato ad un listener per il suo relativo click
* Il logo dell’università di Udine

Notiamo subito che il bottone è rappresentato da un animazione sprite. All’interno di questo file abbiamo anche caricato una musichetta di sottofondo che rappresenta il tema portante del gioco.

## menu-levels

Questa pagina è un po’ più complicata della precedente, infatti contiene 4 immagini che permettono all’utente di scegliere il livello in cui andare. Per memorizzare i livelli sbloccati dall’utente abbiamo usato la libreria di corona che permette di interfacciarsi con sqlite3. Tramite questa implementazione siamo stati capaci di controllare e memorizzare dati utili a noi come il numero di liveli completati dall’utente e i punteggi ottenuti in ogni singolo livello.  
Se l’utente si interfaccia per la prima volta con l’applicazione, creeremo questa semplice tabella all’interno di un db ‘data’

local tableSetup = [[CREATE TABLE levels ( ID INTEGER PRIMARY KEY autoincrement, level, scoreLevel1, scoreLevel2, scoreLevel3, scoreLevel4);]]

la tabelle andrà a contenere un campo primario ID che conterrà un numero univoco per user. Ovviamente questo è superfluo nel nostro caso, infatti il db non si interfaccia con utenza online ma rimane in locale e quindi ci sarà un’unica riga all’interno di questa tabella. Oltre al campo di ID, abbiamo ‘level’ che conterrà un numero intero in base al livello che si deve affrontare: per esempio se è la prima volta che l’utente entrà nel gioco sarà 1, se ho correttamente passato il livello 1 aumenterà a 2. Gli altri campi possiedono al loro interno i vari punteggi ottenuti nei livelli, tali punteggi aumentano in base agli oggetti di plastica raccolti durante il percorso del blob.  
Dato che l’obiettivo del gioco è quello di **raccogliere più plastica possibile,** se l’utente arriverà a raccogliere il numero massimo di oggetti di plastica verrà premiato con 3 stelle, sennò a scendere con 2 stelle o 1 stella.

Il funzionamento di questo file lua non è immediato: abbiamo infatti 4 immagini distinte da cliccare per andare in ogni singolo livello: una per il livello 1, una per il livello 2, una per il livello 3 e una per il 4.   
L’utente, però, deve aver superato il livello precedente per passare al successivo e quindi bisogna rendere non cliccabile la foto del numero del livello che non è disponibile allo user. Per fare ciò abbiamo controllato il campo ‘level’ all’interno del db, e abbiamo reso disponibile al touch solamente i livelli <= al campo del database.   
Per esempio: nel caso il campo del db sia 2 🡪 è possibile andare a giocare all’interno dei livelli 1 e 2.

I livelli non disponibili andranno marcati usando una foto diversa con il lucchetto

## level1

Il primo livello è quello più semplice di tutti. Al suo interno avviene uno scrolling del background infinito e ogni tot secondi vengono richiamati dei nemici grazie a funzioni simili a questa qui sotto

gameLoop = timer.performWithDelay( time\_speed\_min, loop, 0 ) --loop del gioco in cui fa muovere gli sprite

Questi timer infatti andranno a creare e mettere in loop il movimento dei nemici che piano piano scrolleranno verso la parte sinistra dello schermo. Durante il gioco, la velocità di scorrimento dei nemici andrà ad aumentarsi fino ad arrivare ad un determinato massimo imposto da noi. La bravura dell’utente sarà quella di riuscire a saltare ogni singolo nemico e allo stesso tempo riuscire a raccogliere quanta più plastica possibile in modo da ripulire il mondo da una situazione a tratti post-catastrofica.  
La durata del livello è decisa a priori grazie ad una variabile “TimeToPlay”, tale variabile viene valorizzata all’entrata della scena.  
All’interno di quest’ultima abbiamo inserito una funzione per il conteggio dei secondi che andrà a comparare ‘timePlayed’ (variabile che ogni secondo si incrementa di 1) con ‘TimeToPlay’: se tali risultano uguali allora l’utente avrà passato il livello e il suo punteggio verrà salvato all’interno del database e il livello 2 verrà reso disponible ad essere giocato.

### menu-levels.lua

Questo file è la landing page del menu dove l’utente arriverà appena cliccato il bottone ‘play’ nella schermata di home. Questa pagina ha il dovere di controllare a che livello è arrivato l’utente all’interno del gioco tramite un controllo del database *Sqlite3.*   
Prima di tutti, quindi, controlliamo se è la prima volta che l’utente entra nel gioco in modo da sapere se bisogna creare un nuovo database o dovremmo utilizzare uno già esistente. Questo si farà con questa istruzione:

local checkifdbexists = [[SELECT \* from levels]]

    local dbexists = db:exec( checkifdbexists )

    print(dbexists)

Il print potrà ritornare due valori:

* **0:** l’esecuzione della query è andata a buon fine, ciò significa che la tabella è già istanziato all’interno della memoria e non serve crearlo
* **1:** in questo caso, la query non è andata a buon fine. Ciò significa che la tabella *levels* non esiste in memoria, andrà quindi creato.

Questa parte di codice invece è il fulcro della ppagina, infatti costituisce gran parte del funzionamento di essa. All’interno della variabile *livellicompletati* ci sarà il numero dei livelli completati dall’utente che è venuto precedentemente preso dalla tabella levels del database. In base a questo numero renderò disponibili al click o meno le varie immagini del display del cellulare.

* local levels={}
* for i=1, 8 do
* local impath
* --controllo se ho già passato il livello nell'identificativo su cui è posizionata 1
* if (tonumber(livellicompletati) >= i) then
* --assegno al percorso dell'immagine l'immagine corrispondente al livello in modalità SBLOCCATA
* impath = "immagini/menu/livelli/"..i..".png"
* else
* --assegno al percorso dell'immagine l'immagine corrispondente al livello in modalità BLOCCATA
* impath = "immagini/menu/livelli/locked"..i..".png"
* end
* print("percorso: ", impath)
* levels[i] = display.newImageRect( menu, impath, 200, 200 )
* levels[i].anchorX = 0
* levels[i].anchorY = 0
* levels[i].x = checkImagePositionX(i)
* levels[i].y = checkImagePositionY(i)
* end