**Mobile Programming**

**Data:** Roma 4/09/2017

**Nome del Gruppo:** Unicorni in Bicicletta

**Componenti del Gruppo:** Andrea Carrarini, Diletta Lagomarsini, Manuel Manzara

**Nome Applicazione:** *Space Battle*

**Tematiche affrontate**:

* **Introduzione.**
* **Attivazione, Ricerca e Accoppiamento dei dispositivi Bluetooth vicini.**
* **Drag and Drop delle navicelle spaziali sul campo di battaglia.**
* **Scambio dati Bluetooth tra dispositivi (l’effettiva partita).**
* **Gestione e riproduzione di contenuti audio.**

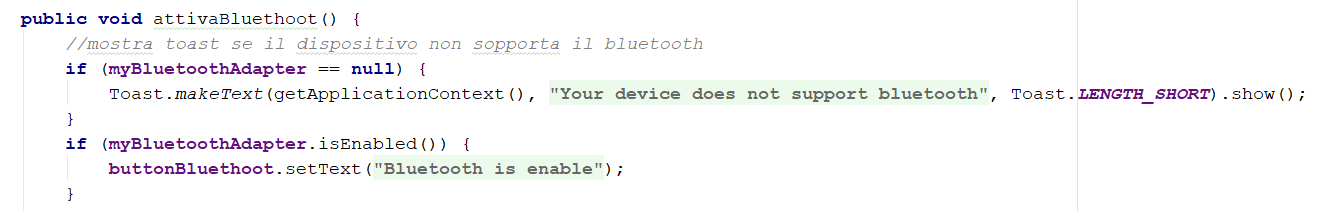
**Introduzione**

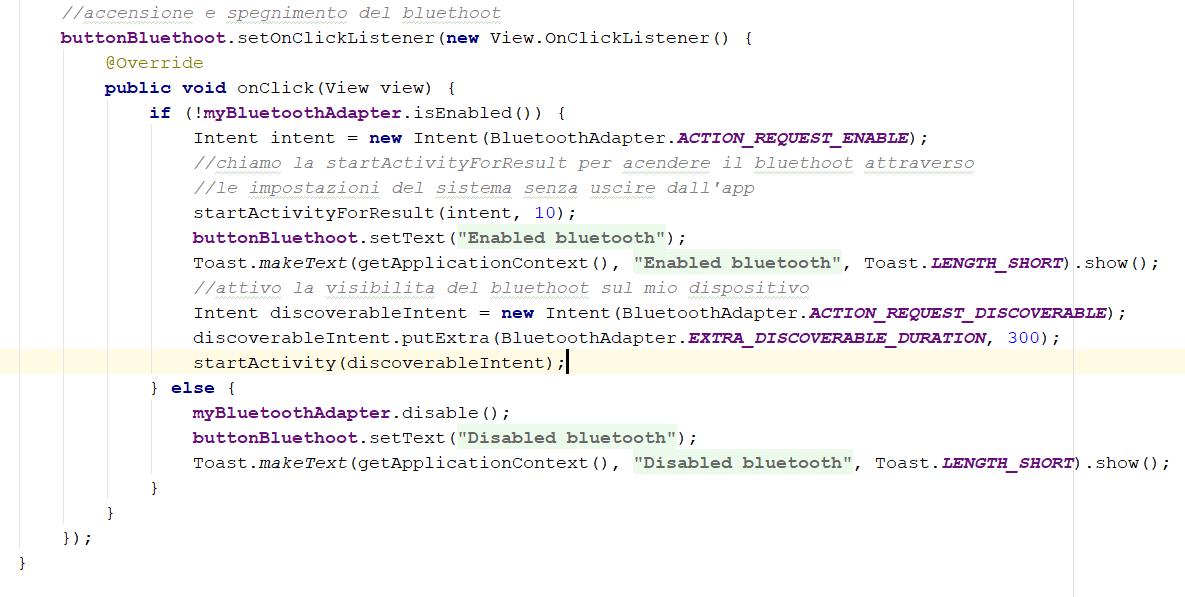
“Space Battle”, come già si capisce dal nome, è un’applicazione Android che permette a due utenti di sfidarsi l’uno contro l’altro ad una versione più spaziale del gioco battaglia navale. Non vedremo infatti, la classica battaglia navale con navi pirata e galeoni della marina, bensì una versione più innovativa, ispirata al tema di Star Wars!

L’applicazione si presenterà agli occhi dell’utente con una semplice schermata intuitiva e User Friendly composta da due bottoni: uno per l’attivazione del Bluetooth e uno per iniziare una nuova partita. Parliamo di Bluetooth, perché la nostra applicazione sarà basata principalmente su quest’ultimo, permettendo lo scambio di quelle informazioni necessarie a far procedere la partita. Sarà dunque necessario che l’utente attivi il Bluetooth del proprio dispositivo prima di iniziare la partita, rendendolo disponibile per 300 secondi, tempo che sarà utilizzato per la ricerca di dispositivi nelle vicinanze, ovvero i nuovi avversari da affrontare.

**Attivazione, Ricerca e Accoppiamento dei dispositivi Bluetooth vicini**

Dopo aver fornito una piccola introduzione a ciò che l’utente si troverà davanti scaricando la nostra applicazione, parliamo dell’aspetto più tecnico e di ciò che realmente c’è dietro la nostra app. La prima tematica da affrontare riguarda l’attivazione, la ricerca e l’accoppiamento dei dispositivi vicini. Cliccando il bottone collegato all’attivazione del Bluetooth andremo a controllare lo stato di attività o inattività del **BluetoothAdapter**. Nel caso in cui il dispositivo non supporti il Bluetooth (rendendo quindi impossibile l’utilizzo dell’applicazione), apparirà un Toast che lo comunicherà in via istantanea all’utente. Se invece il dispositivo supporta il Bluetooth, quest’ultimo verrà attivato e saremo pronti per iniziare la nostra partita.

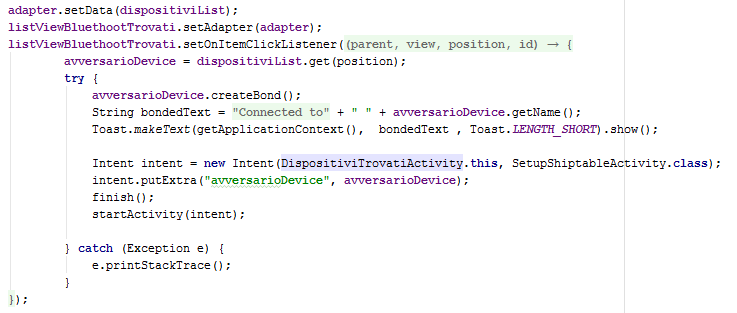




Cliccando sul bottone “inizia la partita” un **BroadcastReceiver** si occuperà di aggiungere ad un ArrayList di Devices, ogni dispositivo localizzato durante la scansione e di mostrare un Dialog, che metterà in attesa l’utente, fino alla fine della ricerca. Il nome e l’indirizzo dei vari dispositivi trovati verranno mostrati in una custom ListView e, cliccando su uno degli item relativi alla lista, si seleziona l’avversario con cui si ha intenzione di giocare. Viene invocato quindi, il metodo **createBond(),** che si occuperà dell’effettivo accoppiamento dei due dispositivi. Nel caso in cui nessun dispositivo si trovi nelle vicinanze durante la scansione, il giocatore verrà avvertito con un Dialog, che lo riporterà alla pagina iniziale, in modo da poter riavviare una nuova scansione.

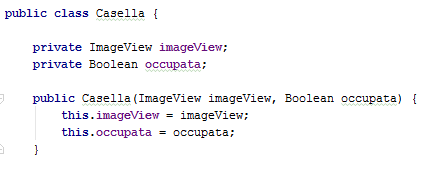
**Implementazione del BroadcastReceiver:**

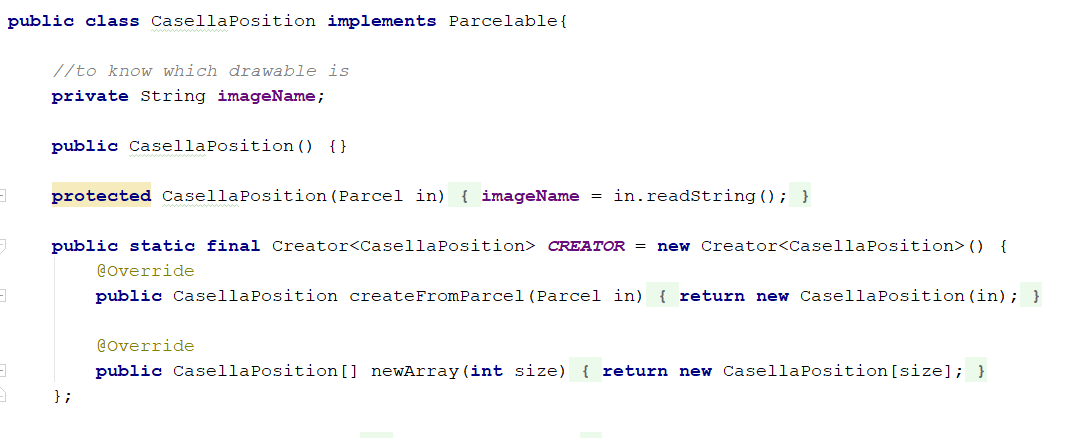


**Invocazione del metodo createBond():**

**Drag and Drop delle navicelle spaziali sul campo di battaglia**

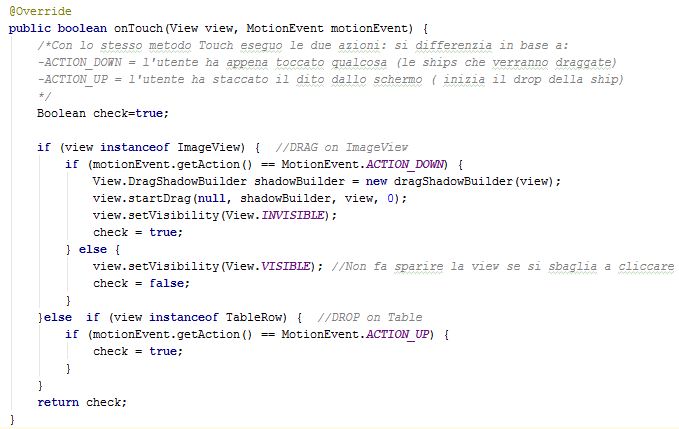
Una volta scelto l’avversario contro cui battersi, inizia per l’utente una delle fasi più scrupolose del gioco: il posizionamento delle navi! Per rendere divertente, ma soprattutto intuitiva questa fase, abbiamo scelto di utilizzare il metodo del Drag and Drop, con il quale l’utente potrà decidere la posizione di ogni navicella spaziale, trascinandola semplicemente sul campo di battaglia prestabilito. Tale campo è stato creato attraverso un **TableLayout** di ImageView. Ogni casella di gioco è descritta nel sistema usando una **struttura Casella,** costituita dall’ImageView della griglia e da un valore di tipo Boolean per impostare il suo campo come occupato o non occupato da una nave. La struttura Casella viene poi utilizzata in tutte le altre Activities attraverso la **struttura CasellaPosition**, che tiene traccia del nome dell’immagine usata su quella Casella.





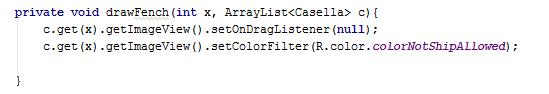
All’interno della classe **SetupShipTableActivity** troviamo uno dei metodi più importanti per la realizzazione di un Drag and Drop, il metodo **OnTouch()**:questo gestisce sia le ImageView del campo di battaglia che le ImageView delle navicelle spaziali, intercettando tutti i **Motion events** più importanti.

Cliccando una delle navicelli disponibili infatti, il metodo onTouch() registra l’evento **MotionEvent.ACTION\_DOWN** costruendo una Shadow dell’imageView attraverso la classe **DragShadowBuilder**, mentre rende invisibile quella cliccata dall’utente, dando quindi l’idea di aver “staccato” l’immagine e di poterla trascinare fino al campo di battaglia nella posizione da lui preferita.



Nel momento in cui l’utente rilascia l’immagine, viene prodotto un **MotionEvent.ACTION\_UP**, che attiva il metodo **onDrag()**. All’interno di questo metodo abbiamo suddiviso i possibili risultati attraverso un costrutto Switch-case e, in particolare, nel case DragEvent.ACTION\_DROP si gestisce singolarmente il cambiamento del Background delle caselle sulla griglia (interessate dal Drop) in base alla grandezza della nave. A quel punto La classe **ShipPosition** aggiornerà i valori dell’ArrayList di Caselle e dell’ArrayList di Caselle Position, che rappresenteranno il campo di battaglia.

Inoltre una delle regole principali nel gioco Battaglia Navale è quella di non poter posizionare due navi a meno di una casella di distanza l’una dall’altra. Per questo motivo, nel momento in cui viene posizionata la prima navicella, si attiva il metodo **drawFench(),** che disegna un recinto di una casella intorno alla nave appena posizionata sul campo di battaglia.



La classe **ShipFence** avrà il compito di selezionare le singole ImageView intorno a quelle usate per posizionare la nave e, dopo aver invalidato il DragListener, applica un filtro sull’immagine presente, rendendo visibile all’utente la differenza tra le caselle occupate come recinto e le caselle ancora disponibili ad ospitare le navicelle.

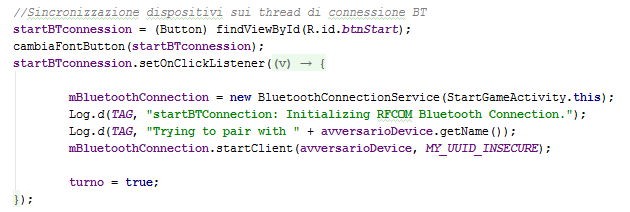
**Metodo onDrag() :**



**Scambio dati bluetooth tra i dispositivi (l’effettiva partita)**

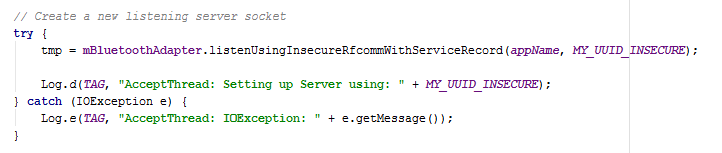
Nell’activity di gioco **StartGameActivity** si svolge l’effettiva partita. Il suo Layout è composto da due TableLayouts, che corrispondono ai due campi di battaglia, aventi però funzionalità diverse. Sul campo di battaglia di sinistra, l’utente potrà vedere le proprie navicelle che sono state posizionate da lui nell’activity precedente, mentre su quello a destra, l’utente potrà colpire le navicelle dell’avversario cercando di individuare la loro posizione nel campo di battaglia.

Ad ogni turno, il TableLayouts di sinistra si aggiornerà in base alle mosse che sono state fatte dall’avversario, in modo da poter vedere quali navicelle sono state abbattute dal nemico e quali no. Scendendo nei particolari della realizzazione della partita troviamo, una connessione Bluetooth che viene attivata in contemporanea dai due giocatori cliccando sul bottone Start, istanziando la classe **BluetoothConnectionService** e invocando il metodo **startClient().**



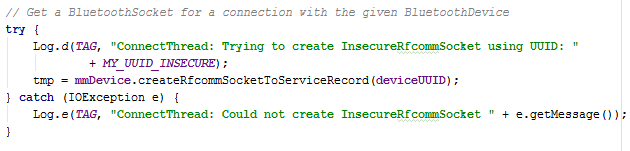
Lo scambio effettivo dei dati avviene per mezzo di tre Threads:

* **AcceptThread(),**

che ha il compito di attendere richieste di connessione in entrata e, attraverso il metodo **listenUsingInsecureRfcommWithServiceRecord(),** crea una **BluetoothServerSocket** sicura *in ascolto* usando l’indirizzo MAC del Device dell’avversario e il codice identificativo UUID. Tale Thread, termina solo quando la connessione stabilita è stata accettata da entrambi i dispositivi.

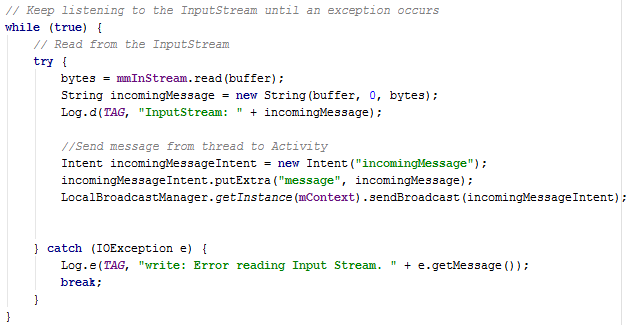
* **ConnectThread(),**

che ha il compito di creare una connessione *in uscita* verso un altro dispositivo e, attraverso il metodo **createRfcommSocketToServiceRecord(),** crea una **BluetoothSocket** utilizzando il codice identificativo UUID. Tale Thread, se termina con successo, restituisce un canale di connessione tra i due giocatori.

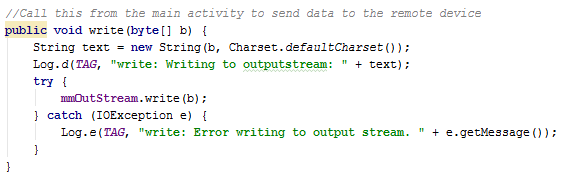
****

* **ConnectedThread(),**

che ha il compito di rimanere sempre in ascolto fino all’invio o la ricezione di dati. Infatti, attraverso un InputStream riceve informazioni in entrata, mentre invia dati utilizzando un OutputStream. Una volta letti i bytes ricevuti, il Thread passa il messaggio alla StartGameActivity attraverso un **LocalBroadcastManager**, sotto forma di Intent.

****

Il metodo **write()** di quest’ultimo Thread, richiamato nell’Activity, permette lo scambio dei dati relativi alla partita, scrivendo sull’OutputStream.



**La domanda che ora ci poniamo è: in che modo si relazionano i thread dei due dispositivi permettono lo scambio dati?**

Quando una delle ImageView del TableLayout viene cliccata, il ConnectedThread del dispositivo **A** scrive sul proprio OutputStream la posizione della casella cliccata, mentre il ConnectedThread del dispositivo **B** la riceve nell’InputStream. A questo punto si attiva il **BroadcastReceiver** del dispositivo B, che disegna l’evento nella casella corrispondente del TableLayout di sinistra (tale evento sarà rappresentato da un’esplosione se la navicella è stata colpita o da un “flop” nell’acqua altrimenti) e invia all’avversario la stessa informazione, scrivendo sull’OutputStream del proprio ConnectedThread.

Dopo aver disegnato su entrambi i dispositivi le relative immagini, i due ConnectedThread si rimettono in stato di attesa e l’ImageView diventa non più cliccabile attraverso **invalidate()**.

I giocatori capiscono quando si tratta del proprio turno in base al colore dello sfondo dell’Activity:

* **ROSSO**, nel caso di attesa dell’avversario,
* **NERO,** quando è possibile cliccare.

Dopo che sono state colpite tutte le caselle della nave, che variano in base alla sua grandezza, compare l’immagine completa della navicella distrutta e il gioco termina ovviamente quando tutte le navi avversarie sono state scoperte.

La partita si conclude quando il dispositivo del giocatore che tra i due azzera per primo i contatori corrispondenti, invia l’ultimo messaggio via Bluetooth all’avversario, che notifica la fine della partita con un Dialog di sconfitta e mostra a se stesso un Dialog di vittoria!

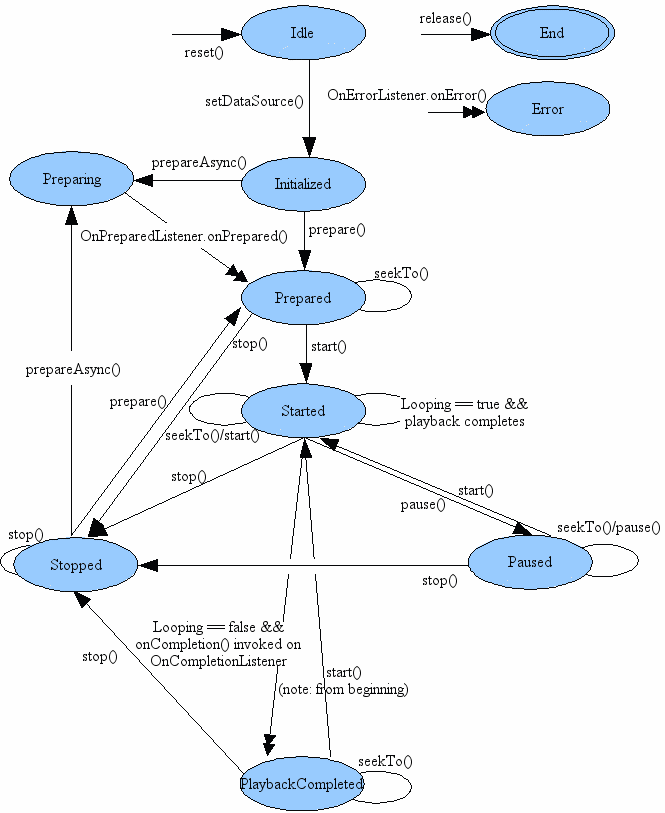
**Gestione e riproduzione di contenuti audio.**

Spesso la differenza tra un bel gioco e uno discreto è rappresentata dalla musica, infatti, come nei film o nei videogiochi, la scelta appropriata della colonna sonora e degli effetti è di vitale importanza.

Quale scelta migliore se non quella di una colonna sonora “StarWarsiana”?

Ebbene sì, “Space Battle” ci faà rivevere, nei limiti di un’applicazione mobile focalizzata su una battaglia navale, il viaggio attraverso la famosa “Galassia lontana lontana...”.

Vediamo ora l’effettiva gestione dei suoni in “Space Battle”.



Per gestire e riprodurre dei suoni in Android si utilizza una classe chiamata **MediaPlayer**, essa ha il compito di preparare il file audio, riprodurlo, fermarlo, reimpostare il puntatore a 0 secondi, se è necessario che il suona venga ripetuto, e cambiare completamente file audio se necessario.

Per evitare di andare in memory leak ne sono state utilizzate solo due istanze.

Questo diagramma a stati rappresenta in maniera esaustiva sia gli stati sia le transizioni tra di essi.

Si parte da uno stato **Idle** e, attraverso il metodo **setDataSource**(), si associa a un MediaPlayer un file audio passando così allo stato **Initialized**.

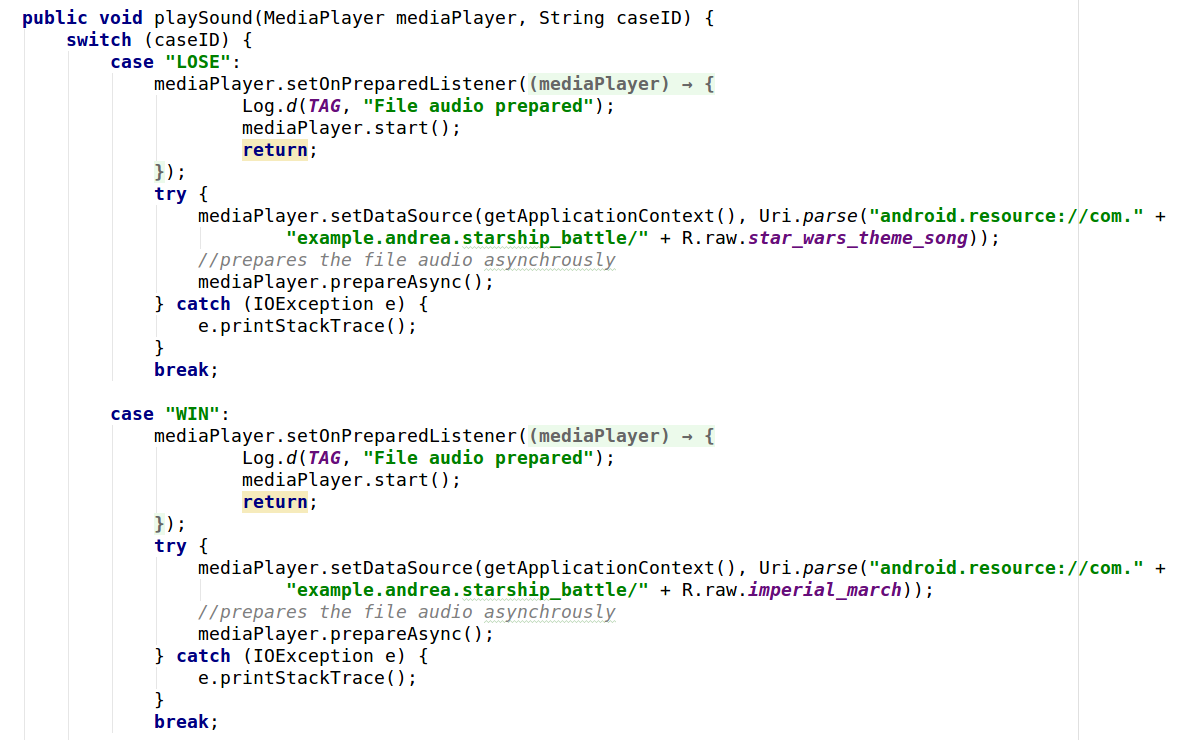
Per poter riprodurre un suono qualunque, un MediaPlayer deve trovarsi nello stato **Prepared** e, per arrivarci, si possono seguire due strade:

* quella sincrona, generalmente consigliata solamente per file audio di piccole dimensioni e breve durata, tramite il metodo **prepare**();
* quella asincrona, tramite il metodo **prepareAsync**(), preferibile alla prima se le dimensioni del file non sono modeste, in quanto, a differenza della precedente, quella asincrona non blocca la UI, evitando così di trasmettere la sgradevole sensazione di non risposta del device durante un caricamento.

La riproduzione vera e propria è attivata tramite l’invocazione del metodo **start**().

Se si intende riprodurre più volte di seguito lo stesso suono, conviene dedicare un MediaPlayer allo stesso e, una volta riprodotto, reimpostare il puntatore a 0 secondi tramite il metodo **seekTo**().

Infine per fermare la riproduzione viene utilizzato il metodo **stop**().

Il metodo più importante è sicuramente **playSound**(), il quale associa ad un MediaPlayer il suono corretto per ogni evento di cui venga gestito il suono.

Oltre a ciò, si occupa anche di settare un OnPreparedListener il quale chiamerà il metodo onPrepared(), di cui viene fatto l’override caso per caso e che a sua volta esegue la riproduzione del suono vera e propria.

Volete sentire risuonare la Marcia Imperiale? Aguzzate l’ingegno e battete il vostro avversario per conquistare la Galassia, altrimenti piegatevi alla Ribellione.

Buona Fortuna.