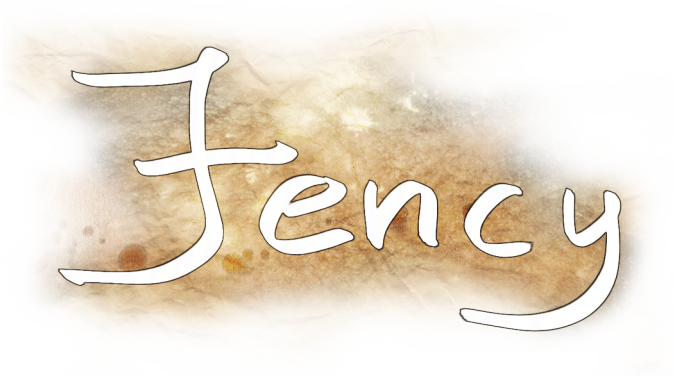


Simulatore di scherma per Android con riconoscimento di gestures

Francesco Bultrini, matricola 278696
Claudio Pannacci, matricola 283526

Anno accademico 2016-2017

Indice

0.1 Abstract

L'obiettivo di questo progetto è la realizzazione di un gioco ispirato allo sport della scherma, per cellulari dotati di sistema operativo Android che, nella sua versione finale, consisterà in una sfida locale tra due giocatori dove lo scambio di informazioni avverrà tramite Bluetooth. Si valuta anche la possibilità di utilizzare la tecnologia NFC per eseguire il *pairing* dei dispositivi.

0.2 Realizzazione

Si è deciso di utilizzare il *modello a spirale* come modello di sviluppo e ad ogni ciclo verrà realizzato un prototipo. Non tutte le funzionalità saranno presenti da subito ma verranno implementate in *release* successive.
Per la codifica è stato adottato il metodo del *Pair Programming* utilizzando Android Studio come ambiente di sviluppo.
Il codice sorgente è pubblicato su GitHub all'indirizzo:

`https://github.com/Disorganizzazione/Fency`

Parte I
Riconoscimento delle gestures e modalità allenamento

1 Analisi dei requisiti

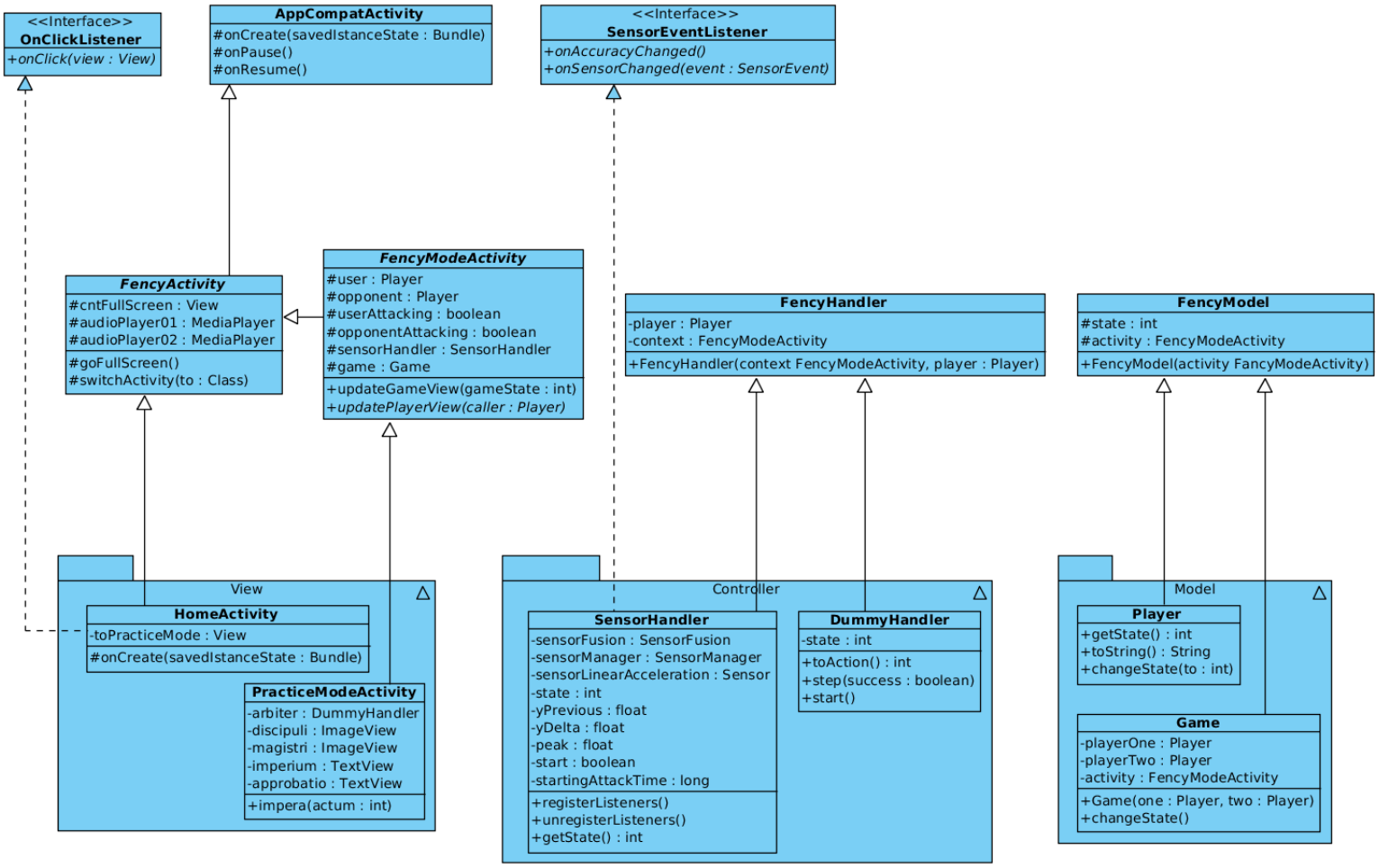
1.1 Requisiti riconoscimento gesture

Al fine di riconoscere le *gestures* si deve accedere, tramite classi di sistema, ai valori di accelerometro, giroscopio e magnetometro. Per ridurre il dominio dei movimenti da analizzare, sarà necessario definire un orientamento del telefono (in direzione orizzontale, con lo schermo rivolto verso l'alto) e assicurarsi che tale stato venga mantenuto durante lo spostamento.
A questo scopo la classe **SensorFusion** di Paul Lawitzki, che combina i dati del giroscopio a quelli del magnetometro riducendo il rumore, permette di avere informazioni affidabili sull'orientamento del dispositivo tramite i valori di *pitch*, *roll* e *azimuth*. La descrizione dell'affondo si basa sui dati forniti dall'accelerometro e, utilizzando la modalità **Linear Acceleration**, si hanno i valori di accelerazione sui tre assi, senza la componente gravitazionale. Il movimento dell'affondo causa un'accelerazione positiva sull'asse Y (parallelo al lato lungo del dispositivo) seguito da una accelerazione di modulo maggiore, stesso verso e direzione opposta, dovuta all'arresto del dispositivo nel momento in cui si conclude l'affondo. Inoltre, per impedire ad altri movimenti (come lo scuotimento) di essere classificati come affondi bisogna accertarsi dell'effettivo spostamento del dispositivo. Non basta quindi impostare una soglia minima di accelerazione ma è necessario anche stimare la durata del movimento.

1.2 Requisiti modalità allenamento

2 Struttura del codice

2.1 Diagramma di classe

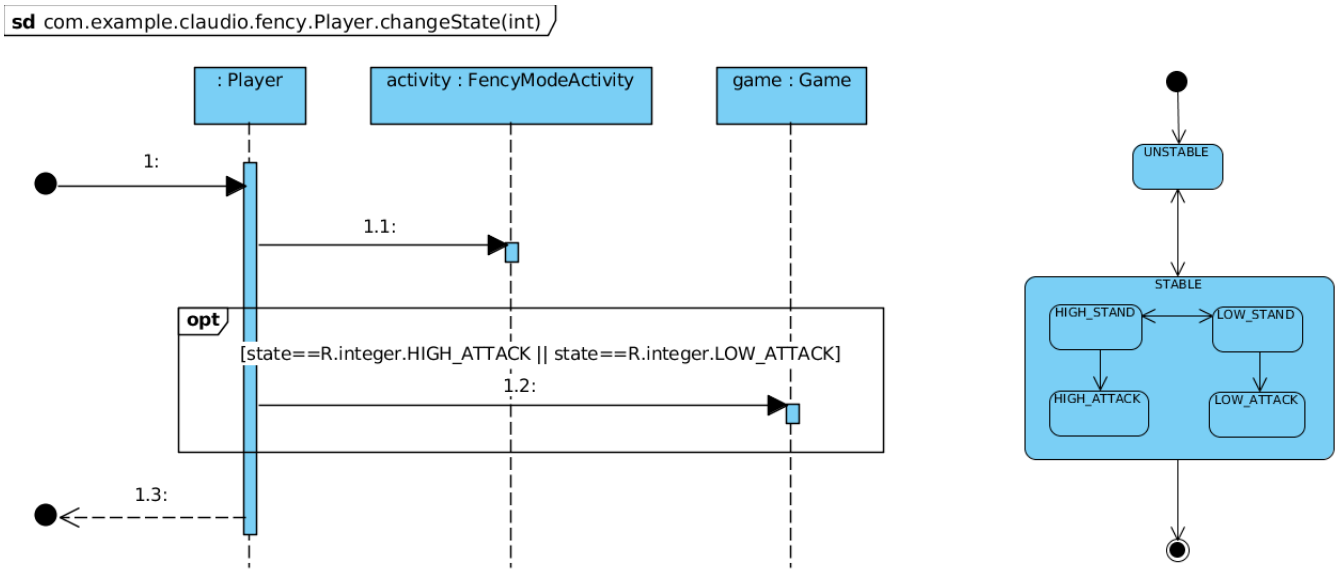


2.2 Design pattern

Come mostrato nel diagramma, abbiamo applicato una variante del design pattern *Model View Controller* nella quale il controller dei modelli giocatore (**SensorHandler**) ne modifica lo stato in base ai valori dei sensori anzi che in risposta ad un' interazione con le loro componenti grafiche da parte dell'utente. Ogni volta che lo stato del modello cambia, esso chiama la funzione *updatePlayerView()* che ne aggiorna la visualizzazione.

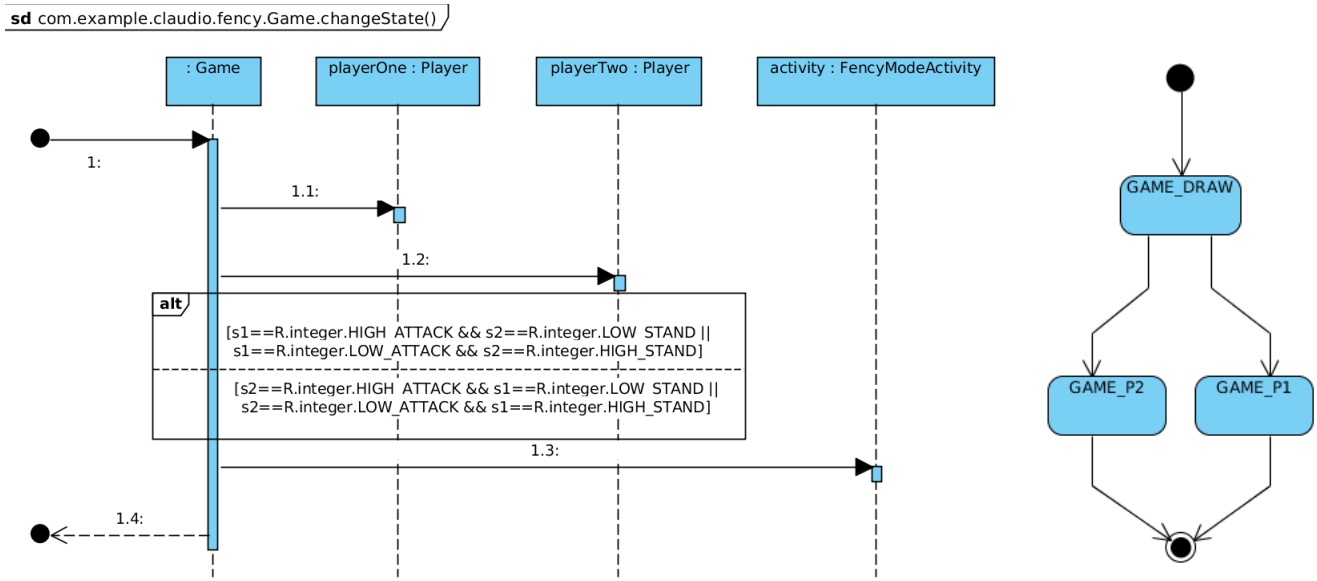
2.3 Diagrammi di sequenza e di stato

2.3.1 Cambiamento dello stato del giocatore



Ogni volta che un giocatore passa ad uno stato di attacco, il **Player Model** chiama la funzione *ChangeState()* del **Game Model**...

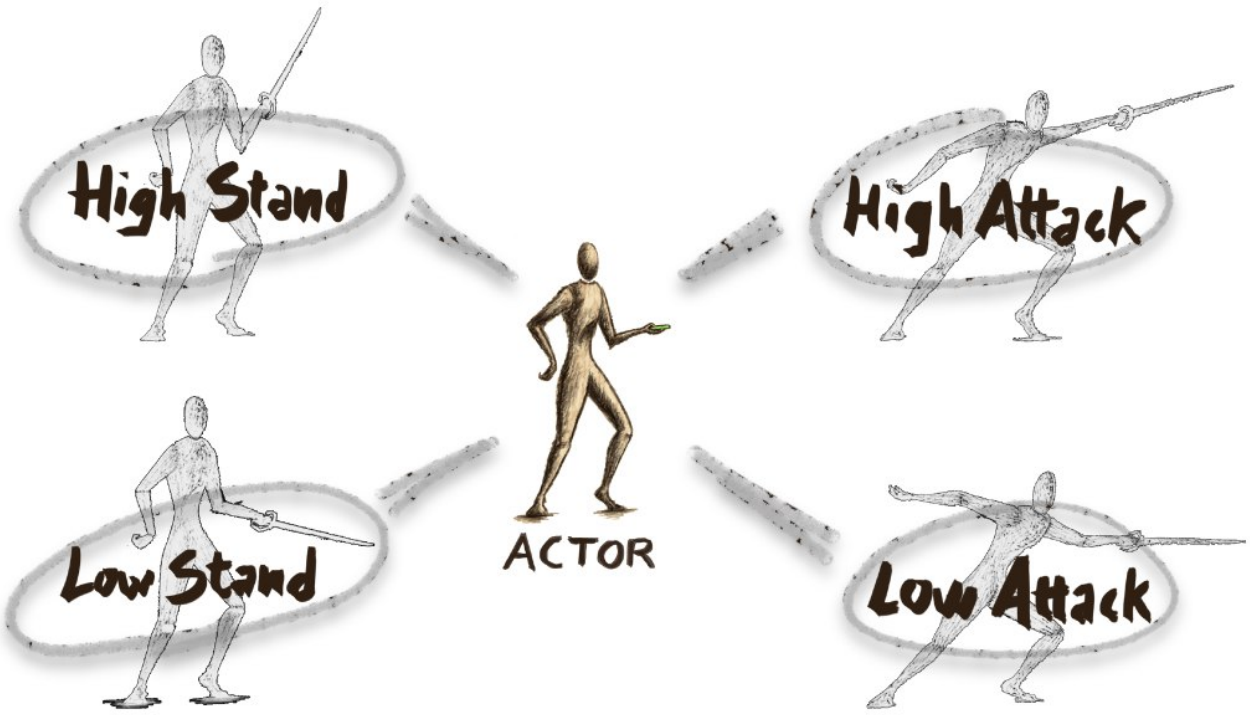
2.3.2 Cambiamento dello stato di gioco



...che controlla gli stati di entrambi i giocatori.

3 Meccaniche di gioco

3.1 Casi d'uso



3.2 Bho

Il giocatore bla bla

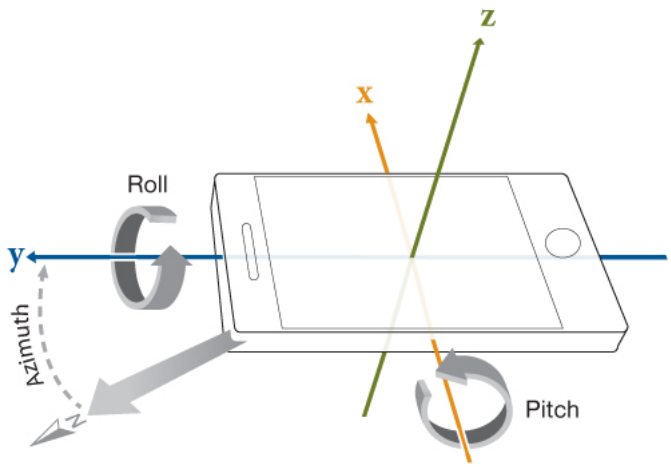
4 SensorHandler

4.1 Stato

4.2 Attività

Glossario

gesture: specifico movimento del dispositivo
pairing: connessione reciproca tra due dispositivi
release: rilascio del software
Pitch, Roll, Azimuth: (come in figura)



Modello a spirale: modello di cilco di vita del software proposto da Barry Boehm nel 1988
https://it.wikipedia.org/wiki/Modello_a_spirale
Pair programming: tecnica di sviluppo del software nella quale due programmatori lavorano insieme a una postazione di lavoro
https://it.wikipedia.org/wiki/Pair_programming
Model View Controller (MVD): è un pattern architetturale molto diffuso nello sviluppo di sistemi software ad oggetti che separa la logica di visualizzazione di un oggetto da quella di gestione.
https://it.wikipedia.org/wiki/Model-View-Controller