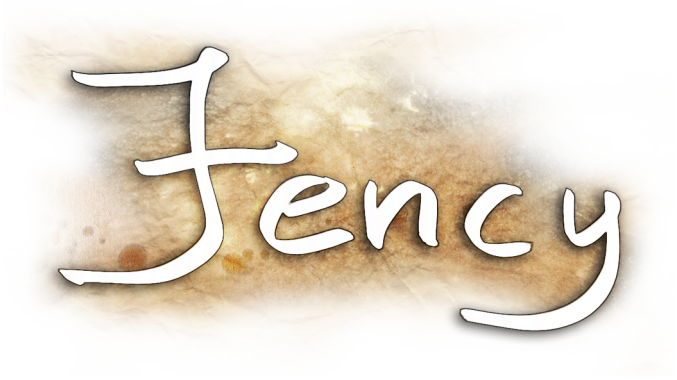


Simulatore di scherma per Android con riconoscimento di gestures

Francesco Bultrini, matricola 278696
Claudio Pannacci, matricola 283526

Anno accademico 2016-2017

Indice

Elenco delle figure

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 interfaccia grafica

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, accessibili mediante l'implementazione dell'interfaccia **SensorEventListener**. 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.

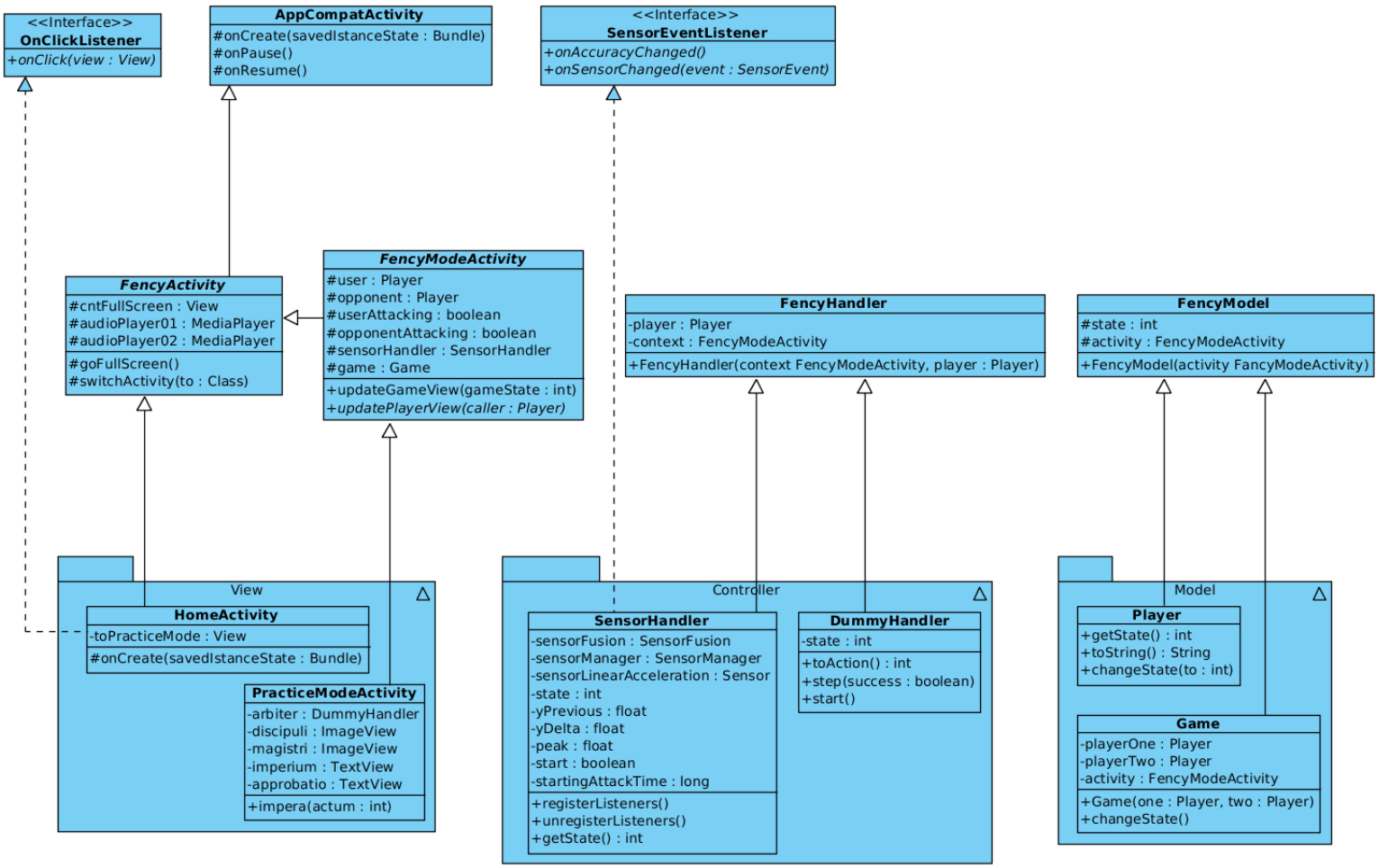
1.2 Requisiti interfaccia grafica

Al fine di interagire con l'utente tramite un'interfaccia grafica, si ricorrerà alla classe di sistema **AppCompatActivity**, la quale si lega ad un *layout* in formato XML e viene eseguita all'apertura dell'app.
Il passaggio da un' attività all'altra sarà gestita implementando l'intefaccia **OnCLickListener**, la quale permette di assegnare agli oggetti visuali una funzione di risposta personalizzata.

2 Descrizione del codice

Il codice presenta tre livelli: classi e interfacce di sistema, classi astratte ***Fency*** e classi concrete.

Figura 1: Digramma di classe completo



2.1 Design pattern MVC

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 **Player Model** cambia, esso chiama la funzione *updatePlayerView()* che ne aggiorna la visualizzazione.
Similmente il **Game Model** chima la funzione *updateGameView()*.

2.1.1 Relazione tra i Model

Ogni volta che un giocatore passa ad uno stato di attacco, il **Player Model** chiama la funzione *ChangeState()* del **Game Model**: esso controlla gli stati di entrambi i giocatori e si aggiorna.

Figura 2: Diagramma di stato e di sequenza per la classe Player

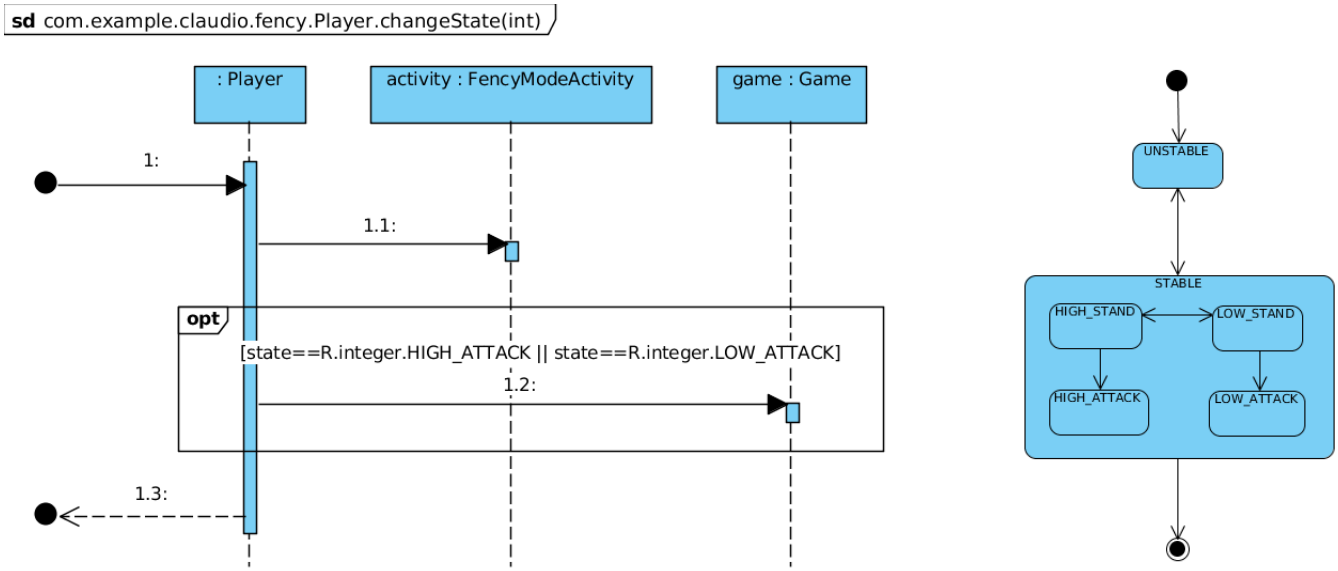
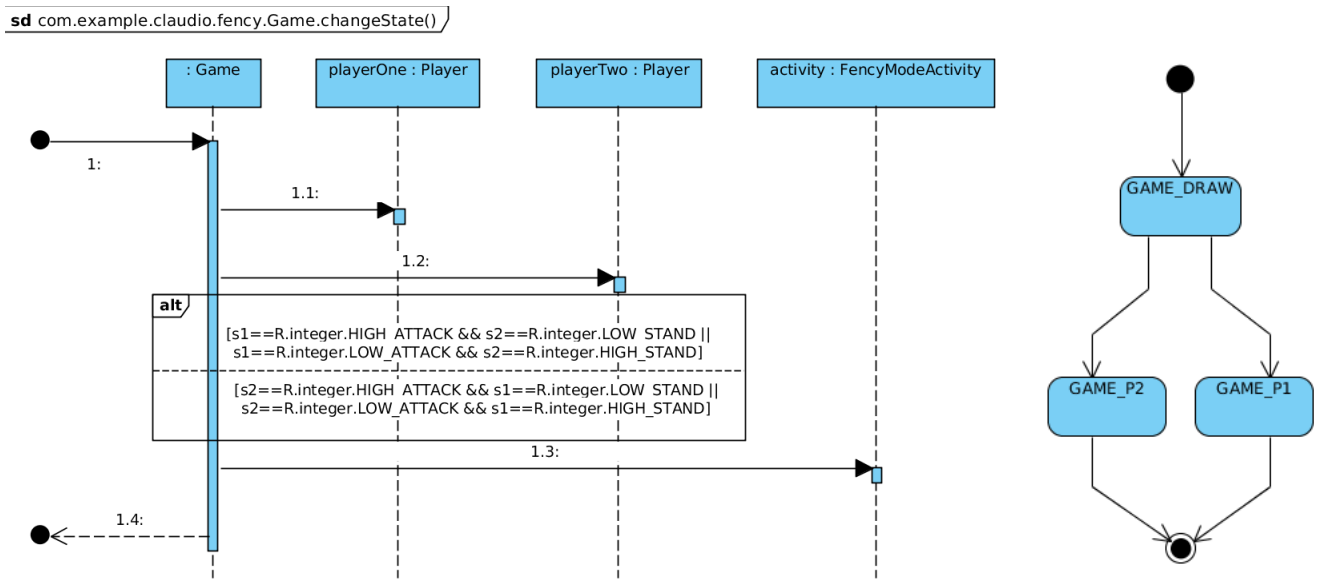
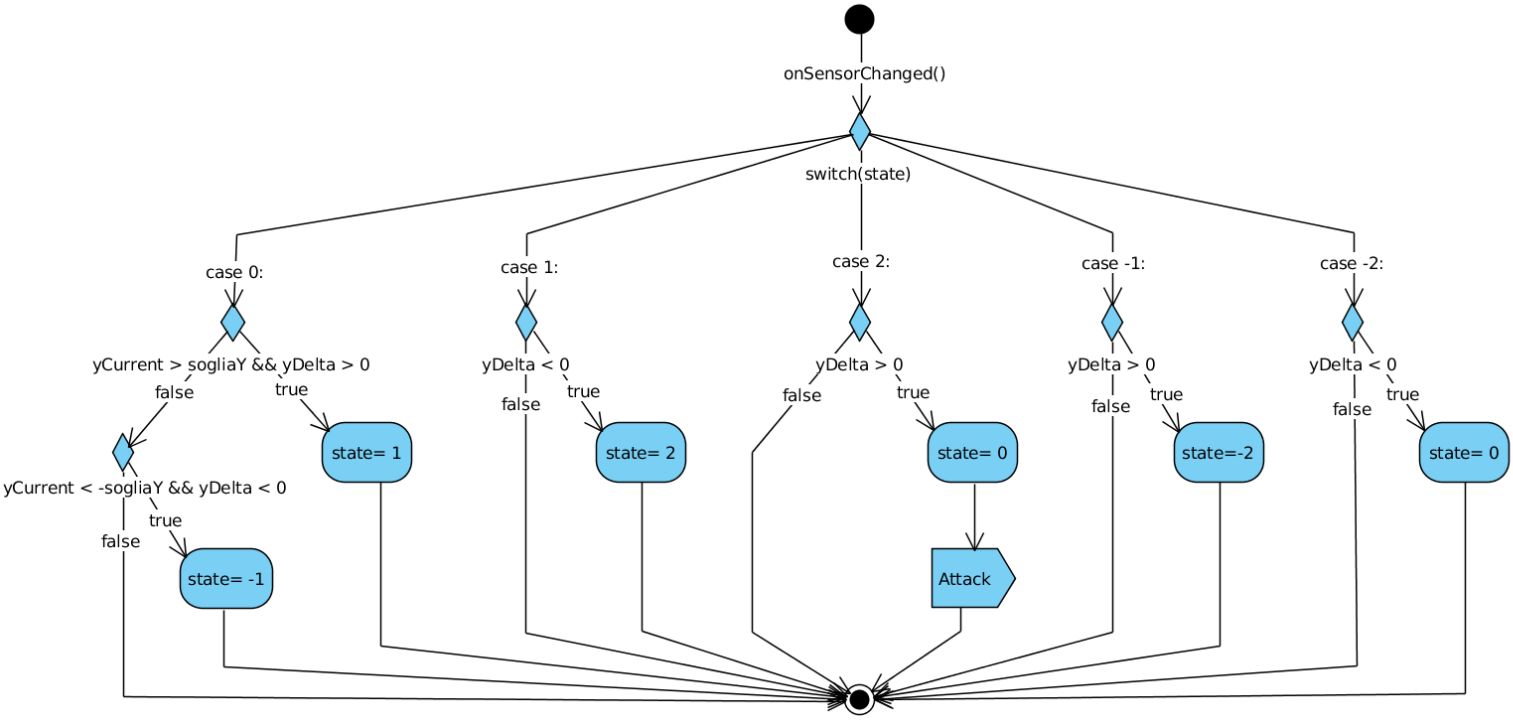


Figura 3: Diagramma di stato e di sequenza per la classe Game



2.2 La classe SensorHandler

Figura 4: Diagramma di attività: SensorHandler.onSensorChanged()



2.2.1 Casi di test funzionali

N	Pitch	Roll	Validità
1	-30	0	Valido
2	-70	0	Non valido
3	5	0	Non valido
4	-30	-8	Non valido
5	-30	8	Non valido

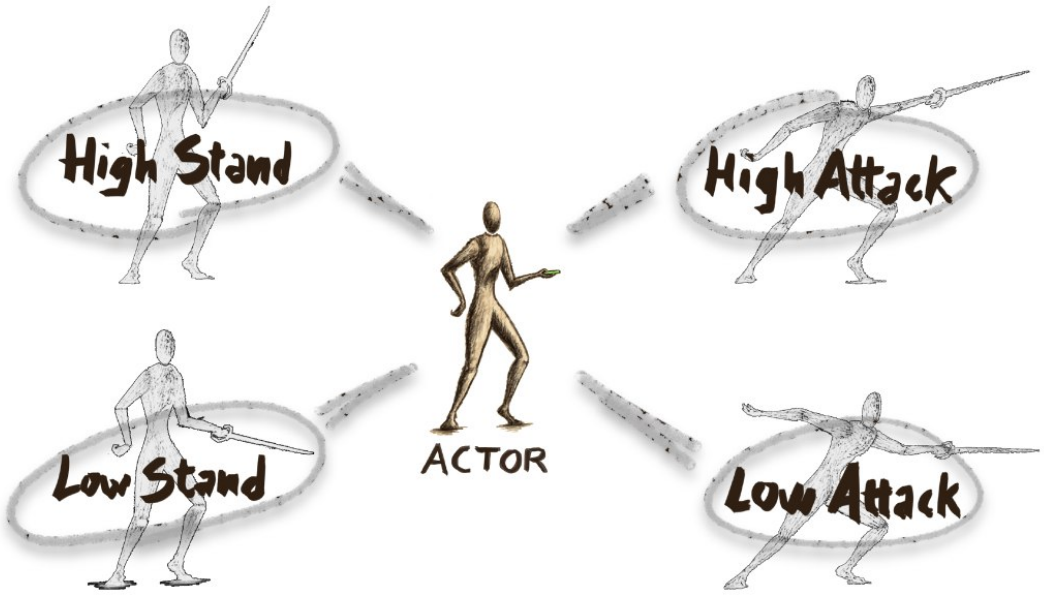
2.2.2 Casi estremi

	Validi	Non valdidi
Pitch	$-60 + DOUBLE_m, -5 - DOUBLE_m$	$-60, -5$
Roll	$-5, 5$	$-5 - DOUBLE_m, 5 + DOUBLE_m$

3 Meccaniche di gioco

Il gioco prevede che l'utente sia possibilmente in piedi e tenga con una mano il dispositivo davanti a sé: da questa posizione può alternarsi rapidamente fra i due stati di guardia HighStand e LowStand semplicemente regolandone il pitch. Partendo da una di queste posizioni è possibile, mimando un affondo con la dovuta velocità, raggiungere rispettivamente gli stati HighAttack e LowAttack.

Figura 5: Casi d'uso: posizioni del giocatore

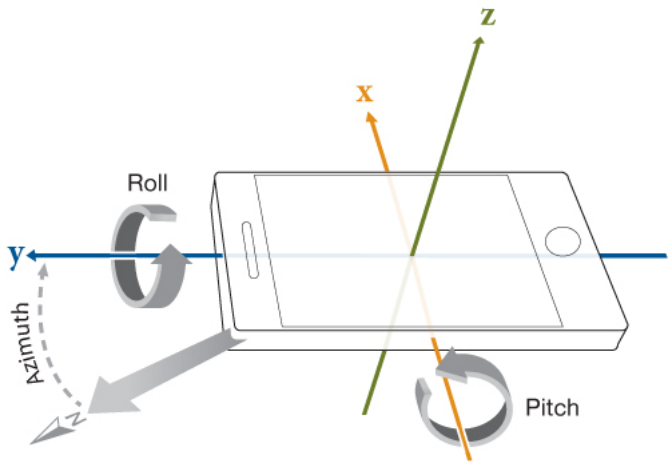


3.1 Scopo del gioco

Lo scopo del gioco è colpire l'avversario attaccandolo all'altezza opposta alla sua guardia. Al contrario, vi è parità nello scambio di colpi quando le due "spade" si scontrano virtualmente: un attacco può essere quindi deflesso sia correggendo lo stato di guardia in base all'attacco in arrivo, sia compiendo un attacco alla stessa altezza dell'avversario.

Glossario

Gesture: specifico movimento del dispositivo
Layout: disposizione grafica degli elementi
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): 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