ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

NINJA TYPING: TYPING GAME SU PIATTAFORMA ANDROID

Relazione finale in

MOBILE WEB DESIGN

Relatore

Dott. Mirko Ravaioli

Presentata da

Michelangelo Arcuri

Sessione III
Anno Accademico 2012-2013

INDICE

Introduzione 1

Capitolo 1: Introduzione ai concetti fondamentali del mondo mobile

1.1 Nascita ed evoluzione dello Smartphone nel tempo 3

1.2 Tablet, Smartphone, i motivi del successo 4

1.3 Applicazioni, App Store, Android 6

1.4 NinjaTyping, concetti principali 8

1.5 NinjaTyping, dispositivi supportati 10

Capitolo 2: Progettazione dell'applicazione

2.1 Analisi dei requisiti 13

2.2 Schermata di gioco e partita 16

2.3 Riepilogo di gioco e condivisione dati partita 18

2.4 Gestione partite online 20

2.5 Riepilogo partita online 22

2.6 Sezione statistiche 24

2.7 Account NinjaTyping 25

2.8 Server e memorizzazione dati 25

2.9 Algoritmo di ricerca di un avversario online 28

2.10 Layout della schermata di gioco 31

Capitolo 3: Implementazione dell'applicazione

3.1 Schermata di Splash e Home iniziale 35

3.2 Implementazione dell'account utente 38

3.3 Scelta della modalità in Partita Singola 40

3.4 Implementazione grafica della partita 41

3.5 Gestione del dizionario 47

3.6 Sviluppo dell'algoritmo di gioco 48

3.7 Sezione Riepilogo di Gioco 51

3.8 Sezione statistiche e database locale 56

3.9 Ricerca di un avversario online, Client 61

3.10 Ricerca di un avversario online, Server …..................... 64
3.11 Gioco della partita online e verifica dell'esito …............ 67

Capitolo 4: Sviluppi futuri del progetto NinjaTyping

4.1 Database e Web site ….. 71
4.2 Miglioramenti prestazionali dell'applicazione …............. 72
4.3 Miglioramenti grafici …... 73
4.4 Nuova modalità di gioco e dizionari stranieri …............. 75
4.5 Sviluppi della modalità di gioco online …....................... 76
4.6 NinjaTyping per iOS e scelte commerciali ….................. 78

Conclusioni ….. 83

Bibliografia 85

Introduzione

La tesi di laurea è l'ultimo passo di un cammino universitario intenso, difficile e fortemente formativo. Ho pensato quindi di scegliere una tesi in cui potessi utilizzare la maggior parte delle competenze che ho assunto in questi anni. La scelta è caduta nello sviluppo di un'applicazione *mobile* su piattaforma *Android.*

Lo sviluppo di una tesi di questo tipo, mi ha permesso di cimentarmi in un'esperienza completa di progettazione e implementazione di un software, grazie alla quale ho praticato e approfondito molte delle conoscenze apprese in questi anni, quali programmazione ad oggetti, progettazione e sviluppo di database, modelli di comunicazione *client-server* e *layout* grafici.

Ho scelto di sviluppare un'applicazione mobile per altri motivi oltre a quelli prettamente didattici appena descritti. La mia intenzione era quella di produrre qualcosa di *concreto,* che potesse essermi utile nella non sempre facile transizione università/mondo del lavoro. Lo sviluppo di questa applicazione mi permette quindi di essere più pronto ad entrare in un ambito commerciale, che ai giorni nostri sta avendo un'espansione davvero impressionante. Sto parlando ovviamente della crescita esponenziale che i dispositivi mobili stanno avendo e continueranno ad avere nei prossimi anni, producendo di fatto un aumento della richiesta di programmatori per dispositivi portatili. Infatti la qualità, le funzionalità sempre maggiori e i prezzi contenuti dei dispositivi mobili quali S*martphone* e T*ablet,* stanno di fatto avendo l'effetto di accelerare il passaggio dell'utenza media, dai *Personal Computer* ai dispositivi *mobile.*

Per quanto riguarda il tipo di applicazione che ho sviluppato, si tratta di un cosiddetto *TypingGame,* cioè un semplice gioco volto ad intrattenere l'utente attraverso la digitazione di determinati *oggetti* presenti su schermo. Il motivo della scelta di questo tipo di applicazione è dettato da una serie di ragionamenti e constatazioni che ho potuto formulare in questi ultimi anni e che ora esplicherò sinteticamente.

L'ambito *mobile*, ha il grosso pregio di riuscire a valorizzare il lavoro di programmatori indipendenti che producono applicazioni di successo, dove il successo è quantizzato nel

numero di persone che utilizzano una determinata applicazione. Per programmatori (sviluppatori) indipendenti, intendo persone che non hanno alle spalle grosse aziende e quindi grossi budget per lo sviluppo di software. Se si fa parte di questa categoria di sviluppatori, il successo può arrivare creando principalmente *applicazioni di scopo* (app che fanno una cosa sola ma la fanno bene) e *applicazioni di intrattenimento* quali possono essere appunto giochi di diverso tipo. Io ho optato per il secondo tipo, andando a sviluppare qualcosa che nel suo nucleo, quindi nel gioco in sé, è banale ma che potrebbe piacere proprio per la sua semplicità.
Negli ultimi tempi molte applicazioni Desktop e Web sono state riproposte con notevole successo, anzi oserei dire maggior successo, per terminali mobili. La mia idea è stata quella di riproporre su mobile quello che in maniera un po' diversa esiste su Desktop, cioè un *gioco* (ma è più corretto chiamarla App d'intrattenimento) che valuti la capacità di un utente nel scrivere, attraverso la tastiera, una frase casuale. Un'applicazione di questo tipo non è ancora stata riprodotta su dispositivi mobili (perlomeno nel momento in cui ho cominciato a svilupparla io), così ho deciso di creare *NinjaTyping,* la mia applicazione nonché tesi di laurea che ha come scopo quello precedentemente descritto. Ho dovuto ovviamente adattare ed ampliare l'idea di base, per renderla più appetibile possibile ad utente *mobile,* introducendo quindi la condivisione dei risultati ottenuti con NinjaTyping, oppure la possibilità di sfidare altre persone in una sorta di *challenge online,* che da quanto ho potuto constatare, sono indice di forte gradimento.

Come si può intuire ho quindi cercato di unire l'utile al dilettevole, sviluppando una tesi di laurea che possa formarmi ancor di più e con un po' di fortuna possa entrare con successo nel mercato delle applicazioni *mobile*.
Nel corso della stesura della tesi descriverò nello specifico la progettazione e l'implementazione del progetto, effettuerò delle considerazioni su cosa posso migliorare in futuro nella mia applicazione, oltre all'introdurre il tutto attraverso un capitolo in cui descriverò il meccanismo che sta dietro al mondo applicativo (lo *store* delle applicazioni e altro) e quindi spiegherò alcune delle scelte, che mi hanno portato a sviluppare questa tesi in un determinato modo e per un certo bacino d'utenza.

Capitolo 1
Introduzione ai concetti fondamentali del mondo mobile

In questo capitolo introdurrò una serie di concetti di base importanti.
Partirò con una brevissima introduzione comprensiva di cenni storici su cos'è uno Smartphone, quando nasce e quando avviene la piena esplosione dei dispositivi per il mercato di massa.
Procederò descrivendo i motivi che hanno decretato il successo di questi dispositivi, quindi elencherò la filosofia commerciale che c'è dietro il mondo mobile, cioè il *mercato delle applicazioni.* Illustrerò infine il funzionamento generale di NinjaTyping, spiegando inoltre i motivi che mi hanno spinto a programmare per la piattaforma *Android* e per un determinato bacino d'utenza.

1.1 Nascita ed evoluzione dello Smartphone nel tempo

Penso sia interessante conoscere in maniera sintetica, la storia dei dispositivi sul quale sarà installata l'applicazione che ho sviluppato.
Innanzitutto uno S*martphone*, è un telefono cellulare basato su un sistema operativo per dispositivi mobili, con capacità di calcolo e di connessione molto più avanzate rispetto ai normali telefoni cellulari. Più semplicisticamente uno Smartphone come lo conosciamo oggi è un'evoluzione dei normali telefoni cellulari, che permette di avere molte delle funzionalità di un Pc letteralmente a portata di mano.
Per trovare quello che è considerato il primo Smartphone della storia, bisogna andare molto più indietro di quanto si possa pensare. Nel 1994, l'azienda statunitense *IBM,* produce il telefono cellulare *SIMON* che comprendeva applicazioni quali un blocco note, un calendario, un client e-mail, giochi e ovviamente poteva telefonare, ma anche inviare fax. Il tutto era gestibile attraverso uno schermo *touchscreen* e un pennino.
Per trovare i primi Smartphone capaci di affermarsi su larga scala, bisogna però

aspettare l'inizio degli anni duemila e la conseguente distribuzione del *BlackBerry*, prodotto dalla società *Research In Motion* (*RIM*), già conosciuta per la produzione dei cosiddetti *cerca-persone* (il termine originale è *Pager*). Questi Smartphone BlackBerry permettevano di leggere email, consultare allegati e navigare in Internet con un *Browser Mobile*.
Oltre all'azienda RIM, anche altri produttori (*Nokia*, *Htc* e altri) si cimenteranno nello sviluppo di Smartphone, ma la vera rivoluzione tecnica e commerciale avviene nel 2007, quando Steve Jobs CEO dell'azienda *Apple Inc*. presenta lo Smartphone *Iphone 2G***.** Il prodotto della casa di Cupertino era una perfetta fusione tra *iPod* (lettore mp3), computer palmare, telefono cellulare e si basava sul sistema operativo *iOS* . Era un prodotto molto semplice da usare, grazie a un'interfaccia intuitiva e a uno schermo capacitivo interagibile direttamente con le dita della mano e non più con lo scomodo pennino.
Il successo del prodotto Apple ha inevitabilmente generato un nuovo impulso commerciale e stimolato la nascita di un'agguerrita concorrenza, che ha portato alla creazione di terminali sempre più innovativi, veloci e comodi che hanno determinato la trasformazione sociale che viviamo oggi.

Nel paragrafo successivo andrò ad illustrare i numeri che decretano il successo dei dispositivi mobili (non solo Smartphone ma anche Tablet) e analizzerò i motivi di questo successo.

1.2 Tablet, Smartphone i motivi del successo

L'esplosione nel mercato mondiale di S*martphone* e T*ablet,* sta di fatto avendo l'effetto di migrare la cosiddetta utenza media, dai Personal Computer ai dispositivi *mobile.* A supporto di tale affermazione, riporto in *figura 1.1* il risultato di uno studio prodotto dalla *Gartner Inc*, una società leader mondiale nella consulenza strategica, ricerca e analisi nel campo dell' *Information Tecnology.*

Worldwide Devices Shipments by Segment (Thousands of Units)				
Device Type	**2012**	**2013**	**2014**	**2017**
PC (Desk-Based and Notebook)	341,263	315,229	302,315	271,612
Ultramobile	9,822	23,592	38,687	96,350
Tablet	116,113	197,202	265,731	467,951
Mobile Phone	1,746,176	1,875,774	1,949,722	2,128,871
Total	**2,213,373**	**2,411,796**	**2,556,455**	**2,964,783**
Source: Gartner (April 2013)				

Figura 1.1 Crescita dei dispotivi nel tempo

Lo studio effettuato dalla società *Gartner,* evidenzia come nel 2013 le unità di dispositivi (Pc, Tablet, Smartphone) siano 2,4 miliardi, con un incremento quindi significativo rispetto al 2012 dove i dispositivi erano 2,2 miliardi. Questa crescita proseguirà anche nei prossimi anni, ma quello che voglio evidenziare è la composizione delle unità. Mi spiego meglio, nei prossimi anni avremo più dispositivi ma chi crescerà in modo significativo saranno i Tablet e gli Smartphone, mentre i PC (fissi e portatili) sono destinati a diminuire.

Tutto questo è possibile e spiegabile principalmente per motivi strettamente correlati tra loro. Innanzitutto i dispositivi mobili stanno diventando sempre più potenti, con i dispositivi di ultima generazione che hanno una potenza di calcolo (destinata ancora a crescere) che non ha nulla da invidiare ai Pc di qualche anno fa. Questo comporta la possibilità di far girare su dispositivi portatili, programmi tradizionali che l'utente medio utilizza con il proprio Pc .

Su un dispositivo mobile è infatti possibile telefonare, navigare, leggere/scrivere mail, guardare filmati, ascoltare musica e giocare.

La vera potenza dei dispositivi mobili che ne decreta l'appetibilità dell'utenza, sono però le applicazioni installabili. In uno Smartphone o Tablet, le funzionalità preinstallate cioè disponibili alla prima accensione, sono relativamente limitate. Oltre ad applicazioni che ci permettono di scattare foto, ascoltare musica, telefonare, navigare online e consultare mappe non sono fornite altre funzionalità *out of the box*.

Il vero potenziale nonché fascino dei dispositivi mobili è la personalizzazione delle funzionalità a seconda dell'esigenza dell'utente, che può scaricare e installare

applicazioni come NinjaTyping dal cosiddetto *Application Store* che introdurrò nel prossimo paragrafo.

1.3 Applicazioni, App Store, Android

La potenza dei nuovi terminali (e allo stesso tempo la fortuna degli sviluppatori) è data dalla personalizzazione degli stessi attraverso il download delle app. Il termine a*pp* è un'abbreviazione di *applicazione* e già questo lascia in un certo senso intuire la correlazione che c'è tra un'app di un dispositivo mobile e un'applicazione di un Personal Computer. Un app per Smartphone e Tablet, è simile per concezione alle tradizionali applicazioni presenti su PC. Infatti un'app è un *software* che fornisce in qualche modo un servizio ad un utente, con la differenza che lo fa con velocità, "leggerezza" e chiarezza.
Ogni dispositivo mobile, monta un sistema operativo che a sua volte permette il download e l'installazione di applicazioni attraverso opportuni servizi, che generalmente vengono chiamati *Market* o *Store.*
I dispositivi più diffusi sono quelli che utilizzano i sistemi operativi prodotti da Apple (iOS) e da Google (Android). Il successo in termini di diffusione di un tipo di dispositivo con un determinato sistema operativo rispetto a un altro, è strettamente collegato alla qualità dello store di applicazioni, messo a disposizione per quel determinato SO. Infatti il successo di un prodotto mobile è dato soprattutto dal numero (e qualità) di applicazioni che un utente può scaricare per personalizzare il proprio dispositivo. La conferma di quanto appena detto è riscontrabile nelle tabelle in *figura 1.2* e *1.3,* che mostrano i dati riguardanti i due principali market per dispositivi mobili chiamati A*pp store(*servizio creato da Apple per utenti iOS) e G*oogle play store* (servizio creato da Google per utenti Android).

Anno	Mese	Applicazioni disponibili	Downloads
2009	Marzo	2.300	
	Dicembre	16.000[9]	
2010	Marzo	30.000[10]	
	Aprile	38.000[11]	
	Agosto	80.000[12]	1 miliardo
	Ottobre	100.000[13]	
2011	Maggio	200.000	3 miliardi[14]
	Luglio	250.000[15]	6 miliardi
	Ottobre	319.000[16]	
	Dicembre	380.297[17]	10 miliardi[18]
2012	Gennaio	400.000[19]	
	Febbraio	450.000[20]	
	Maggio	500.000[21]	15 miliardi
	Giugno	600.000	20 miliardi[22]
	Novembre	700.000[23]	25 miliardi

Figura 1.2 Quantitativo di applicazioni scaricate e disponibili nello store di Google

Date	Applicazioni disponibili	Download accumulati
11 luglio 2008[8]	500	inaugurazione
9 settembre 2008[9]	3.000	100.000.000
22 ottobre 2008[10]	7.500	200.000.000
5 dicembre 2008[11]	10.000	300.000.000
16 gennaio 2009[12]	15.000	500.000.000
10 febbraio 2009[13]	20.000	
5 marzo 2009[14]	25.000	800.000.000[15]
23 aprile 2009[16][17]	35.000	1.000.000.000
8 giugno 2009[18]	50.000	
14 luglio 2009[19]	65.000	1.500.000.000
9 settembre 2009[20]	75.000	1.800.000.000
28 settembre 2009[21]	85.000	2.000.000.000
4 novembre 2009[22]	100.000	
5 gennaio 2010[23]	130.000	3.000.000.000
20 marzo 2010[24]	150.000	3.000.000.000
7 giugno 2010	220.000	5.000.000.000
22 gennaio 2011[25]	325.000	10.000.000.000
6 giugno 2011[26]	425.000	14.000.000.000
7 luglio 2011[27]	425.000 (100.000 per iPad)	15.000.000.000
4 ottobre 2011	500.000 (140.000 per iPad)	18.000.000.000
2 marzo 2012	585.000 (200.000+ per iPad)	25.000.000.000
11 giugno 2012	650.000	30.000.000.000
12 settembre 2012	700.000 (250.000+ per iPad)	35.000.000.000
16 maggio 2013	850.000 (350.000+ per iPad)	50.000.000.000

Figura 1.3 Quantitativo di applicazioni scaricate e disponibili nello store di Apple

Entrambi gli store mettono a disposizione degli utenti oltre 1.000.000 di applicazioni, che negli ultimi 6 anni sono state scaricate cumulativamente più di 50 miliardi di volte. Per dare un'idea dell'importanza di questi numeri, basterebbe pensare che nel mondo vivono "solo" poco più di 7 miliardi di persone.
Quando un programmatore indipendente sviluppa un'applicazione, spesso (come nel mio caso) deve decidere per quale sistema operativo sviluppare e di conseguenza in quale "mercato" andare ad inserire la propria applicazione. Io ho sviluppato NinjaTyping per terminali Android, quindi la mia applicazione andrà a collocarsi all'interno del Google Play store nella categoria Giochi.
Ho fatto questa scelta per sviluppare utilizzando il linguaggio Java e in quanto Android è un sistema più aperto, caratterizzato da una maggiore collaborazione tra la folta comunità di sviluppatori che ne fanno parte.

1.4 NinjaTyping concetti principali

Dopo la breve introduzione in cui ho espresso i concetti generali che determinano la filosofia del mondo mobile, procederò finalmente con una breve panoramica riguardante NinjaTyping.
NinjaTyping è un TypingGame molto semplice che vuole mettere alla prova l'utente riguardo l'abilità di quest'ultimo nello scrivere una serie di frasi senza senso logico, composte in maniera totalmente casuale.
Per ogni nuova partita, l'utente avrà una frase diversa da digitare. Il compito dell'utente è scrivere tutti i caratteri presenti su schermo (anche gli spazi) fino alla fine, cercando di farlo il più velocemente possibile ma anche il più correttamente possibile (evitando cioè di digitare caratteri sbagliati), per ottenere un punteggio finale. L'algoritmo (che spiegherò nel capitolo dedicato all'implementazione dell'app) che genera il punteggio, assegna uno *score* più alto in maniera proporzionale alla velocità con cui l'utente digita in modo corretto i caratteri su schermo. Se l'utente digita i caratteri sbagliati sarà penalizzato in sede di punteggio. In *figura 1.4* è riportato un esempio della schermata principale di gioco.

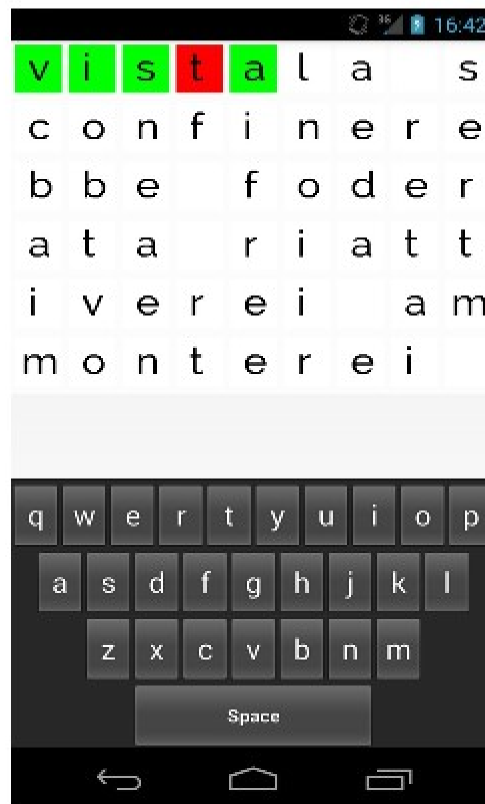


Figura 1.4 Schermata di gioco di NinjaTyping

I punteggi che un utente ottiene attraverso la schermata di gioco precedente, verranno inseriti in un *database locale* e saranno consultabili nella sezione delle statistiche.

Un utente in NinjaTyping può giocare sia nel caso in cui abbia effettuato la *registrazione* (indicando Username e Password) presso l'apposito *server*, ma anche nel caso non si sia registrato. Ho fatto questa scelta, per evitare che l'utente piuttosto di effettuare una registrazione, eviti di giocare disinstallando l'applicazione. D'altro canto però, un utente registrato, avrà l'opportunità di condividere i propri record sul server e a seconda del punteggio ottenuto, essere inserito nella lista dei 100 migliori punteggi ottenuti dai giocatori di NinjaTyping (lista visualizzabile anch'essa nella sezione statistiche dell'app). Registrarsi permette inoltre di poter sfidare altri utenti online in partite a turni.

Un'altra *feature* importante di questa applicazione, è la possibilità di condividere i propri punteggi, sulla bacheca del S*ocial Network* più “popolato” al mondo, *Facebook.*

Il lato *Social,* rappresenta a mio parere un aspetto fondamentale nella vita e soprattutto nel successo di un'applicazione. Non è da sottovalutare infatti quello che io chiamo *Spam inconscio,* cioè la possibilità che l'utente pubblicizzi l'applicazione di uno sviluppatore in modo totalmente inconscio, utilizzando cioè funzioni create per mostrare risultati personali.

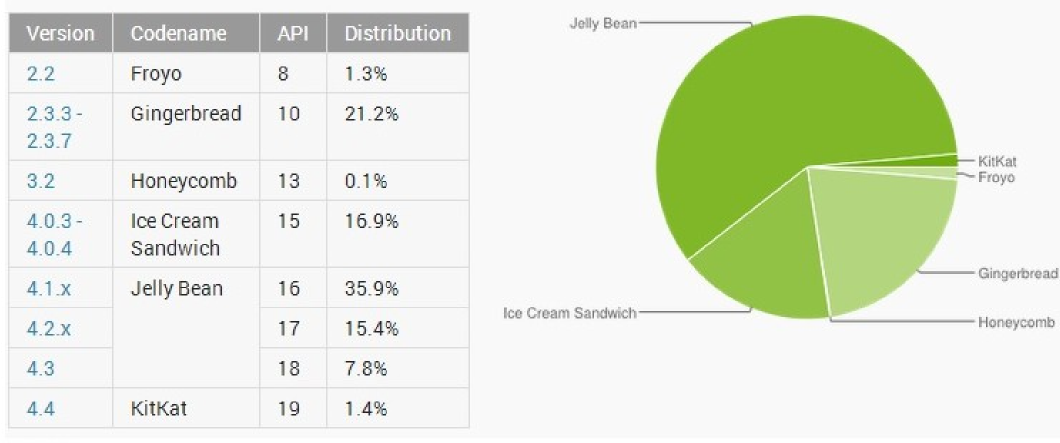
Questa era solo una breve panoramica delle principali funzioni di NinjaTyping, nei capitoli successivi procederò con la progettazione e implementazione completa del progetto che si sviluppa sia su lato Client (la parte più "visibile") ma anche lato server (la meno visibile ma altrettanto importante). Prima però mostrerò nell'ultimo paragrafo di questo capitolo il perché di alcune precise scelte d'implementazione del software sviluppato.

1.5 NinjaTyping, dispositivi supportati

NinjaTyping è una tipologia di gioco che esiste (con un discreto successo) da molto tempo per Personal Computer. Ovviamente tra NinjaTyping e i giochi simili sviluppati per PC, esistono notevoli differenze che portano a volte dei vantaggi, mentre altre degli svantaggi.
Ad esempio un ipotetico NinjaTyping per Pc, consente sicuramente partite più longeve e con un più alto grado di sfida, grazie al più ampio schermo e quindi alla possibilità di poter generare frasi molto lunghe, che richiedono più tempo per essere "terminate". Su NinjaTyping un utente molto veloce può riuscire a portare a termine una partita anche in meno di 10 secondi. I terminali sui quali verrà giocato NinjaTyping, hanno schermi ridotti che nel migliore dei casi arriva a poco più di 10". Quindi come un piccolo Netbook, con la differenza che gran parte dello schermo sarà occupato dalla tastiera "virtuale", in quanto la quasi totalità dei dispositivi mobili sono sprovvisti di tastiera fisica.
Siccome l'implementazione del progetto prevede sfide dirette (challenge online) e indirette (condividere il proprio record sul server), ho deciso per correttezza e parità di opportunità tra gli utenti, di creare partite in cui la lunghezza della frase da digitare è

standard. Quindi un Tablet e uno Smartphone avranno a schermo lo stesso numero di caratteri da digitare. Nello stesso tempo, ho dovuto prendere però decisioni per così dire "dolorose". Voglio che NinjaTyping sia un'esperienza piacevole, non snervante e per questo non ho reso disponibile l'applicazione per tutti i modelli che utilizzano il sistema operativo Android. NinjaTyping sarà giocabile solo da utenti possessori di un dispositivo con uno schermo maggiore o uguale a 4".

Un'altra scelta che ho voluto fare è stata quella di sviluppare il software solo per dispositivi che hanno una versione del sistema operativo superiore alla 4.0.
Prima di giustificare tale scelta, faccio una piccola premessa. Da quando nel 2008 Android è stato rilasciato (*versione 1.0*), gli aggiornamenti dello stesso atti a migliorarne le prestazioni ed eliminare i bug delle versioni precedenti sono stati molti. Oggi siamo arrivati alla *versione 4.4 (KitKat)*, ma non tutti gli aggiornamenti del sistema operativo vengono installati sui dispositivi già presenti sul mercato. Questo vuol dire che sul totale dei Tablet/Smartphone, ci sarà una percentuale con una determinata versione di Android, un'altra percentuale con un' altra versione e così via. Quanto appena detto è traducibile e riscontrabile nella *figura 1.5* che riporta dati ufficiali rilasciati da Google.



Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	21.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.9%
4.1.x	Jelly Bean	16	35.9%
4.2.x		17	15.4%
4.3		18	7.8%
4.4	KitKat	19	1.4%

Figura 1.5 Suddivisione delle varie versioni di Android nella totalità dei dispositivi mobili

Come si può notare, ad ogni versione di Android, corrisponde una nuova versione delle *API*.
Le *Application Programming Interface,* sono librerie messe a disposizione dello sviluppatore generalmente all'interno di un *SDK (Software development kit)* cioè un

pacchetto utile per lo sviluppo di software (applicazioni in questo caso).
Se uno sviluppatore sceglie di sviluppare software, utilizzando *feature* introdotte in API recenti, andrà incontro a un lavoro oneroso in termini di tempo nel caso in cui voglia garantire la retrocompatibilità (non sempre possibile però) con dispositivi che si basano su *API Level* inferiori. Di contro se uno sviluppatore non vorrà garantire retrocompatibilità, sarà costretto a rinunciare a un determinato bacino d'utenza.
Io ho scelto di sviluppare utilizzando le *API Level 15*, rivolgendomi quindi a quei dispositivi sul quale è installata una versione uguale o superiore alla *4.0 (Ice Cream Sandwich).* Questo vuol dire che NinjaTyping non potrà essere installata e utilizzata da dispositivi con una versione del Sistema Operativo antecedente alla 4.0. Questa è stata una scelta pensata proprio osservando le statistiche di *figura 1.4.* Infatti così facendo andrò a coprire un bacino d'utenza molto elevato, pari quasi all' 80% dei dispositivi Android!

Dopo questo capito in cui ho parlato dei concetti fondamentali che circondano l'applicazione NinjaTyping, andrò a descrivere nel dettaglio la progettazione del *software* sviluppato.

Capitolo 2
Progettazione dell'applicazione

Dopo le informazioni per così dire "astratte" che ho riportato nel *Capitolo 1,* riguardo NinjaTyping e in generale il mondo *mobile*, mi concentrerò su qualcosa di più concreto. Illustrerò infatti attraverso opportuni schemi, la progettazione che sta alla base dello sviluppo del mio *software*.
Procederò inizialmente con un'analisi dei requisiti (e un diagramma dei casi d'uso) per indicare *cosa* può fare l'utente attraverso l'applicazione, ma successivamente affronterò la progettazione dividendola in varie sezioni, quali ad esempio la parte *client* riguardante le varie schermate di gioco, la parte *client-server* riguardante la sfida online, la parte di gestione dei database e infine la parta grafica in cui progetterò un *layout* in modo che sia usabile e portabile su tutti i dispositivi di diverse dimensioni di schermo.
Premetto che non tutto quello che ho progettato potrebbe servire nell'immediato sviluppo. La progettazione guarda più avanti e struttura il *software* in modo da prevedere anche eventuali sviluppi futuri.

2.1 Analisi dei requisiti

Inizierò il capitolo sulla progettazione presentando quello che in gergo tecnico è chiamato *documento dei requisiti.* L'utilità di questo paragrafo è spiegare lo scopo della mia applicazione dalla prospettiva dell'utente, indicando quindi *cosa* essa debba fare, in maniera più dettagliata e ad ampio raggio rispetto al capitolo precedente. Evidenzierò in grassetto tutti quei concetti che costituiscono la struttura fondamentale dell'applicazione e che verranno approfonditi nel corso della stesura della tesi.
L'applicazione NinjaTyping comincerà con una schermata di presentazione iniziale detta "*splash*", che avrà il solo scopo di caricare determinati dati e non sarà interattiva per l'utente, nel senso che comparirà a schermo per pochi secondi e scomparirà

automaticamente. Successivamente sarà visibile la *Home* (o menù principale) di NinjaTyping, che serve a direzionare l'utente in precise sezioni dell'applicazione. Attraverso la Home*,* sarà possibile accedere a 4 sezioni principali quali *gioca partita singola, gioca partita online, visualizza statistiche e accedi.*

Descriverò brevemente ogni singola sezione:

- **Accedi:** L'utente, attraverso questa sezione, avrà la possibilità di creare un proprio *account NinjaTyping* personale, che sarà essenziale per accedere a determinate funzioni dell'applicazione, ma non sarà in alcun modo invasiva. Per invasiva intendo dire che l'utente non dovrà fornire nessun dato personale, nessun indirizzo e-mail o altro per poter effettuare la registrazione. Dovrà solamente indicare uno *username* e una *password*, registrandosi presso il ***server dedicato*** in pochissimi secondi. L'utente potrà, una volta registrato, effettuare una *login* (un'autenticazione) attraverso altri dispositivi mobili sul quale è installata l' app NinjaTyping, senza creare un altro account fornendo quindi nuovamente password e username.
- **Gioca partita singola:** l'utente accedendo a questa sezione, potrà giocare una partita singola (che negli sviluppi futuri potrebbe dividersi in sottosezioni, come ad esempio in livelli di difficoltà) attraverso l'apposita ***schermata di gioco*** e dopo aver concluso il *match*, arriverà nella sezione ***riepilogo partita*** che ha una doppia funzione. La *prima funzione* è quella di mostrare all'utente l'esito della partita mostrando quindi dati quali "punteggio", "errori", "tempo di gioco". Questi dati, saranno nello stesso tempo memorizzati automaticamente, in un database locale che sarà consultabile attraverso la sezione statistiche. La *seconda funzione* si occupa per così dire dell'aspetto *social.* Infatti i dati appena memorizzati nel database locale, saranno inviati (ma solo in alcuni casi memorizzati) anche al database presente sul *server dedicato*, ma solo nel caso l'utente abbia effettuato la registrazione presso quest'ultimo. Inoltre l'utente potrà scegliere, se condividere o meno sul social network Facebook, i dati della partita giocata. La *condivisione su Facebook* sarà possibile anche in caso l'utente non abbia effettuato la registrazione presso il server dedicato all'applicazione.
- **Gioca partita online:** Questa sezione è accessibile solo se l'utente ha creato un

proprio account effettuando la registrazione presso il server dedicato. Nel caso quindi l'utente sia registrato, potrà procedere a sfidare altri giocatori che utilizzano NinjaTyping. La sfida consiste prima di tutto nella ricerca di un avversario. Una volta trovato l'avversario, sempre attraverso questa sezione, l'utente potrà cercane un altro e allo stesso tempo giocare la partita con l'avversario già trovato in precedenza. E' sicuramente la parte più complicata a livello d'implementazione ed è anche la parte che in futuro subirà maggiori modifiche, novità e aggiustamenti.

- **Visualizza statistiche:** Come facilmente intuibile l'utente potrà visualizzare varie statistiche riguardanti NinjaTyping. Potrà controllare l'esito delle sfide online giocate contro altri utenti, potrà controllare i propri record personali e potrà inoltre controllare i migliori record (memorizzati sul server dedicato) effettuati d ai vari giocatori di NinjaTyping.

Ho usato molto spesso il termine *"server dedicato"*. Il server è un aspetto fondamentale dell'applicazione NinjaTyping. Compito del server sarà occuparsi della registrazione degli utenti, della gestione degli *score* effettuati dagli utenti nelle partite *single game* e del meccanismo che permette a due utenti di "trovarsi" e "scontrarsi", registrando sul database l'esito finale della partita.
Vorrei ora offrire un contributo grafico, per fissare meglio parte dei concetti appena descritti. Nella *figura 2.1* un semplice *Use Case Diagram*, dalla vista dell'attore *Utente.*

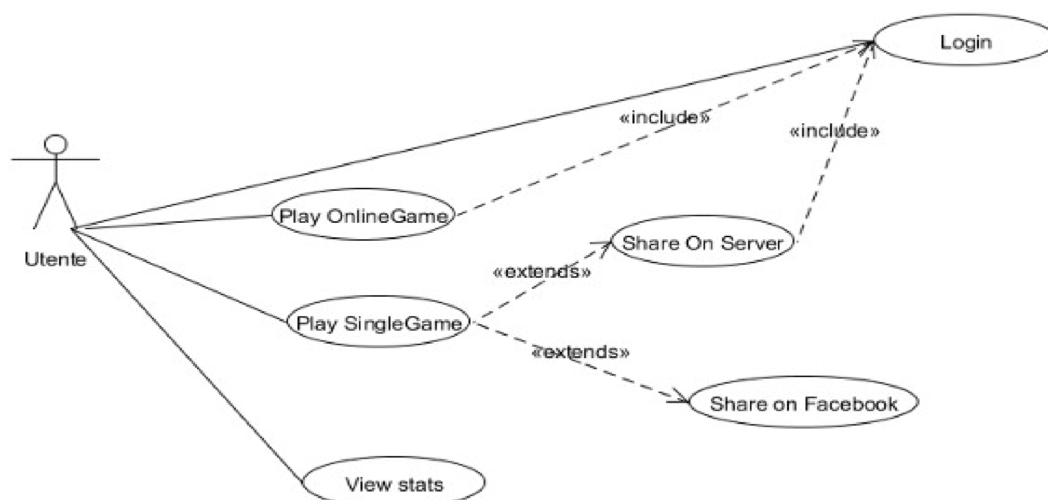


Figura 2.1 Diagramma dei casi d'uso

Come detto, l'utente può visualizzare le proprie statistiche di gioco, effettuare login, giocare una partita online (ma questo implica la *login* obbligatoria), giocare una partita singola scegliendo se condividere i propri punteggi su *Facebook* e inviare il punteggio al server (ma anche questo è possibile solo se si effettua login).
Dopo questa parte in cui ho evidenziato e fissato i principali concetti, procederò descrivendo singolarmente ogni *sezione di gioco* alla quale l'utente può accedere.

2.2 Schermata di gioco e partita

Uno dei concetti fondamentali di NinjaTyping è "giocare una partita". L'utente arriva nella sezione "*Gioca Partita*" attraverso il menù principale (*Home*), e dopo aver concluso una partita dovrà accedere automaticamente alla sezione "*Riepilogo partita*" che sarà argomento del prossimo paragrafo. Per giocare una partita, ho bisogno di implementare diverse componenti.
Ho bisogno prima di tutto di definire una classe (la chiamerò *GameView*) che contenga tutti quei componenti utili per giocare la partita. La *GameView* dovrà contenere in un campo di tipo *String,* la frase che l'utente dovrà digitare. Questa frase dovrà essere creata in maniera random ogni volta che l'utente decide di giocare una nuova partita. Per comporre questa frase mi avvarrò dell'uso di una classe che chiamerò *Dictionary,* che avrà lo scopo appunto di restituire una frase casuale composta da parole estrapolate da un dizionario. Ho detto che l'utente dovrà digitare questa frase e per farlo avrà bisogno di una tastiera quindi di un oggetto di tipo *Keyboard.* Per concludere, devo considerare anche la parte grafica. La frase che otterrò attraverso l'oggetto *Dictionary,* dovrà essere disegnata su schermo, ma soprattutto dovrà essere interattiva. Per interattiva intendo dire che quando l'utente digiterà un tasto della tastiera fisica, io dovrò colorare il carattere corrispondente (della frase disegnata su schermo), di rosso (se l'utente ha commesso un errore) oppure di verde (se il carattere è stato correttamente digitato). Per fare questo dovrò quindi gestire ogni singolo carattere della frase e per farlo utilizzerò una classe appositamente creata chiamata *WordCell*. Quindi avrò all'interno di *GameView,* tanti oggetti *WordCell* dipendenti dalla lunghezza della frase. La lunghezza della frase sarà fissa. Voglio decidere a priori quanti caratteri devono essere presenti su

schermo e nel paragrafo dedicato al Layout della schermata Gioca partita singola, spiegherò meglio il perché di tale scelta. La *figura 2.2* aiuta a fissare meglio questi concetti.

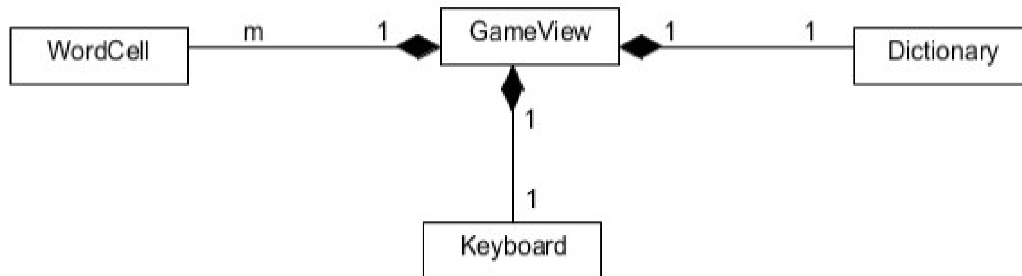


Figura 2.2 Progettazione della schermata di gioco

Attraverso questo insieme di classi potrò giocare quindi una partita. La partita è da considerarsi come un insieme di dati che descrivono la prestazione appena effettuata dall'utente. Quindi un oggetto *partita* dovrà tenere conto del numero di **errori** commessi, del **tempo** di gioco trascorso, del **punteggio** finale e della **frase** che ho digitato per ottenere un determinato punteggio. Tutti questi dati saranno attributi quindi di una classe che chiamerò *GameRecord.* La classe GameRecord è rappresentata in *figura 2.3*.

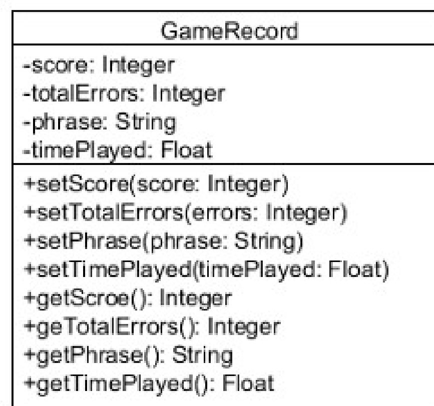


Figura 2.3 La classe GameRecord

Quando la partita sarà conclusa, il controllo del gioco passerà dalla classe GameView alla classe GameRecap.

2.3 Riepilogo di gioco e condivisione dati partita

Il GameRecap o *Riepilogo Partita*, è una schermata che ha un'importanza da non sottovalutare. Infatti non avrà solo il compito di mostrate i dati contenuti nell'oggetto *Partita,* bensì dovrà anche occuparsi di interagire con il *database locale,* il *server dedicato* e *il server di Facebook.* L'utente accede a questa schermata nel momento in cui viene terminata la partita giocata (*figura 2.4*) grazie alle classi precedentemente definite.

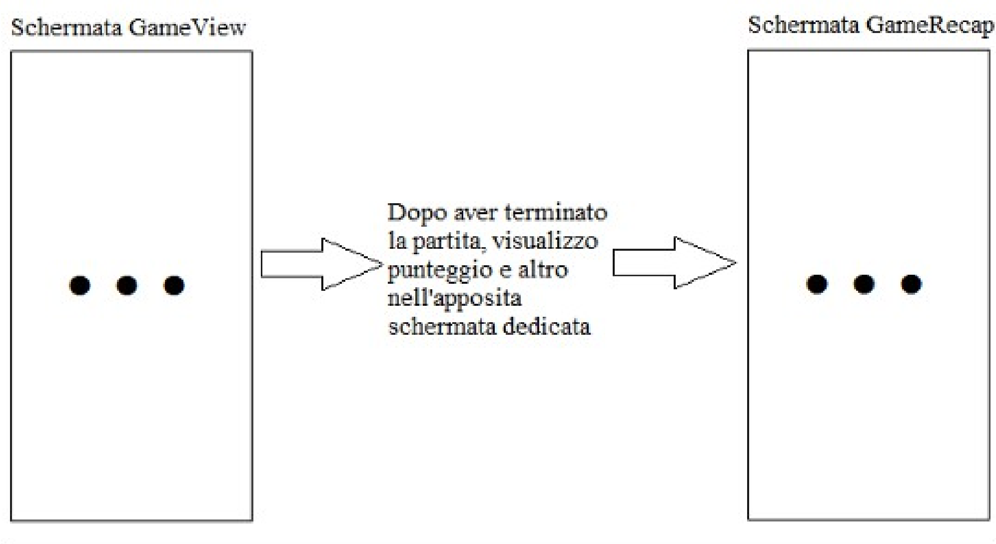


Figura 2.4 Situazione immediatamente successiva alla terminazione di una partita di gioco

L'oggetto *Partita (GameRecord),* definito nel precedente paragrafo, dovrà essere memorizzato nel database locale (quindi creerò un'apposita classe *MyDatabase* per questo), nell'apposita tabella. In questo modo, quando l'utente accederà alla sezione statistiche, provvederò a recuperare i dati inseriti nel database e mostrarli all'utente attraverso una lista di oggetti *GameRecord.*

Oltre ad inserire i dati nel database, provvederò attraverso un'apposita interrogazione, a verificare se il punteggio partita appena ottenuto dall'utente, è anche il migliore che l'utente ha ottenuto da quando gioca a NinjaTyping. In sintesi, se dopo aver giocato la partita l'utente ha effettuato il suo miglior punteggio, attraverso la classe *GameRecap* provvederò a notificarglielo. Oltre a questo, se l'utente ha effettuato registrazione

mediante l'apposita sezione, il record verrà condiviso presso il server dedicato. Infine, come detto nell'analisi dei requisiti, l'utente potrà scegliere se condividere questo oggetto partita (*GameRecord*) sul proprio account Facebook. Quanto detto è espresso in forma schematica nel diagramma delle attività in *figura 2.5*

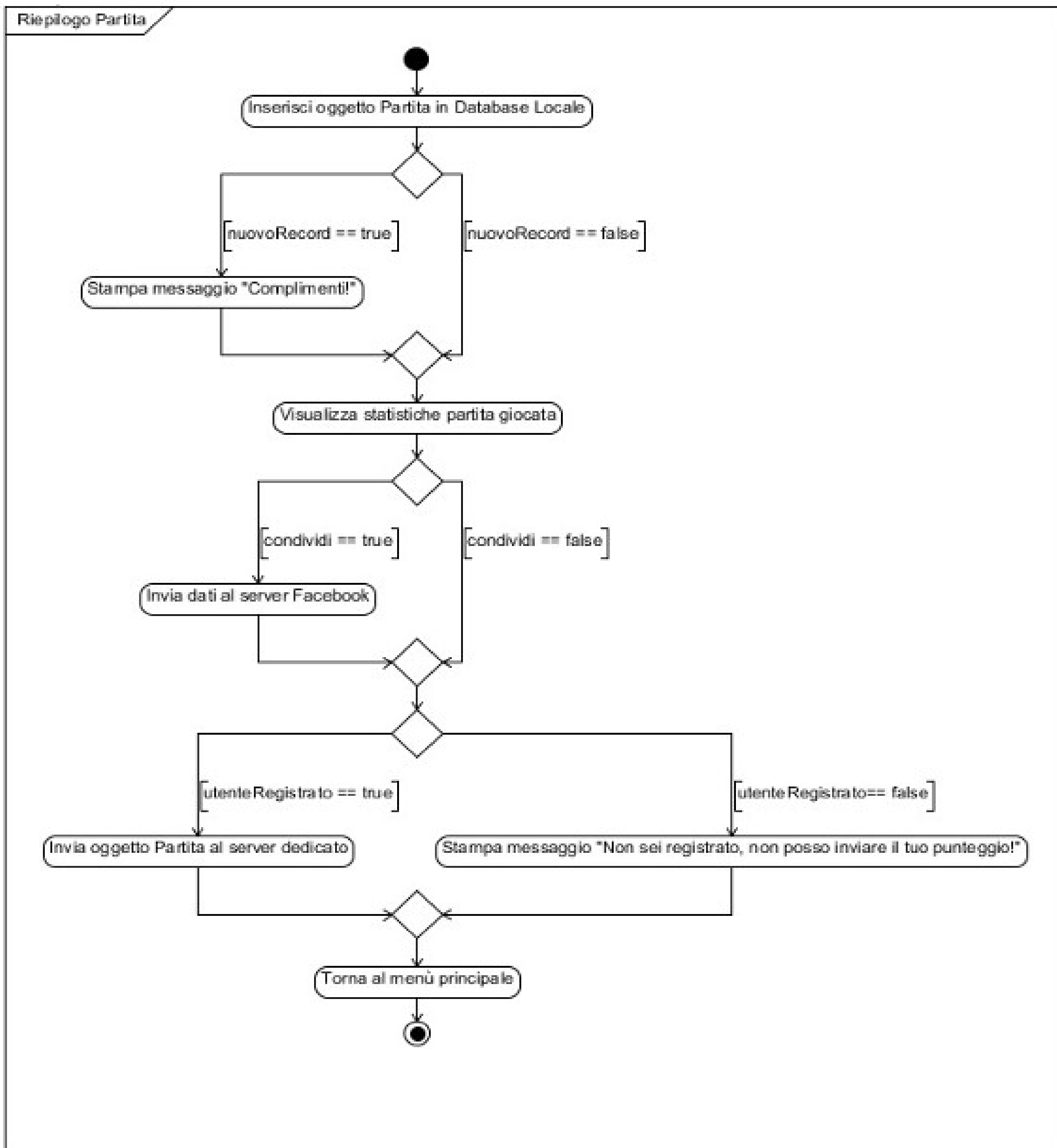


Figura 2.5 Diagramma delle attività in Riepilogo Partita

Ovviamente in questa schermata l'utente avrà la possibilità di tornare al menù principale ed effettuare un'altra tra le scelte evidenziate nell'analisi dei requisiti. Nel prossimo

paragrafo procederò con la progettazione della sezione che si occupa di permettere all'utente di giocare partite online.

2.4 Gestione partite online

Se l'utente ha effettuato l'accesso/registrazione presso il server dedicato, potrà giocare una partita contro un avversario casuale, accedendo alla schermata di gioco *Multiplayer,* rappresentata in *figura 2.6.*

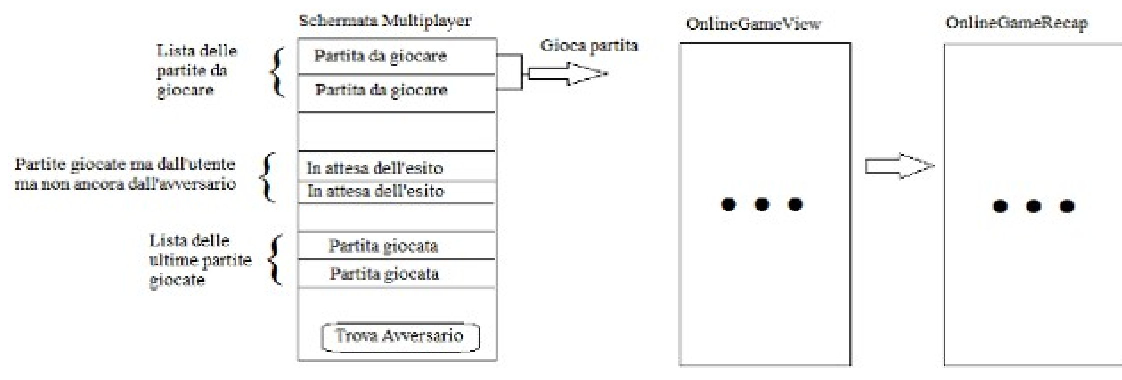


Figura 2.6 Schermata Multiplayer

In questa schermata, voglio che l'utente possa avere la possibilità di:

-Trovare un avversario, che è sinonimo di trovare una partita da giocare. Come il sistema si occuperà di trovare l'avversario verrà spiegato nell'apposita sezione.

-Giocare le partite precedentemente trovate (rappresentate in *figura 2.6* da "lista delle partite da giocare"). L'utente avrà la possibilità di giocare tali partite accedendo alla schermata OnlineGameView.

-Conoscere gli esiti delle ultime partite giocate (rappresentate in *figura 2.6* da "lista delle ultime partite giocate").

-Visualizzare la lista delle partite giocate, ma in attesa dell'esito.

Mi rendo conto che "partita in attesa dell'esito" e "partita da giocare" siano concetti nuovi che non ho ancora introdotto. Questi due concetti sono da attribuire alla modalità di gioco online. Mi spiego meglio. Una partita online è da intendersi come una semplice

partita a turni. Questo vuol dire che due utenti, giocheranno una stessa partita, in momenti temporali differenti e di conseguenza, ci potranno essere momenti in cui una partita giocata da un utente, dovrà ancora essere giocata da un altro e viceversa. Questo introduce un altro fondamentale concetto, cioè di una partita che ha degli stati. Per spiegare cosa intendo e come sarà composta una partita online, mi avvalgo dell'aiuto della *figura 2.7* in cui mostro la classe GameOnline comprensiva degli attributi fondamentali.

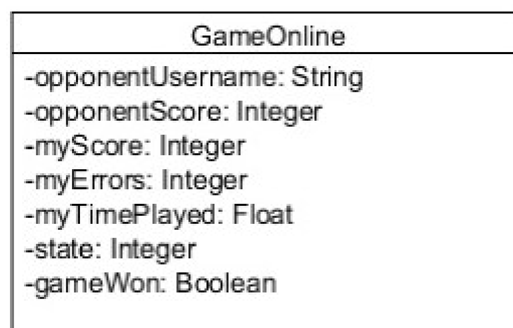


Figura 2.7 Classe GameOnline

L'online game dovrà contenere il punteggio effettuato dall'utente, il punteggio effettuato dall'avversario, il nome dell'avversario, l'identificativo univoco della partita e lo **stato** della partita. Come ho detto, è una partita a turni. Questo comporta che una partita possa assumere tre diversi stati così suddivisi:

- state = Played.
- state = Played_both.
- state = Not_played.

La partita deve necessariamente avere uno (e solo uno) di questi tre stati. Se il campo *state* avrà valore “Not_Played”, vuol dire che l'utente ha trovato un avversario, la partita è stata creata, ma l'utente non ha ancora giocato la partita. Con *state* “Played” indico che l'utente ha giocato la partita, ma è in attesa di conoscere il risultato dell'avversario. Con *state* “Played_both” vuol dire che la partita si è conclusa, entrambi i giocatori hanno giocato e quindi ne conosco l'esito finale.

Come facilmente intuibile, le partite con stato “Not_played” passeranno allo stato

"Played", nel momento in cui l'utente gioca la partita attraverso la schermata di gioco *OnlineGameView.* Le partite con stato *Played* invece passeranno allo stato *Played_both* nel momento in cui arriverà l'esito della partita giocata dall'avversario.
Ovviamente, la prima volta che l'utente accede a questa sezione, non vi sarà nessun oggetto *GameOnline* con dati da esporre. Man mano che l'utente troverà avversari con cui scontrarsi (vedi *paragrafo 2.9*), verranno creati oggetti di tipo *GameOnline* (i quali dati verranno presi dal database locale). Gli attributi come ad esempio *myScore* dell'oggetto *GameOnline,* saranno inizialmente nulli e verranno inizializzati quando un utente giocherà una tra le partite online che ha trovato. Per giocare una di queste partite online, verrà utilizzata la schermata (*classe*) *OnlineGameView* che è pressoché identica alla *GameView* già esposta nel *paragrafo 2.2*. La composizione della schermata (cioè la frase da digitare e la tastiera fisica da utilizzare) infatti sarà identica, sia per quanto riguarda una partita singola che una partita online. La partita online differisce quindi da una partita singola, principalmente per lo stato. Infatti una partita singola non ha stati, viene creata quando l'utente vuole giocare una partita e viene *completata* quando l'utente finisce di digitare la frase ottenendo un punteggio. La partita online invece, viene creata e in un certo senso lasciata in *pending.* La partita online può considerarsi completata o chiusa, solo nel momento in cui entrambi i giocatori hanno disputato il match, attraverso le rispettive schermate *OnlineGameView.*
La *figura 2.6* riportata a inizio paragrafo, mostra inoltre come dopo aver giocato una partita nella sezione online, l'utente avrà accesso alla sezione (classe) di *riepilogo partita online* (*Online Game Recap*). Questa classe è molto simile a quella che otteniamo dopo aver giocato una partita singola (*GameRecap*) definita nel *paragrafo 2.2,* ma differisce per alcuni particolari che spiegherò nel prossimo paragrafo.

2.5 Riepilogo partita online

Questa classe *OnlineGameRecap*, come detto, dovrà essere molto simile alla classe *GameRecap* della sezione *Partita Singola.* Ci sono però alcune differenze. Innanzitutto, il risultato della partita, non verrà salvato automaticamente nelle apposite tabelle del database locale. Infatti prima che ciò avvenga, è assolutamente necessario che i dati della partita giocata vengano inviati e salvati correttamente sul server nelle apposite

tabelle che gestiscono le partite online. E' assolutamente necessario per l'implementazione dell'algoritmo che utilizzerò in fase di sviluppo, che il database locale e il database del server abbia gli stessi dati riguardanti una determinata partita online, altrimenti potrebbero verificarsi dei pericolosi effetti collaterali. Quindi se i dati della partita online non verranno salvati correttamente sul server dedicato, allora il sistema deve annullare la partita giocata dall'utente. Praticamente la partita giocata dall'utente non passerà dallo stato *Not_Played,* allo stato *Played.* Di conseguenza l'utente sarà costretto a rigiocare la partita attraverso la *schermata Multiplayer.* Se invece i dati della partita saranno correttamente inviati e memorizzati sul server, allora lo stato cambierà da *Not_played* a *Played.* Quando anche l'utente avversario giocherà la medesima partita, memorizzando correttamente i dati sul server, allora lo stato passerà da *Played* a *Played_both* e l'utente potrà verificarne l'esito sia nella *schermata Multiplayer* che nell'apposita *sezione statistiche.* In *figura 2.8* cerco di fissare questi concetti attraverso un diagramma delle attività.

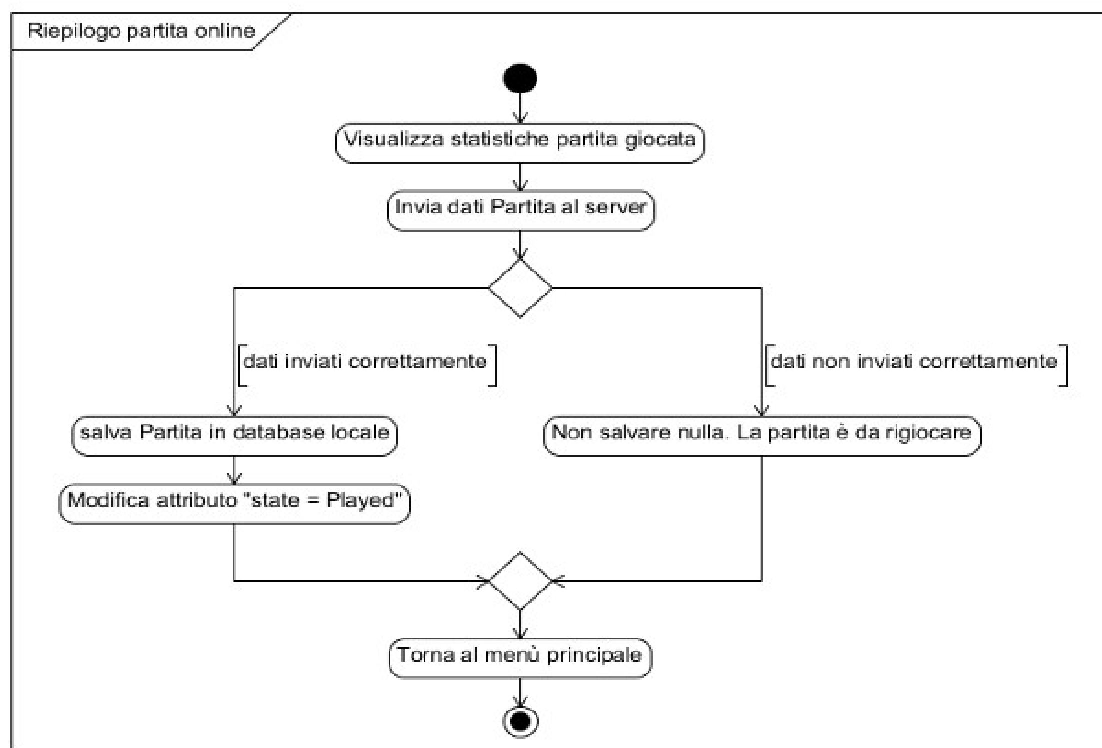


Figura 2.8 Diagramma delle attività Riepilogo partita online

Anche in questa schermata l'utente avrà la possibilità di tornare al *menù principale*

(*Home*) ed effettuare una tra le scelte precedentemente indicate nell'analisi dei requisiti. Nel prossimo paragrafo spiegherò quindi la parte probabilmente più semplice dell'intero progetto e cioè della classe che si occupa di mostrare i dati delle partite giocate dall'utente.

2.6 Sezione statistiche

Attraverso questa sezione dovrò mostrare all'utente i risultati conseguiti durante l'esperienza con *NinjaTyping.* Provvederò quindi a mostrare i 10 migliori record personali, i 100 migliori record mondiali, e tutti gli esiti delle partite online giocate, cioè gli oggetti *OnlineGame* che hanno stato *Played_both.* Nel caso in cui l'utente non abbia mai giocato una partita potrà visualizzare solo le statistiche riguardanti i migliori 100 record mondiali. Per quanto riguarda i record personali di ogni utente, che siano quindi record ottenuti giocando partite in *single game* oppure record ottenuti giocando partite *online*, dovrò estrapolare i dati (precedentemente salvati) dal database locale. Questi dati saranno inseriti in appositi oggetti descritti dalle classi in *figura 2.9* .

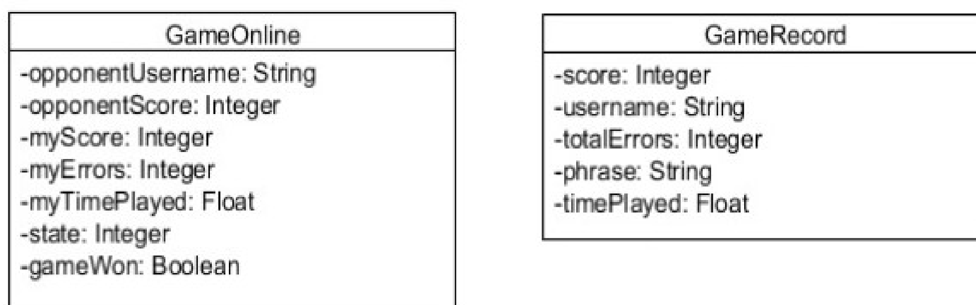


Figura 2.9 Classi dedite al recupero statistiche

Per quanto riguarda invece i migliori record mondiali, recupererò i dati dal server dove sono salvati i risultati e li visualizzerò attraverso oggetti di tipo GameRecord.
Nel prossimo paragrafo procederò alla descrizione della progettazione della creazione di un account e registrazione sul server.

2.7 Account NinjaTyping

Come ormai noto, NinjaTyping garantisce all'utente la possibilità di creare un account personale, registrandosi attraverso il server dedicato. Questa è stata una scelta quasi obbligata, in quanto per sviluppare e progettare determinate funzioni ho bisogno in qualche modo di identificare l'utente. Allo stesso tempo non ho reso questa *registrazione*, obbligatoria. Non voglio che l'utente sia costretto a fare nulla. Ovviamente se l'utente non si registra, non potrà sfruttare al meglio l'applicazione e quindi l'esperienza di gioco.
E' possibile registrarsi o *loggarsi* (se in passato si è già creato un account NinjaTyping) attraverso un'apposita schermata. Se l'utente si è già registrato in passato, basterà inserire il proprio *username* e *password* e inviarle al server. Il server effettuerà un controllo sul database e restituirà un valore positivo in caso di accesso eseguito con successo, oppure un valore negativo in caso contrario. Per gli utenti che invece non hanno mai eseguito la registrazione, la metodologia sarà comunque la stessa. Il server provvederà a dare una risposta affermativa o negativa all'utente, dopo aver controllato la validità dello *username,* ossia dopo aver verificato che nel database non si sia già iscritto in passato un utente con il medesimo username richiesto dall'utente.
La schermata di accesso sarà accessibile manualmente dalla *Home* dell'applicazione, oppure automaticamente nel caso l'utente cerchi di accedere alla sezione *Multiplayer.* Infatti ricordo che solo nel caso in cui l'utente abbia effettuato l'accesso/registrazione, potrà giocare partite online. I dati di accesso dell'utente dovranno essere memorizzati sul dispositivo e di conseguenza dovrò fare in modo che l'utente non possa più accedere alla schermata di accesso. Infatti su un singolo dispositivo sarà possibile eseguire l'accesso con un solo account. Questo non vieta però il fatto di poter accedere con lo stesso account su più dispositivi.

2.8 Server e memorizzazione dati

Ho usato spesso, nel corso della stesura di questo capitolo, il termine *Server dedicato.* Il progetto che ho sviluppato è infatti *client-server*, cioè un'architettura logica di rete a livello applicativo, in cui il server fornisce un qualche servizio ad un C*lient* che ne fa

richiesta. Nello specifico, il C*lient,* è l'applicazione lato mobile che gestita dall'utente, chiede determinate *funzioni* al server. Ad esempio quando un utente (C*lient*) vuole creare un account NinjaTyping, invia una richiesta al server che provvederà a soddisfarla.
Ma il server NinjaTyping come gestisce le registrazioni? Come gestisce gli oggetti *Partita (GameRecord)* che l'utente gli invia chiedendone la memorizzazione? Ovviamente gestisce queste *problematiche* avvalendosi di un database. In *figura 2.10* riporto lo schema ER riguardante la progettazione della gestione degli account e delle partite singole effettuate dall'utente.

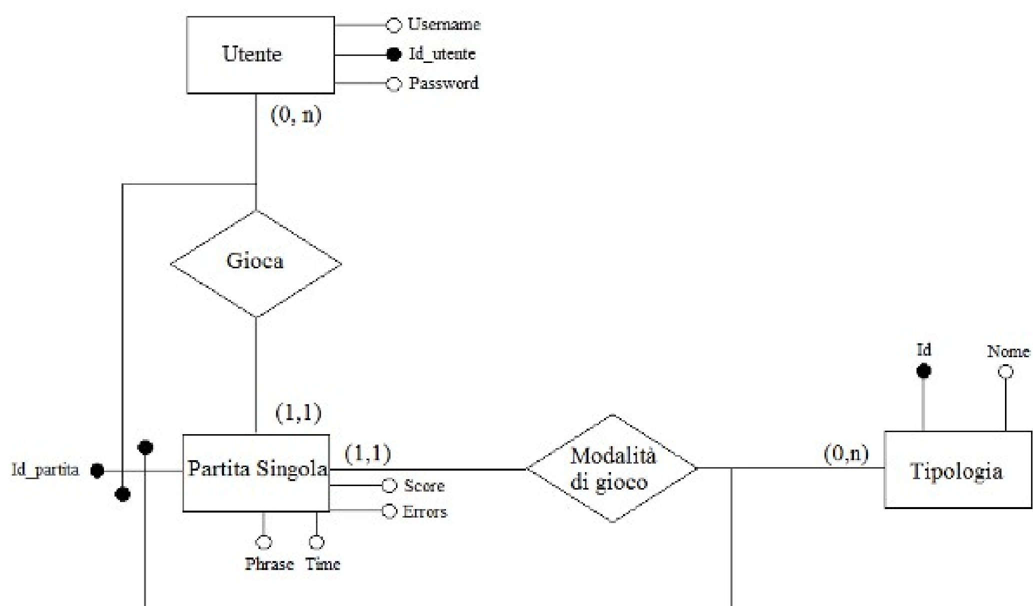


Figura 2.10 Schema ER parziale Database Server

Come si può notare, quando un utente si registra sul server, verrà memorizzato il suo account in un database attraverso uno username, una password e un identificativo univoco. Una partita singola contiene tutti quei dati che l'utente ottiene giocando una partita, quali gli errori commessi, il tempo di gioco trascorso (in una singola partita), la frase digitata per ottenere un determinato punteggio, lo score. Una istanza di partita singola, sarà identificata univocamente dall'id del giocatore che ha effettuato una determinata partita, dall'id della partita stessa e dall'id del tipo di partita giocata. Attenzione, per tipo di partita giocata, non intendo partita singola oppure multiplayer.

La partita multiplayer sarà trattata in maniera distaccata rispetto alla partita singola. Per *tipo* di partita, indico la modalità di gioco singola. Ad esempio potrei voler implementare più livelli di difficoltà a seconda della composizione della tastiera. Per esempio oltre a una normale partita con una normale tastiera *qwerty,* potrei decidere di permettere all'utente di giocare una partita singola con una tastiera *reversed*, cioè specchiata, quindi aumentando il grado di difficoltà. Quindi avrei un tipo di partita singola chiamata *modalità reversed.* Oppure potrei implementare una tastiera con numeri, apostrofi, punteggiatura garantendo un'altra modalità di gioco. Uso il condizionale in quanto potenzialmente potrei inserire un numero veramente elevato di tipologie differenti di gioco riguardanti la *partita singola,* ma ovviamente (all'inizio) solo poche saranno sviluppate.

Continuando con l'analisi del database, l'identificatore misto di partita singola, garantirà inoltre quei vincoli referenziali che sono in realtà già stati in un certo senso garantiti lato Client. Intendo dire che se un utente non è registrato presso il database non potrà giocare la partita singola in quanto il database non permetterà la creazione di partite singole in cui non siano indicati gli identificativi dell'utente.

Per quanto riguarda invece la memorizzazione dei dati lato Client (quindi fisicamente sul dispositivo mobile), anche in questo caso userò un database. Sul Client il database sarà sicuramente ridotto in termini di tabelle rispetto a quanto sarà presente sul server. Ad esempio non ci sarà la tabella *utente* dedicata alla registrazione, dato che su un dispositivo mobile un solo utente può loggarsi, quindi presumibilmente salverò *username e password* non in un database, ma magari in un file. Lato Client quindi mi avvarrò di tabelle che avranno l'unico scopo di memorizzare i "dati personali" dell'utente. Per dati personali intendo semplicemente i dati riguardanti le partite effettuate dall'utente, sia in single game che in online game. Semplicemente in fase di sviluppo quindi provvederò a trasformare in tabelle le entità proposte in *figura 2.11.*

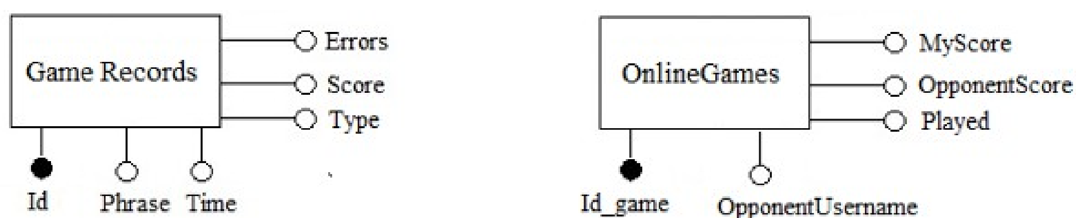


Figura 2.11 Tabelle lato client

Ricordo che non tutti i dati presenti sul dispositivo mobile saranno presenti sul database. Infatti se l'utente non ha effettuato la registrazione, i dati delle partite "*single game*" giocate non verranno inviati al server, ma saranno memorizzati sul dispositivo in modo che l'utente possa attraverso l'apposita sezione statistiche accedere ai propri record personali.
Non ci sarà invece nessuna tabella dedita alla memorizzazione dei migliori punteggi effettuati da tutti gli altri utenti che utilizzano NinjaTyping. Infatti come ho già accennato nella sezione dedicata alle statistiche, quando l'utente chiederà di visualizzare la lista dei migliori punteggi, effettuerò una richiesta al server che mi restituirà tale lista, ma una volta chiusa l'applicazione, questa lista sarà eliminata e per riottenerla dovrò effettuare nuovamente una richiesta al server.
Un'altra funzione che ha il server, è quella di gestire le richieste effettuate dai Client utenti, di giocare partite online. A questo procedimento teorico, dedicherò il prossimo paragrafo.

2.9 Algoritmo di ricerca di un avversario online

Procederò spiegando il meccanismo che permette a due utenti di trovarsi e scontrarsi in una partita online. Come ormai noto, l'utente attraverso il Client NinjaTyping, potrà cercare un avversario con il quale scontrarsi effettuando una richiesta al server. Il server dovrà gestire le richieste di gioco attraverso un'apposita tabella composta come in *figura 2.12*

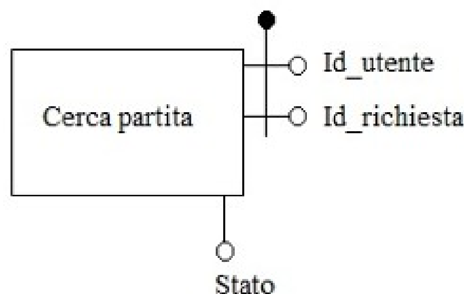


Figura 2.12

L'identificatore composto sta ad indicare che una richiesta dovrà essere identificata univocamente dalla coppia id_utente, id_richiesta. Questo non preclude certo all'utente di poter effettuare più richieste. L'attributo stato invece può assumere due valori, che in fase di sviluppo saranno numerici, ma per semplicità e comprensione darò loro in fase di progettazione un valore "nominale".

I valori che può assumere l'attributo stato, sono:

- stato = "in attesa"
- stato ="pronto".

Quando l'utente avrà "stato in attesa" vorrà dire che non è stato trovato un avversario, mentre se avrà "stato pronto" allora un avversario è stato trovato e si può procedere al giocare la partita. La *figura 2.13* aiuta a capire meglio quanto detto.

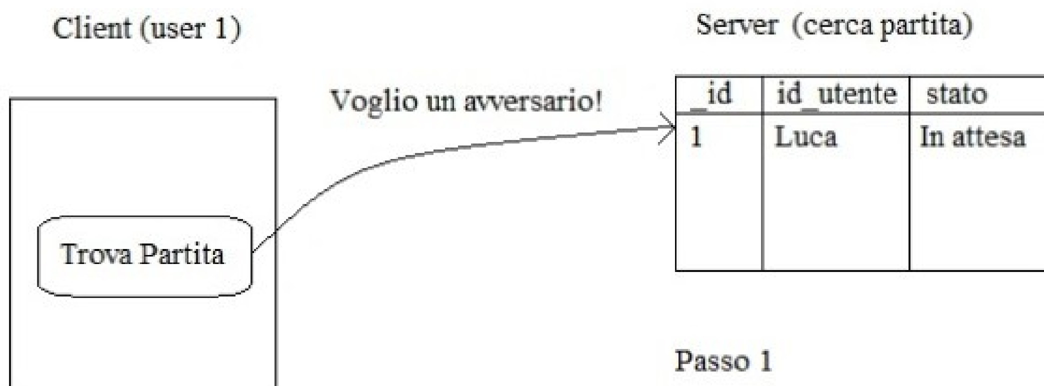
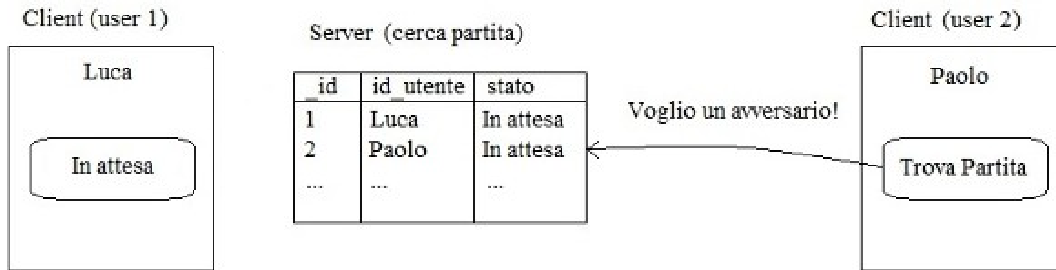


Figura 2.13 L'utente chiede al server un avversario

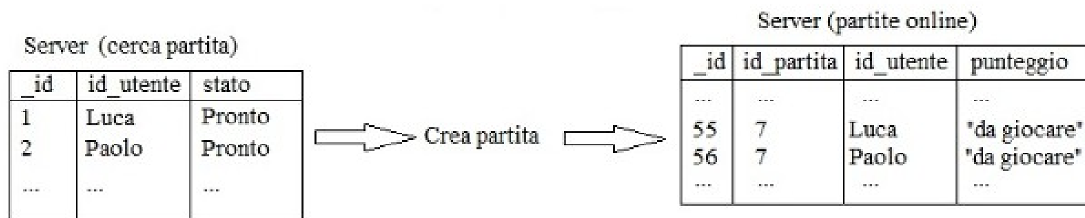
La figura simula (e ovviamente semplifica) attraverso il passo 1, una richiesta di un utente che vuole giocare una partita online. Appena arriva la richiesta, il server procede ad inserire tale richiesta nella tabella del database con stato "in attesa". Al posto di un identificativo numerico ho inserito un nome di persona esclusivamente per favorire la comprensione del procedimento. In fase di sviluppo non sarà così.

Nel passo 2 riportato in *figura 2.14*, un altro utente chiede al server di poter giocare una partita. Anche in questo caso il server procede inserendo la richiesta nella tabella del database e dopo aver rilevato che vi è già un altro giocatore in attesa, procede (passo 3) creando la partita mediante l'apposita tabella. Come si può osservare sempre in *figura 2.14* il campo *id_partita* della tabella *partite online* contiene lo stesso valore (del tutto indicativo) per entrambi gli utenti. Il valore di questo campo verrà assegnato grazie al

campo “stato” della tabella “cerca partita”. Come effettivamente questo avverrà, sarà spiegato in modo dettagliato nel capitolo dedicato allo sviluppo.



Passo 2



Passo 3

Figura 2.14

Dopo aver creato la partita, il server notificherà ai due Client (passo 4 in figura 2.15) che un avversario è stato trovato e la partita può quindi essere giocata (attraverso la *schermata multiplayer* del Client progettata nel paragrafo 2.4). Il server (come vedremo in fase di sviluppo) provvederà a fornire ad ogni Client il nome dell'avversario contro il quale dovrà giocare la partita e l'identificativo della partita stessa.

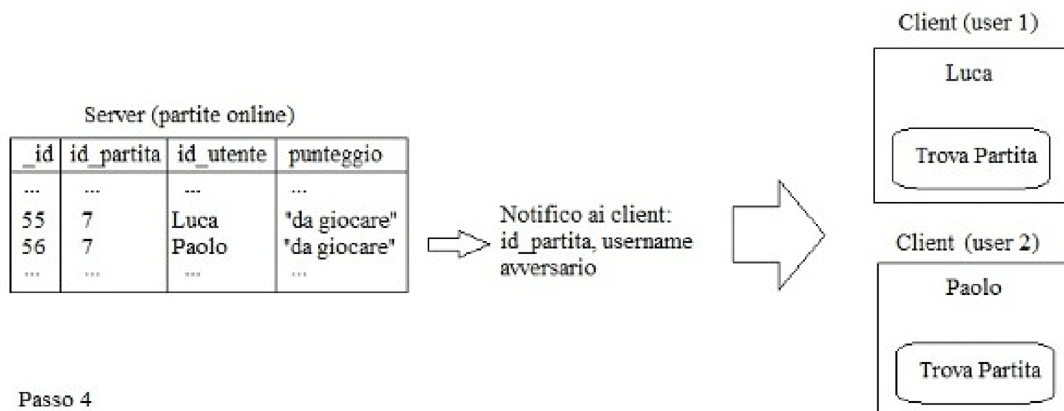


Figura 2.15

La partita è quindi stata creata, ed entrambi i Client sfidanti ne hanno ricevuto notifica. Ora gli utenti potranno procedere giocando la partita, oppure potranno effettuare un'altra domanda al server (tornando quindi al *passo 1*), creando così una lista di partite da giocare.
Infatti ricordo che in NinjaTyping si potranno avere più partite in *pending* (in attesa di essere giocate), ma l'utente non potrà effettuare contemporaneamente più richieste del tipo espresso nel *passo 1* per una mia precisa scelta di progettazione e sviluppo.

2.10 Layout della schermata di gioco

Il Layout dell'applicazione rappresenta una parte fondamentale per il successo della stessa. Ho cercato di usare semplici linee guida per rendere l'esperienza piacevole. Ad esempio nella scelta dei colori delle varie schermate ho utilizzato spesso il rosso, il verde e il bianco, che saranno quindi preponderanti nell'applicazione. E' stata una scelta precisa in quanto nella schermata di gioco (GameView), quando un utente sbaglierà a digitare il carattere, vedrà quest'ultimo colorarsi di rosso, mentre se l'utente digiterà il carattere giusto allora sarà colorato di verde. Questi però sono discorsi certamente importanti, ma un po' futili nell'ambito di una progettazione del software. Quindi non andrò oltre nel descrivere il design dell'applicazione, ma mi concentrerò sulla parte fondamentale del Layout per importanza e per difficoltà. Sto parlando appunto della schermata di gioco, quella che ho chiamato GameView in fase di progettazione. Ricordo che l'applicazione sarà eseguita su differenti dispositivi mobili, con una risoluzione dello schermo differente e anche una grandezza espressa in pollici differente. Infatti esistono Smartphone da 4" ma anche Tablet da 10". La schermata di gioco però nonostante le diverse dimensioni e risoluzioni degli schermi, dovrà essere sempre la stessa (ovviamente in proporzione). Tutti gli utenti dovranno giocare digitando lo stesso numero di caratteri, non voglio infatti che un utente con uno schermo più grande, possa digitare più caratteri rispetto a un utente con lo schermo più piccolo. Non sarebbe corretto in ambito di calcolo del punteggio. La progettazione di questa parte è piuttosto semplice, la vera difficoltà sarà in sede di sviluppo.
Tutti i dispositivi (Tablet e Smartphone) dovranno mostrare la *GameView verticalmente*,

che in gergo tecnico si dice *Orientation Portrait.* La GameView ricordo dovrà essere composta da una frase suddivisa per righe, e una tastiera. La frase dovrà occupare metà dello schermo, mentre la tastiera andrà ad occupare (se necessario) l'altra metà. In *figura 2.16* è possibile riscontrare quanto appena detto.

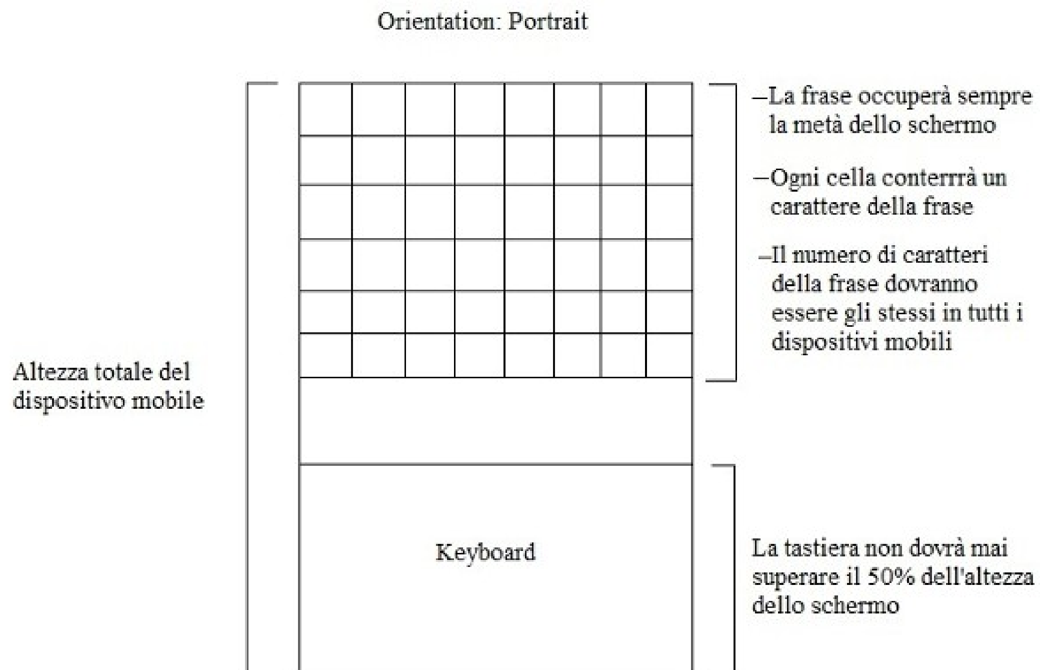


Figura 2.16 Progettazione del layout grafico della partita

Ogni rettangolo nella parte alta della figura, rappresenta quello che nel *paragrafo 2.2* ho denominato oggetto *WordCell.* Infatti ribadisco che la frase da disegnare su schermo, in fase di sviluppo sarà in un cero senso “scomposta”, e trattata quindi come un insieme di caratteri, che saranno contenuti appunto in una serie di oggetti WordCell. Il numero di caratteri che compongono la frase, saranno probabilmente compresi nel range *9*6* (9 righe e 6 colonne) oppure *9*7,* quindi 54/63 caratteri. Questo perché credo dia un'esperienza accettabile, garantendo un buon numero di caratteri da digitare e rendendoli assolutamente facilmente leggibili su qualsiasi dispositivo.
Come poi effettivamente riuscirò a fare quanto detto, verrà spiegato nel capitolo dedicato allo sviluppo.

Con quest'ultimo paragrafo, si conclude il *Capitolo 2* dedicato alla progettazione. Ho

tentato di tracciare le linee guida che serviranno nell'implementazione dell'applicazione vera e propria. In fase di sviluppo come spesso succede, alcune cose potrebbero cambiare in base al tipo di linguaggio/piattaforma che si utilizza per sviluppare il software. Molte componenti che ho definito sommariamente saranno in un certo senso specializzati, mentre altri ridotti in caso di necessità di ottimizzazione del software.

Capitolo 3
Implementazione dell'applicazione

Dopo aver affrontato nel capitolo precedente la progettazione riguardante NinjaTyping, è il momento di esporre nel dettaglio lo sviluppo che sta alla base dell'applicazione.
Sarà quindi un capitolo molto tecnico, in cui riporterò frammenti di codice e *Screenshot* dell'applicativo.
L'applicazione prevede uno sviluppo sia lato *Client* che lato *Server.* Come espressamente dichiarato nel primo capitolo, NinjaTyping sarà un applicativo diretto a dispositivi mobili, aventi il sistema operativo *Android.* Per sviluppare NinjaTyping, ho utilizzato l'ambiente di sviluppo *Android SDK* con l'*IDE* *Eclipse.* Le parti dinamiche dell'applicazione sono state scritte utilizzando il linguaggio di programmazione *JAVA,* mentre le parti statiche in *XML* (questa è una dualità che caratterizza tutte le applicazioni Android). Per quanto riguarda la parte Server ho utilizzato il linguaggio *PHP* e per la gestione del database *MySQL.*
Lo sviluppo dello scheletro del software, sarà fedele a quanto riportato nell'ambito della progettazione. Alcuni dettagli riguardo implementazioni di determinate sezioni e algoritmi, possono invece subire sostanziali modifiche, dettate dalla necessità di efficienza richieste da un dispositivo mobile.

3.1 Schermata di Splash e Home iniziale

Prima di procedere con questo primo paragrafo, vorrei fare una premessa notazionale. Un' attività (*Activity*) in Android è da intendersi sostanzialmente come una schermata dell'applicazione, definibile graficamente attraverso appositi Layout grafici dichiarati in fil .xml. Le attività vengono create come oggetti di classe *Activity* (definite nelle API LEVEL 1), da cui si ereditano proprietà e metodi. Quindi tutte le schermate principali della mia applicazione, avranno un nome (che sarà il più possibile descrittivo) composto

dal termine "Activity", ad esempio "MainMenùActivity", "GameActivity" e così via. La prima schermata visualizzata, quando l'utente preme l'icona NinjaTyping sul proprio dispositivo, è detta *SplashActivity*. Questa attività ha il compito principale di caricare in *background* determinati dati, che saranno utili nel prosieguo dell'esperienza applicativa. I dati in questione riguardano il dizionario di italiano (o inglese), che sarà caricato come risorsa dalla cartella *assets,* che può essere considerata come un piccolo *file system* accessibile in lettura dall'applicazione. Il caricamento di questa risorsa avverrà grazie ad un metodo statico della classe *Dictionary.* Tale classe è stata creata appositamente per gestire il dizionario ed estrapolare grazie ad un apposito metodo, la frase random che popola la schermata di gioco di NinjaTyping. La classe *Dictionary* sarà appositamente trattata in un paragrafo dedicato. Tornando alla *SplashActivity,* il processo di caricamento del dizionario (unito ad altre funzioni che potrei aggiungere in sviluppi futuri), può richiedere qualche secondo. Approfitterò di questo breve lasso di tempo, per proporre a tutto schermo un'immagine di benvenuto, con la scritta NinjaTyping e la *mascotte* dell'applicazione. In *figura 3.1* è riportato lo *Screenshot* della *SplashActivity.*

Figura 3.1 SplashActivity, schermata di caricamento

L'immagine in figura è stata creata utilizzando *Inkscape,* un programma *free software*

per il disegno vettoriale, basato sul formato *Scalable Vector Graphics* (*SVG*). La *SplashActivity,* sarà visualizzata per un totale di tre o quattro secondi massimo, dopodiché verrà automaticamente chiusa e il comando passerà alla classe *MainMenùActivity.* Questa classe è già stata indirettamente introdotta nel primo paragrafo del *capitolo 2*, quando ho esposto l'analisi dei requisiti. Questa classe produrrà una schermata diversa da quella proposta in *figura 3.1,* non solo per quanto riguarda la parte grafica, ma soprattutto perché è interattiva per l'utente. Infatti attraverso l'apposito file di Layout ".xml", verranno creati quattro pulsanti che avranno il compito di direzionare l'utente nelle quattro sezioni principali dell'applicazione, cioè "Sezione Accedi", "Sezione Partita Singola", "Sezione Partita Online" e "Sezione Statistiche". Questa schermata potrà essere modificata nel tempo, nel momento in cui l'utente effettua la registrazione (o il Login) presso il server dedicato. Infatti il pulsante (in gergo *Button*) che si occupa di indirizzare l'utente nella "Sezione Accedi", dovrà essere nascosto e sostituito da una stringa di testo (in gergo *TextView)* di benvenuto. In *figura 3.2* sono rappresentate le due versioni della schermata *MainMenùActivity.* Quella disposta sul lato sinistro mostra la schermata nel caso in cui l'utente non sia *loggato*. Quella a destra mostra invece la schermata con l'utente *luciod* (nome prettamente dimostrativo) che ha eseguito l'accesso/registrazione.

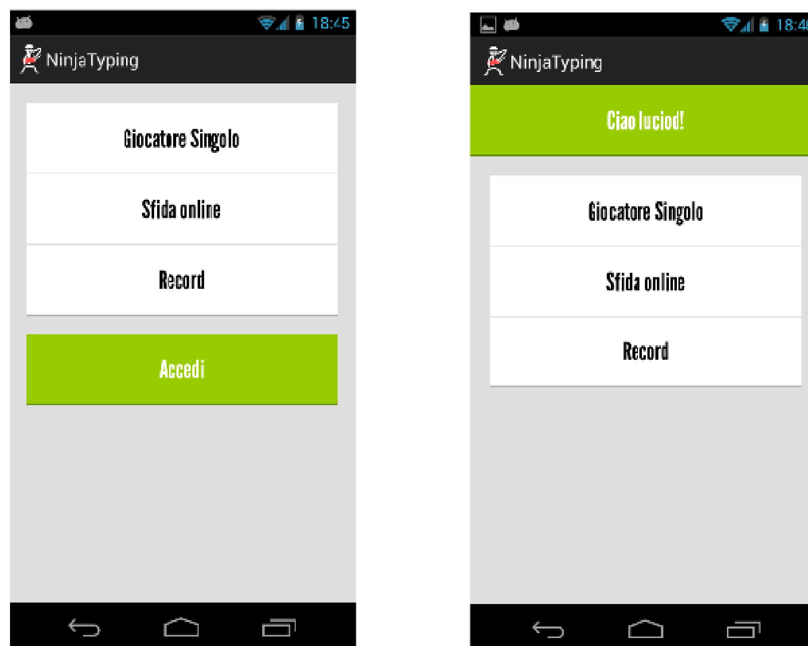


Figura 3.2 Menù iniziale. Utente loggato (sinistra), utente non loggato (destra)

Con questi *Screenshot*, chiudo il primo paragrafo dedicato allo sviluppo del menù principale. Ora andrò ad affrontare singolarmente le varie sezioni, intervallandole da importanti paragrafi, utili per la spiegazione di importanti algoritmi o funzioni, utilizzate in determinate sezioni.

3.2 Implementazione dell'account utente

Attraverso la schermata gestita dal *MainMenùActivity,* l'utente, sotto la pressione dell'apposito pulsante, potrà entrare nella "sezione Accedi". Questa sezione per concezione è molto simile a moltissime *form* di *Login* che è possibile trovare in siti Web, Forum e altre app. L'utente avrà la possibilità di effettuare una completa registrazione, oppure se già registrato, potrà semplicemente accedere, inserendo username e password utilizzate al momento della registrazione. In questo modo l'utente potrà quindi giocare con lo stesso username, con più dispositivi. In *figura 3.3* ci sono due Screenshot, che descrivono la schermata in cui l'utente può effettuare l'accesso (quella a sinistra) e la schermata in cui l'utente può effettuare la registrazione (a destra).

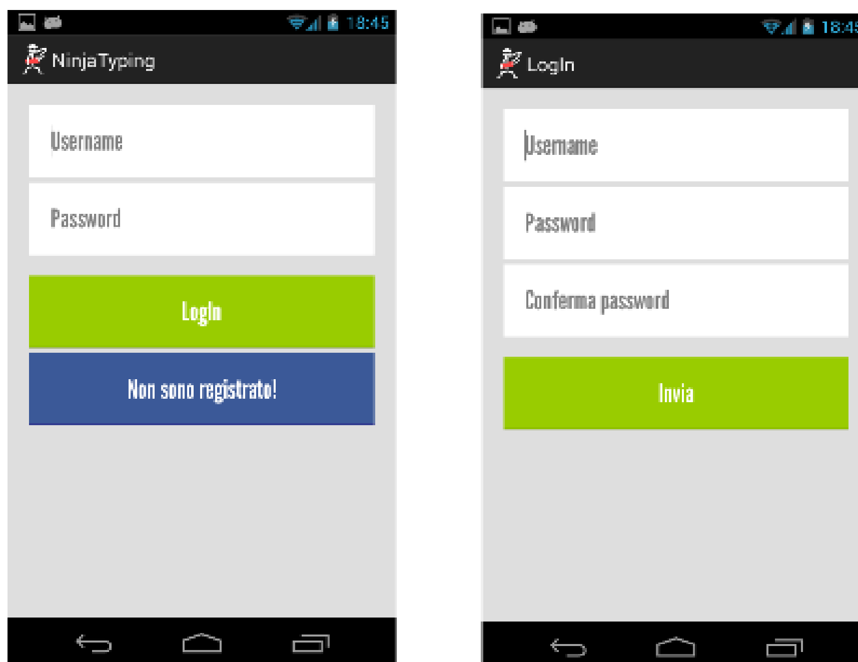


Figura 3.3 Schermate di registrazione (destra) e accesso (sinistra)

Nel caso di una registrazione (schermata a destra in f*igura 3.3*), l'utente dovrà inserire uno *username* di lunghezza compresa tra 3 e 15 caratteri. La *password* dovrà essere invece lunga tra 4 e 10 caratteri. In caso l'utente non rispetti questi parametri, verrà visualizzato un messaggio di errore e i dati non verranno inviati al server. In caso invece, i dati d'accesso fossero conformi ai vincoli di cui sopra, verrà inviata una richiesta al server utilizzando classi e interfacce presenti nel package "org.apache.http" fornito nelle API LEVEL 1. Il server restituirà un valore booleano positivo (contenuto in un oggetto JSON), nel caso in cui lo username indicato dall'utente, fosse disponibile e quindi correttamente registrato nel database. Per quanto riguarda invece il semplice accesso (schermata a sinistra in *figura 3.3*), il procedimento lato Client sarà assolutamente identico.
Quindi una volta che il Client avrà ricevuto risposta affermativa da parte del server, lo Username e la password indicati dall'utente, dovranno essere memorizzati sul dispositivo, in modo che al successivo accesso non venga richiesto nuovamente di effettuare la registrazione/login. Salverò entrambi i dati come oggetti di tipo *String*, attraverso la classe *android.content.SharedPreferences* che Android mette a disposizione degli sviluppatori, per salvare settaggi applicativi in un file, per poi condividerli nella stessa applicazione o tra tutte le applicazioni. Nel mio caso ovviamente renderò il file contenente Username e Password, visibile solo agli altri componenti dell'app NinjaTyping. Per quanto riguarda la registrazione lato server, il procedimento è molto banale. Utilizzerò una tabella contenente i campi *username* (*varchar* da 15 caratteri massimo), *password* (*varchar* da 10 caratteri massimo) e *_id* (chiave primaria con valore auto incrementante). La *collation* (cioè criterio di codifica, ordinamento e confronto caratteri) utilizzata per i campi username e password è utf8_unicode_ci, che tra le altre cose è una codifica *case-insensitive,* ciò vuol dire che due username del tipo "GioVanni" e "giovanni", saranno considerati univocamente. Le richieste di registrazione/login che arriveranno al database, verranno gestite attraverso appositi script scritti in linguaggi *php* che avranno il semplice compito di controllare la validità della richiesta effettuata dall'utente. Se si tratta di una richiesta di accesso dovrò semplicemente verificare che username e password inviati dall'utente, siano già presenti nel database, restituendo un oggetto JSON booleano settato a *true*. In caso di richiesta di registrazione dovrò semplicemente verificare che lo username richiesto dall'utente non

sia già “occupato” e quindi presente nella tabella.
Questa sezione che ho denominato “sezione accedi”, sarà visualizzata oltre che in maniera esplicita premendo l'apposito *Button* in MainMenùActivity, oppure in maniera automatica, nel caso in cui l'utente cerchi di giocare una partita online senza essersi registrato/loggato. Nel prossimo paragrafo tratterò la sezione dedicata alla partita in *single game*, indicandone le due possibili modalità di gioco.

3.3 Scelta della modalità in Partita Singola

Come visto dagli Screenshot del secondo paragrafo, l'utente potrà attraverso l'apposito pulsante presente in MainMenùActivity, accedere alla sezione di gioco partita singola, che viene gestita dalla classe *GameTypeChooserActivity.* Questa classe permetterà all'utente (attraverso apposito Layout grafico con due pulsanti), di scegliere quale modalità di gioco singolo affrontare. Le modalità sono due e le ho denominate *Normal Mode* e *Reversed Mode.*
Il criterio di gioco è assolutamente identico per entrambe modalità, così come la composizione della frase ed il calcolo del punteggio. Quello che cambia è esclusivamente il livello di difficoltà dettato dalla composizione della tastiera fisica. Attraverso l'apposito file xml, ho creato due differenti tastiere, che generano il risultato visibile in *figura 3.4.* Entrambe le tastiere presentano solo caratteri alfabetici, ma giocare in *Reversed Mode* è più complicato in quanto la tastiera è invertita.

Figura 3.4 Tastiera normal e tastiera reversed

Quindi l'utente sotto pressione dell'apposito pulsante, deciderà quale tipo di partita lanciare. Se sceglierà una partita in modalità *Normal*, il sistema lancerà la schermata gestita dalla classe *GameActivity,* passandogli un valore contenuto nella costante *GAME_MODE_NORMAL,* definita nella classe *Utils.* Se l'utente sceglierà invece una partita di tipo *Reversed*, allora verrà lanciata la classe *GameActivity,* che in ingresso riceverà il valore definito nella costante *GAME_MODE_REVERSED.* Entrambe le modalità verranno quindi gestite dalla stessa classe, che a seconda del parametro in ingresso, caricherà l'apposita risorsa xml (nello specifico *keyboard.xml* oppure *keyboard_reversed.xml),* per visualizzare l'adatta tastiera. Come detto lo svolgimento della partita sarà poi identico in entrambi i casi, quello che cambierà sarà il salvataggio dei dati della partita nel database locale e del server come spiegherò nel prosieguo del capitolo. Nei prossimi paragrafi procederò spiegando l'implementazione che permette all'utente di giocare effettivamente una partita, quindi la gestione del dizionario, lo sviluppo grafico della schermata partita attraverso una View personalizzata e l'algoritmo di calcolo di punteggio.

3.4 Implementazione grafica della partita

Il Layout grafico in Android viene definito attraverso una serie di file *xml*. Dalle API LEVEL 1 sono state messe a disposizione degli sviluppatori delle componenti grafiche di default (ad esempio i Button visti in *MainMenuActivity*) personalizzabili, utili a costruire l'interfaccia utente. Tutti questi componenti (in continua espansione ad ogni nuova API rilasciata) sono stati creati attraverso *sottoclassi* della classe *View,* definita nel package *android.view.View.* Nonostante però Android fornisca un grande quantitativo di elementi grafici "già fatti" e pronti all'uso, non sempre questi elementi sono in grado di coprire tutte le necessità degli sviluppatori. In tal caso è possibile sviluppare un proprio componente grafico, creando una classe che erediti da *android.view.View.* E' esattamente quello che ho fatto per sviluppare la schermata di gioco in NinjaTyping. Mi serviva infatti un componente (*View*), totalmente "nuovo" del quale potessi gestire autonomamente dimensioni e comportamento. Ho così creato la classe *TypingView,* che come detto estende la classe *View,* per disegnare su schermo la

frase generata casualmente dal sistema, che l'utente dovrà digitare attraverso la tastiera fisica. Per lo sviluppo ho proceduto rispettando le direttive esposte in fase di progettazione. Voglio che la frase contenuta nella *TypingView*, occupi sempre metà dello schermo (con *Orientation Portrait*). L'altra metà sarà occupata dalla tastiera, attraverso un oggetto Keyboard che spiegherò successivamente. La classe che si occuperà di disegnare su schermo ogni carattere è la classe *WordCell*. Prima di proseguire mi avvalgo di un aiuto grafico, per capire meglio i primi concetti appena descritti.

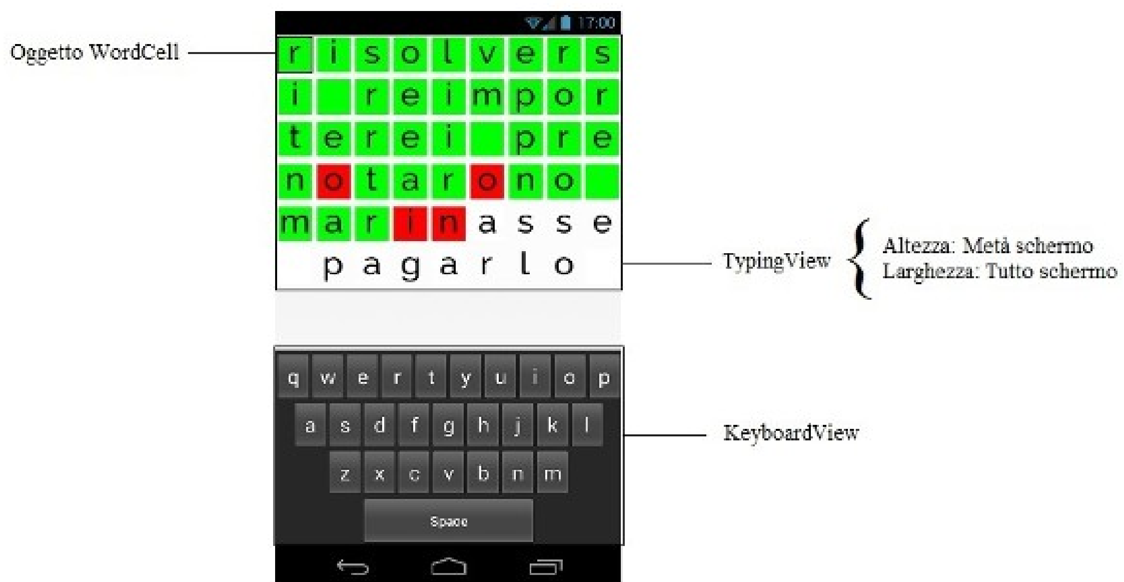


Figura 3.5 Dimensionamento grafico della partita

Come intuibile dalla figura soprastante l'oggetto TypingView conterrà al suo interno tanti oggetti WordCell, quanti ne sono necessari per "disegnare" su schermo la frase. In figura ho specificato che la mia View personalizzata, dovrà essere larga quanto tutto lo schermo e alta quanto la metà dello schermo. La classe TypingView avrà il compito, attraverso l'override di un apposito metodo, di misurare le dimensioni reali a propria disposizione (in base alla grandezza dello schermo) e di conseguenza calcolare quanto dovrà essere grande ogni oggetto WordCell e indicare ad ognuno di questi oggetti dove andare a disegnarsi su schermo e quale carattere della frase disegnare. Questo vuol dire che ogni oggetto WordCell, avrà il compito di disegnare se stesso attraverso l'apposito metodo pubblico `draw(Canvas canvas)`, grazie alle coordinate passate in ingresso al costruttore, dalla classe TypingView al momento della inizializzazione. Per disegnare

un oggetto WordCell utilizzerò gli strumenti per la grafica, che Android mette a disposizione dalle API LEVEL 1, definite nel package android.graphics. Nello specifico utilizzerò un oggetto Canvas, definito in android.graphics.Canvas, che è da intendere come una sorta di tela sulla quale disegnare. Il disegno verrà concretamente generato grazie ad un oggetto Paint, definito in android.graphics.Paint, che è da intendere come una matita grazie alla quale disegnare sulla tela (il Canvas precedente). Ci sono diversi metodi per disegnare attraverso un Canvas. Inizialmente avevo implementato l'interfaccia grafica disegnando ogni oggetto WordCell, invocando sull'oggetto Canvas il metodo

```
public void drawText (String text, float x, float y, Paint
paint).
```

Tale metodo prende in ingresso le coordinate da cui partire per disegnare il testo, un oggetto Paint e una stringa, che nel mio caso si traduceva in un semplice carattere, in quanto ricordo che ogni oggetto WordCell, conterrà un carattere della frase che l'utente dovrà digitare in partita. Il risultato non era però apprezzabile, perché a seconda della risoluzione dello schermo, il carattere veniva disegnato non propriamente al centro della cella, o addirittura fuori dalla cella. Quindi ho optato per un metodo più macchinoso ma efficace, invocato sempre attraverso l'oggetto Canvas, così definito

```
public void drawBitmap (Bitmap bitmap, Rect src, Rect dst, Paint
paint).
```

Attraverso questo metodo non scrivo un carattere, ma lo disegno in maniera vera e propria. Per farlo devo passare al metodo un oggetto Bitmap, che verrà inizializzato recuperando dalla cartella res/drawable, l'immagine appositamente creata (utilizzando Inkscape) per rappresentare il carattere. Dove effettivamente questa immagine dovrà essere disposta su schermo, verrà indicato attraverso il parametro *Rect dst,* che sarà un oggetto rettangolo che creerò inizializzandolo attraverso le coordinate passate al costruttore e all'interno del quale verrà disegnata l'immagine Bitmap. In questo modo, avrò la certezza che l'immagine sarà disegnata in maniera corretta come visibile nella *figura 3.5* riportata in precedenza. Sempre osservando la figura, si nota come ogni

oggetto WordCell, possa avere però tre colori differenti. Questi tre colori sono rappresentati da tre oggetti di tipo Bitmap (presenti in ogni oggetto WordCell), che rispecchiano lo stato di ogni cella. Infatti le gli oggetti WordCell creati, potranno assumere nel corso della loro “vita” tre differenti stati (definite come costanti nella classe stessa):

- **STATE_INACTIVE**: sarà lo stato iniziale di ogni cella (carattere). Il Bitmap corrispondente sarà di colore bianco e come detto, dovrà essere recuperato nella cartella res/drawable.
- **STATE_FALSE**: la cella assumerà questo stato in caso l'utente abbia digitato il carattere corrispondente in maniera errata. In questo caso il Bitmap sarà di colore rosso.
- **STATE_RIGHT**: la cella assumerà questo stato in caso l'utente abbia digitato il carattere in maniera corretta. Il Bitmap sarà verde.

Come un oggetto passerà dal diventare bianco, al diventare verde o rosso, lo spiegherò nel paragrafo dedicato allo sviluppo dell'algoritmo di gioco.
Ho descritto quindi i concetti fondamentali della classe WordCell, ma ora farò un passo indietro ed entrerò nel dettaglio della classe TypingView, per descrivere come viene assegnata la misura che dovrà avere ogni WordCell. Come detto la classe in questione estenderà la classe View, in modo da poterne utilizzare i principali metodi. I metodi in questione utili per definire la grandezza della TypingView e degli oggetti contenuti e implementarne il disegno su schermo, saranno due:

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec);

protected void onDraw(Canvas canvas);
```

Attraverso il metodo onDraw(), è possibile disegnare la schermata e le celle che la compongono. Come visto in precedenza, ogni oggetto WordCell, disegna in un certo senso sé stesso attraverso il metodo pubblico draw(). Tale metodo viene invocato sull'oggetto, attraverso metodo onDraw() della TypingView, passandogli come parametro l'oggetto Canvas sul quale poi disegnare le immagini rappresentanti i caratteri. Il metodo onDraw(), sarà invocato ogni qual volta ci sarà bisogno di

ridisegnare la TypingView e le sue celle. Questo accade nel caso in cui l'utente stia giocando una partita e di conseguenza alla pressione di un pulsante della tastiera, un determinato carattere disegnato su schermo, dovrà cambiare stato e quindi colore.

Per quanto riguarda invece il dimensionamento della TypingView e degli oggetti WordCell, utilizzerò il metodo onMeasure() precedentemente riportato. Attraverso i due parametri in ingresso, riuscirò ad ottenere le misure esatte per ogni dispositivo in larghezza e altezza, che la TypingView ha a disposizione per disegnare la frase. Gli oggetti WordCell avranno una dimensione quadrata e questa dimensione sarà il minimo tra due variabili così calcolate:

```
wordCellWidth = getWidth() / CELL_NUMBER_COLUMN;
wordCellHeight = getHeight() / CELL_NUMBER_LINES;
```

Il campo *wordCellWidth* è dato dalla larghezza dello schermo diviso il numero di colonne, cioè il numero di celle da disporre in una riga. Il campo wordCellHeight invece è dato dall'altezza dello schermo a disposizione della TypingView, diviso il numero di righe, cioè il numero di celle da disporre per colonna. Tra questi due dati verrà scelto il minimo che sarà la dimensione effettiva delle celle in altezza e larghezza. La dimensione della cella sarà contenuta nella variabile *cellDimension.* Una volta stabilita la grandezza delle celle, procederò alla creazione ed inizializzazione di tutti gli oggetti WordCell, passando al costruttore le coordinate in cui voglio siano disegnate le celle, contenti i caratteri delle dimensioni preposte. Il costruttore della classe WordCell, sarà il seguente

```
public WordCell(float leftX, float topY, float rightX, float
bottomY, Context context, char currentChar);
```

Oltre al Context di un' Activity (utile per recuperare risorse di vario genere dalla cartella *res*), sarà presente una variabile di tipo *char,* grazie alla quale ogni oggetto WordCell, capirà quale Bitmap dovrà recuperare dalla cartella *res/drawable.* Inoltre come detto, ci saranno le coordinate che rappresenteranno rispettivamente il lato sinistro, il lato superiore, il lato destro e il lato inferiore in cui disegnare il rettangolo. Verranno quindi inizializzati attraverso un ciclo *for,* tutte le celle necessarie. L'algoritmo che calcola le

coordinate è molto macchinoso e riportare il codice sarebbe piuttosto incomprensibile. L'algoritmo in questione ha comunque il compito di assegnare ogni coordinata ai vari rettangoli, che dovranno avere una grandezza pari a *cellDimension* (riportato precedentemente), al quale va sottratto un piccolo margine per evitare di avere tutte le celle "incollate" le une con le altre.
Il disegno della frase su schermo non è però l'unico elemento che deve essere rappresentato in proporzione a seconda dei vari dispositivi. Infatti come noto, la schermata di gioco, avrà una tastiera che sarà contenuta nella seconda metà della schermata. Come rendere proporzionato ogni tasto della tastiera, ad ogni schermo di un qualsiasi dispositivo, è molto semplice. Infatti Android mette a disposizione dell'utente un oggetto KeyboardView, che è concettualmente abbastanza simile alla TypingView da me costruita. La KeyboardView contiene infatti al suo interno un oggetto di tipo Keyboard. Una Keyboard non è altro che un componente, personalizzabile in xml, che grazie a una particolare struttura, permette di definire e dichiarare i tasti della Keyboard suddividendoli per righe. E' possibile inoltre dare ad ogni tasto (chiamato *key*) di ogni riga (chiamata *row*), una particolare dimensione (in altezza e larghezza) espressa in percentuale rispetto alla totalità dello schermo (in altezza e larghezza). Quindi per esempio in NinjaTyping in cui ho una tastiera con quattro righe, ogni elemento (key) della riga, non dovrà avere un valore in altezza superiore a 12.5%. Questo perché come già detto, la tastiera quindi la KeyboardView, dovrà occupare al massimo la metà dello schermo, in quanto altrimenti andrei a sovrascrivere l'altra metà occupata dalla TypingView. Nello specifico ho sviluppato la tastiera assegnando ad ogni tasto, un'altezza inferiore a 12.5%, in modo da far occupare alla KeyboardView meno del 50% dello spazio disponibile. Il motivo di questa scelta è abbastanza curioso e verrà spiegato nel *capitolo 4,* dedicato agli sviluppi futuri dell'applicazione.

Dopo aver visto quindi il meccanismo che permette al sistema di disegnare la frase di gioco, andrò ad affrontare nel prossimo paragrafo, l'implementazione degli algoritmi che permettono lo svolgimento di una partita.

3.5 Gestione del dizionario

La classe che si occupa di implementare la logica di gioco è la GameActivity. Quando tale classe viene lanciata, una delle prime cose che fa, attraverso l'override del metodo *onCreate,* è inizializzare una variabile di tipo String attraverso il metodo

```
public static String getPhrase();
```

contenuto nella classe Dictionary. Questo metodo ha il compito di restituire una frase con parole prese in maniera random dal dizionario (caricato nella SplashActivity). La frase restituita, sarà lunga esattamente quanto il numero di celle che comporrà la TypingView (vista nel paragrafo precedente). Il dizionario verrà caricato attraverso il metodo statico della classe Dictionary

```
public static void loadDictionary(Context context, String
fileName);
```

che ha il compito di inizializzare la variabile statica

```
public static ArrayList<String> buffer= new ArrayList<String>();
```

Il metodo *loadDictionary()* prende in ingresso oltre al Context dell'Activity chiamante, anche il nome del dizionario da caricare (ad esempio *italian.txt* oppure *english.txt*). L'oggetto buffer invece conterrà in ogni elemento dell'Array una parola diversa del dizionario. Quindi attraverso il metodo *getPhrase(),* estrarrò in maniera casuale degli elementi dall'oggetto *buffer,* componendo la frase da digitare, prestando però attenzione a generare una frase di lunghezza esatta. La lunghezza che dovrà avere la frase random generata, è indicata dalla costante *CELL_NUMBER_TOTAL* definita nella classe Utils.
Prima di concludere, devo fare una precisazione riguardo le parole che compongono il dizionario. Il file dizionario caricato nella cartella *assets*, è limitato. Per limitato intendo dire che ho tolto tutte le parole comprensive di apostrofi e accenti, in quanto le tastiere che ho creato (*figura 3.4* del precedente paragrafo) non prevedono accenti o apostrofi.
Quindi ogni volta che l'utente comincia una nuova partita (che sia in modalità singola,

oppure modalità online), verrà utilizzato il procedimento appena descritto per ottenere una frase.

3.6 Sviluppo dell'algoritmo di gioco

Dopo aver descritto tutti i componenti fondamentali di una partita, procedo indicando lo sviluppo vero e proprio del gioco. La classe che si occupa della logica di gioco è GameActivity. GameActivity attraverso il metodo onCreate(), crea il Layout grafico (*paragrafo 3.4*), tastiera compresa, ed ottiene una frase da digitare (paragrafo precedente) in modo da far cominciare la partita. Ad ogni pressione di un tasto della tastiera devo intercettare il tocco, capire quale carattere l'utente ha digitato, confrontare il carattere digitato dall'utente, con quello corrente da digitare nella frase disegnata su schermo e colorare tale carattere di rosso, in caso di errore, oppure di verde in caso contrario. Tutto questo è gestibile implementando l'interfaccia *KeyboardView.OnKeyboardActionListener* introdotta con le API LEVEL 3. Si tratta di un cosiddetto *listener*, per gli eventi della tastiera virtuale. Tale interfaccia offre una vasta quantità di metodi per gestire i più disparati tipi di tocco della tastiera. Quello che serve nel mio sviluppo, è il metodo

```
public void onPress(int keyCodes)
```

dove implementerò (facendone l'override) tutta la logica di gioco. Questo metodo prende in ingresso un intero che rappresenta un codice numerico univoco, atto ad identificare il tasto premuto dall'utente. Io dovrò quindi confrontare tale carattere, con il carattere corrispondente disegnato su schermo. Per tenere conto del carattere corrente mi avvarrò di un semplice campo di tipo intero chiamato *touchCount*, che verrà incrementato ogni volta che verrà invocato il metodo onPress(). Questo contatore servirà anche per capire, quando l'utente ha digitato tutti i caratteri e di conseguenza quando sarà il momento di terminare la partita calcolando il punteggio. Infatti quando touchCount avrà un valore numerico pari alla lunghezza della frase da digitare, la partita terminerà.

E' importante però fare un passo indietro e tornare a parlare del parametro in ingresso

del metodo onPress(), cioè del keyCodes. Per capire se l'utente ha digitato il carattere corretto, dovrò operare un confronto tra il carattere digitato e il carattere da digitare. Il problema è che il keyCodes di un carattere, non corrisponde al codice ASCII del carattere stesso. Quindi per ottenere un confronto corretto, ho implementato un metodo statico definito nella classe Dictionary, che prenderà in ingresso il keyCodes del carattere digitato e ne restituirà il codice ASCII. Di seguito il codice del metodo

```
public static char keyCodeToAscii(int keyCode) {

                if (keyCode >= KeyEvent.KEYCODE_A && keyCode <=

      KeyEvent.KEYCODE_Z) {
                      keyCode = keyCode - KeyEvent.KEYCODE_A + 'a';

                } else if (keyCode == KeyEvent.KEYCODE_SPACE){
                      keyCode = ' ';
                }

                return (char) keyCode;

      }
```

Per spiegare il codice di cui sopra mi avvarrò di un piccolo esempio. Ipotizzando che l'utente prema il carattere “a”, il sistema onPress() restituirà il keyCodes numerico “29”. Il valore intero della lettera “a” in ASCII è 97. Se l'utente premesse il carattere “b” il keyCodes restituito sarebbe “30”, mentre in ASCII dovrebbe essere “98”. Più in generale quindi, i valori keyCodes dalla “a” alla “z” sono compresi nel range 29(a), 54(z). I valori ASCII dei caratteri dalla “a” alla “z” sono invece compresi tra 97(a) e 122(z). Quindi per convertire un numero da keyCodes a ASCII, basterà sottrarre al keyCodes in questione, se stesso meno il keyCodes del carattere “a” e quindi sommare il valore ASCII sempre del carattere “a”. Discorso diverso per i caratteri speciali (punteggiatura, spazi, asterischi) che devono essere forzati non algoritmicamente ma "brutalmente". Il mio unico carattere speciale è lo *spazio*, e sarà convertito grazie alla condizione *else if,* del codice sopra riportato.

Il prosieguo del metodo onPress(), diventa a questo punto molto lineare. Dopo aver ottenuto il codice ASCII del keyCodes digitato, provvederò a confrontarlo con il carattere corrente che l'utente avrebbe dovuto digitare. A questo punto si verificano due sole possibilità, cioè l'utente ha sbagliato a digitare, oppure l'utente ha digitato

correttamente. In entrambi i casi verrà chiamato il metodo

```
public void onUpdate(boolean err)
```

della classe TypingView, che ha il compito di ridisegnare l'intera View che contiene la frase, invocandone il metodo onDraw(), aggiornandone lo stato di ogni cella. Come detto, ogni carattere viene disegnato (con un immagine Bitmap) su schermo, grazie agli oggetti di tipo WordCell. Questi oggetti possono avere tre stati, che corrispondono a tre colori. Quando al metodo onUpdate(), verrà passato un booleano con valore settato a *true,* vorrà dire che il carattere corrente dovrà cambiare stato e passare da *STATE_INACTIVE* (lo stato iniziale che tutti hanno prima di essere in un certo senso"digitati"), a *STATE_ERROR* (il Bitmap dell'immagine corrispondente cambia colore e diventa rosso). Viceversa se il parametro passato sarà settato a *false,* allora lo stato dell'oggetto WordCell corrente, dovrà passare a *STATE_RIGHT* (il Bitmap diventerà verde).
Questo procedimento, verrà ripetuto ogni volta che l'utente premerà un tasto della tastiera, quindi presumibilmente avremo un numero di questi cicli, pari alla lunghezza della frase. Come detto l'algoritmo e la partita si concludono in prossimità della digitazione dell'ultimo carattere. A quel punto il controllo passa al metodo (della classe GameActivity corrente)

```
private void calculateScore(int lenght)
```

che ha il compito di calcolare il punteggio finale e inizializzare l'oggetto *GameRecord,* che dovrà contenere tutti i dati della partita e dovrà essere passato all'Activity (quindi schermata) successiva cioè *GameRecapActivity.* Di seguito il codice con l'algoritmo che calcola il punteggio:

```
private void calculateScore(int lenght){
                float result =0;
                timePlayed=(float) (timePlayed/1000);
                result= (float)(lenght-errors)/ timePlayed;
                result= result*1000;
                score =(int) (result-(errors*100));
                [...]
    } //end method
```

Prima di procedere un paio di precisazioni. La variabile *errors* nel codice di cui sopra, è un semplice contatore di tipo intero che viene incrementato ogni qualvolta l'utente digita un carattere sbagliato. La variabile *tymePlayed* è una variabile di tipo *Float,* che è ottenuta calcolando il tempo trascorso dal primo tocco dell'utente (cioè quando comincia la partita), all'ultimo tocco (cioè quando termina la partita). Per il resto l'algoritmo tiene conto della velocità dell'utente nel digitare una frase e nel numero di errori che l'utente commette. Ad esempio la prima inizializzazione della variabile *result* (riportata nel precedente codice) è calcolata in modo da restituire un numero più alto, se l'utente commette meno errori e nel minor tempo possibile. Infatti divido la lunghezza della frase digitata per il tempo trascorso. Se l'utente commette qualche errore, sottrarrò il numero di errori alla lunghezza della frase e la dividerò per il tempo trascorso. In questa maniera ho la certezza di dare peso sia agli errori che al tempo. Questo è però un risultato parziale, in quanto il numero ottenuto sarà moltiplicato per 1000. In questo modo, ho un range di punteggio più alto, ed è più difficile che gli utente facciano gli stessi punteggi. Per concludere a quest'ultimo valore ottenuto, sottrarrò un'ulteriore penalità, pari a 100 punti per ogni errore commesso. Ho fatto questa scelta per dare ancora maggior peso agli errori che un utente commette.

Dopo aver calcolato il punteggio, procedo quindi a creare e inizializzare l'oggetto *GameRecord,* che conterrà il punteggio (*score),* il numero di errori commessi, il tempo trascorso per giocare la partita, il tipo di partita giocata (inteso modalità *Reversed* o *Normal*) e la frase digitata. La classe GameRecord, implementa l'interfaccia *Parcelable* , grazie alla quale sarà possibile passare l'intero oggetto appena creato ad una nuova Activity (*GameRecapActivity*), che poi provvederà attraverso un oggetto dello stesso tipo, a recuperarne i dati. Il paragrafo corrente è concluso, procederò quindi trattando l'implementazione dell'Activity GameRecapActivity.

3.7 Sezione Riepilogo di gioco

Quando la partita termina, il controllo passa alla schermata Riepilogo di Gioco, implementata attraverso la classe *GameRecapActivity.* Questa sarà un'Activity in cui

verranno mostrate in delle apposite TextView, i vari dati che l'utente ha totalizzato in partita. In più saranno presenti due pulsanti (Button) che avranno il compito di permettere all'utente di tornare al menù principale, oppure condividere il proprio punteggio attraverso il proprio profilo utente Facebook. Se l'utente avrà totalizzato un primato personale, verrà visualizzato a schermo un apposito messaggio di congratulazioni. Inoltre se l'utente ha effettuato precedentemente una registrazione/accesso, allora i dati partita verranno inviati al server dedicato, in modo da dare all'utente la possibilità di entrare nei migliori 100 record della modalità effettuata. In *figura 3.6* uno Screenshot del risultato visivo dell'Activity.

Figura 3.6 Schermata di riepilogo della partita giocata

I dati riguardanti il punteggio, gli errori e i secondi, sono stati passati dall'Activity chiamante attraverso l'oggetto GameRecord, che implementa l'interfaccia parcelable. La statistica "tocchi per secondo" invece, non è stata passata dall'Activity chiamante e non sarà memorizzata da nessuna parte, né sul server né nel database locale. Questo perché si tratta di un dato ricavabile, dividendo il numero totale di parole che compongono la

frase, per il tempo impiegato. Ho voluto semplicemente evitare una ridondanza. Proseguendo nella spiegazione dell'implementazione della GameRecapActivity, è importante sottolineare che il punteggio effettuato contenuto nell'oggetto *GameRecord,* verrà memorizzato nel database locale chiamando un'apposita funzione, che restituirà un valore booleano positivo, in caso il punteggio effettuato dall'utente sia il migliore mai effettuato dallo stesso (un nuovo record quindi).
La classe in cui ho implementato tutte le funzioni che si occupano di gestire i dati del database locale, si chiama *MyDatabase.* Il metodo di tale classe, che si occupa di inserire in tabella la partita e restituire un booleano in caso di nuovo Record è:

```
public boolean createRecord(int score, int errors, String time,
String phrase, int type)
```

e come prevedibile prende in ingresso tutti quei dati, che sono stati passati attraverso l'oggetto di tipo GameRecord, dalla GameActivity alla GameRecapActivity. Il metodo createRecord(), come detto avrà una doppia funzione. Inserirà i dati nella tabella principale contenente tutte le partite effettuate dall'utente. Inoltre provvederà a chiamare un metodo privato della classe MyDatabase, che dopo un apposito controllo, procederà ad inserire il punteggio partita, nella tabella che mantiene i 10 migliori punteggi dell'utente. Nel caso il punteggio inviato sia addirittura il migliore, verrà restituito un valore booleano *true,* in modo da dare l'opportunità all'Activity chiamante di notificarlo graficamente all'utente. Il metodo che si occupa di gestire i 10 migliori punteggi dell'utente è:

```
private boolean insertNewScore(int score, int errors, String
time, String phrase, int type)
```

Il corpo di tale metodo prevede un algoritmo molto banale, che mi limiterò a illustrare brevemente, senza riportare il codice. Il metodo *insertNewScore(),* effettua innanzitutto una query sulla tabella TOP_TEN_SCORES, che restituisce tutti e 10 i punteggi del tipo indicato dal parametro in ingresso *type.* Ricordo infatti che questa tabella contiene al massimo 10 punteggi per ogni modalità di gioco implementata (*Reversed* e *Normal).* Se la query in questione restituisce meno di 10 punteggi, allora i dati passati come

parametri al metodo, verranno inseriti direttamente nel database. Se invece la tabella contiene già 10 punteggi di un determinato tipo, allora provvederò semplicemente nella ricerca del punteggio più alto e di quello più basso. Se il punteggio appena effettuato dall'utente, è maggiore del punteggio più basso presente in tabella, allora procederò semplicemente ad aggiungere il nuovo punteggio, togliendo il più basso. In caso poi il punteggio inserito fosse anche il più alto presente in tabella, restituirò il valore booleano settato a *true,* il che vorrebbe dire che l'utente ha effettuato un nuovo *Personal Record.*

Tornando ora all'Activity GameRecapActivity, la successione di eventi sarà scaturita dal comportamento dell'utente. Come visibile nella *figura 3.6* precedentemente riportata, il giocatore potrà premere il pulsante “Condividi su Facebook” e “Torna al Menù”. Supponendo che venga premuto il pulsante che permette di tornare al menù principale, verrà attivato il meccanismo che cerca di condividere il punteggio precedentemente inserito nel database locale, anche sul server (ma solo nella tabella dei migliori record). Per effettuare una richiesta al server attraverso l'apposito procedimento (molto simile a quanto già visto nel *paragrafo 3.2*), dovranno verificarsi due importanti condizioni, cioè l'utente dovrà essere *registrato/loggato* e ci dovrà essere connettività a internet. Per verificare queste due condizioni che saranno utili in varie porzioni del codice dell'applicazione, ho implementato due metodi statici. Il metodo utile per capire se l'utente è registrato o meno, è implementato nella classe *MyPreferences,* ed è il seguente:

```
public static String getUsername(Context context).
```

Semplicemente attraverso questo metodo controllo se esiste un file (creato come visto nel *paragrafo 3.2*), con uno username. Se il metodo restituisce il valore *null,* allora l'utente non si è registrato quindi non posso mandare il suo record nel server. In caso l'utente fosse registrato, prima di procedere con l'invio dei dati al server, devo verificare la connessione a internet e lo farò attraverso il metodo statico della classe *Utils*, seguente:

```
public static boolean verifyConnection(Context context).
```

Si tratta di un metodo che sfrutta l'oggetto *NetworkInfo* (fornito da Android a partire dalle API LEVEL 1) per controllare se l'utente è connesso a internet. Se il dispositivo mobile è connesso a internet, restituisce *true.* Quindi se entrambi i metodi daranno esito positivo, si può procedere ad inviare i dati della partita al server. Come detto il procedimento di invio dati al server è molto simile a quanto già visto nel *paragrafo 3.2*, quel che cambia è l'URL di richiesta al server e i parametri che vengono inviati al server stesso. Lato server ci sarà uno script *php,* che prenderà in ingresso i parametri *score, errors ,phrase, time, type e username* che altro non sono che i dati appena totalizzati dall'utente in partita. La tabella che si occupa di gestire questi record lato server, è strutturata in maniera quasi identica a quella presente sul database locale. La tabella presenta infatti i campi *id, score, errors, time, phrase, username, game_type*. Lo script *php,* invece è molto simile al metodo *insertNewScore(),* implementato nella classe *MyDatabase,* che ho precedentemente esposto. L'unica differenza, sta nel numero totale di record coinvolti. Infatti nella tabella locale i record coinvolti erano al massimo 10 per ogni tipo di partita giocata, mentre sul server, verranno memorizzati 100 record per ogni tipo. Il server restituirà poi un oggetto JSON con valore *true* in caso l'utente sia entrato nei 100 migliori punteggi mondiali, altrimenti *false.* Quindi dopo la condivisione del punteggio sul server il comando tornerà al MainMenùActivity.

Nel caso invece l'utente decidesse di condividere il proprio punteggio partita premendo l'apposito *Button,* il comando passerà all'Activity *ShareOnSocial,* che ha il compito di interfacciarsi con il server di Facebook attraverso le API che il social Network mette a disposizione degli sviluppatori. Attraverso questa classe, verrà presentata un'interfaccia di login (fornita dalle API di Facebook), che permette all'utente di effettuare il login con il proprio account. Infatti è assolutamente impossibile condividere il record se prima non si effettua il login. Una volta completata la *form* di Login e dopo aver accettato i permessi vari che NinjaTyping richiede (come ad esempio pubblicare in maniera autonoma un messaggio in bacheca), il record verrà automaticamente condiviso e sulla bacheca dell'utente, sarà visualizzato un messaggio che indica il punteggio effettuato dal giocatore, oltre a un logo identificativo per l'applicazione NinjaTyping e un link all'applicazione sul Google Play Store. Con la sezione dedicata alla condivisione su Facebook si conclude questo paragrafo per lasciare spazio a quello dedicato all'implementazione della sezione statistiche e del database locale.

3.8 Sezione statistiche e database locale

Attraverso l'Activity MainMenùActivity, è possibile premendo il pulsante *Record*, accedere alla schermata che permette di visualizzare le più importanti statistiche di NinjaTyping. Questa schermata è implementata attraverso la classe *RecordActivity,* che presenta una semplice interfaccia comprensiva di cinque pulsanti (Button), che permetteranno di accedere alla lista dei migliori 10 punteggi effettuati dall'utente giocando in Single Game, dei 100 migliori punteggi totalizzati da tutti gli utilizzatori dell'app e alla lista che riassume tutte le partite online che l'utente ha effettuato. Quindi la schermata in se è molto banale per implementazione, come è possibile osservare nello Screenshot riportato in *figura 3.7*.

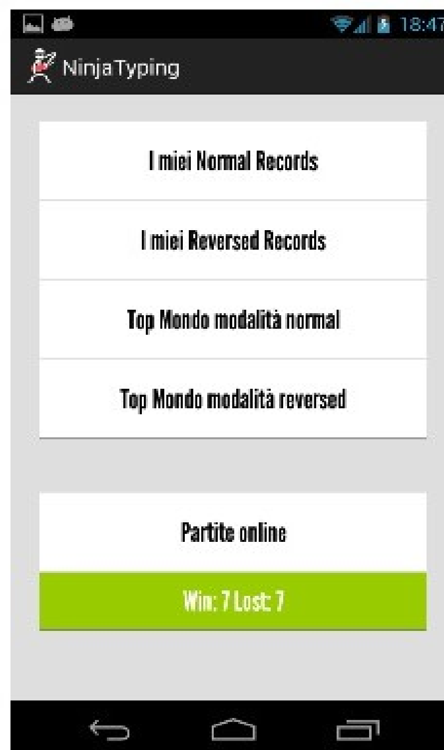


Figura 3.7 Schermata delle statistiche

La costruzione dell'Activity non è del tutto statica, infatti attraverso l'ultimo pulsante (Partite Online), viene anche inserita una sorta di *preview* sotto forma testuale, che indica il computo totale di partite giocate dall'utente, dividendole in sconfitte e vittorie. I dati che compongono questa preview, vengono estrapolati dal database attraverso l'apposito metodo (della classe MyDatabase)

```
public int[] getWinLosePreviewRecord()
```

che restituisce un vettore di interi con solo due elementi, in cui nel primo elemento saranno indicate il numero di vittorie effettuate dall'utente e nel secondo il numero di sconfitte. Prima di procedere mi sembra doveroso fare una illustrazione del database locale utilizzato in NinjaTyping. Il database locale chiamato db_ninja_typing, è definito nella classe *DatabaseNinjaTyping*. Questa classe estende SQLiteOpenHelper definita nel package android.database.sqlite (disponibile dalle API LEVEL 1). Il database è composto da 3 tabelle in seguito riportate:

- **ALL SCORES** (_id, errors, score, time, type): questa tabella conterrà tutti i punteggi che l'utente ottiene giocando una partita in *single game*. A prescindere dal fatto che l'utente sia registrato o meno, tutte le partite saranno salvate nella tabella automaticamente.
- **TOP_TEN_SCORES** (_id, errors, score, time, phrase, type): questa tabella è per composizione simile a quella precedente. Ho deciso di creare una tabella apposita per contenere i 10 migliori record (suddivisi per tipologia di partita giocata), per limitare i tempi di attesa di consultazione delle statistiche. Infatti se non avessi utilizzato questo metodo, l'alternativa sarebbe stata ottenere ogni volta 10 record tra tutti quelli presenti in tabella ALL_SCORES, il che sarebbe stata un'operazione molto lenta in caso le partite memorizzate in tale tabella fossero un numero molto elevato.
- **ONLINE_GAMES**(id_game, my_score, opponent_score, opponent_username, played): questa tabella verrà utilizzata per gestire il meccanismo delle partite online che l'utente dovrà giocare o avrà giocato. Sarà presente sul database del server una tabella identica.

Attraverso queste semplici tre tabelle riuscirò a gestire gran parte dei dati presenti in NinjaTyping. La classe che si occupa invece di inserire, modificare ed eliminare i record dal database è la classe MyDatabase. Tornando all'interfaccia riportata precedentemente in *figura 3.7*, l'utente potrà, attraverso l'apposita pressione dei pulsanti, visualizzare una lista con le varie statistiche. Se l'utente sceglierà di premere il primo pulsante, il sistema

lancerà la schermata gestita dalla classe *MyTopTenRecordsActivity,* passandogli il contenuto nella costante *GAME_MODE_NORMAL,* definita nella classe *Utils.* Se l'utente premerà il secondo *Button*, allora il sistema lancerà sempre la classe MyTopTenRecordsActivity, ma il parametro costante passatogli sarà questa volta *GAME_MODE_REVERSED.* Per visualizzare invece le statistiche riguardanti i migliori punteggi effettuati dagli utenti mondiali di NinjaTyping, l'utente potrà premere i secondi due pulsanti dedicati. Il funzionamento è identico a quanto appena visto per i record personali, quello che cambia è l'Activity dedicata alla visualizzazione della lista, che sarà *TopWorldRecordsActivity* e soprattutto il database dal quale attingere i dati. Infatti come detto in fase di progettazione, i migliori punteggi degli utenti di NinjaTyping, saranno contenuti solo sul server e non nel database locale. Procedo ora con l'illustrazione nel dettaglio di queste due classi (Activity) appena citate. La classe MyTopTenRecordsActivity, come detto prenderà in ingresso una costante che sta a indicare il tipo di punteggio che vogliamo estrapolare dal database. Lo sviluppo dell'Activity procederà invocando su un oggetto di tipo MyDatabase il metodo

```
public ArrayList<RecordDataWrapper> getTopRecords(int type)
```

che come prevedibile, prende in ingresso il tipo di partita da estrarre dal database. La tabella dalla quale estrarre i dati è TOP_TEN_SCORES. Il metodo restituirà i migliori record attraverso un ArrayList di oggetti di tipo *RecordDataWrapper,* che servirà a contenere tutti i dati della partita presenti in tabella. Una volta acquisiti i dati, bisogna inserirli nell'interfaccia grafica, dichiarata come sempre attraverso l'apposito file xml. In questo caso ho utilizzato per la visualizzazione dei dati, un componente fondamentale di Android chiamato *ListView,* definito nel package android.widget. Un componente di questo tipo permette la visualizzazione di una serie di informazioni sotto forma di liste di oggetti. Queste informazioni verranno gestite grazie a un altro componente chiamato *Adapter.* Android fornisce vari tipi di Adapter, ma per ottenere una completa personalizzazione grafica di ogni oggetto della ListView, ho deciso di creare un adapter "personale". La classe che si occupa di definire l'Adapter è MyTopRecordsAdapter ed estenderà la classe *BaseAdapter* (anch'essa definita nel package android.widget) per ereditarne i principali comportamenti. Il risultato grafico è quello visibile nello Screenshot in *figura 3.8* sottostante

Figura 3.8 Lista dei record personali

I punteggi vengono visualizzati in ordine d'importanza, ed inoltre se l'utente cliccherà su un qualsiasi elemento della lista, sarà possibile consultare la frase che è stata digitata, per ottenere il record in questione. Attraverso la pressione del tasto fisico *back*, l'utente tornerà alla schermata RecordActivity e potrà effettuare un'altra scelta, come per esempio la visualizzazione dei migliori record mondiali di NinjaTyping. La schermata che si occupa di gestire questi record, è definita come detto precedentemente attraverso la classe TopWorldRecordsActivity. Il funzionamento è pressoché identico rispetto a quanto visto nella classe MyTopTenRecordsActivity. L'interfaccia grafica sarà di fatto differente solo per l'aggiunta, in ogni elemento dell'ArrayList, del nome dell'utente (username) che ha effettuato un determinato record. Quello che cambia realmente è come ottenere i dati che l'Adapter dovrà gestire. Infatti nella classe MyTopTenRecordsActivity, i dati venivano estratti dal database locale. In TopWorldRecordsActivity, invece i dati verranno chiesti direttamente al server. La metodologia sarà sempre quella vista fini qui e utilizzata per registrarsi, oppure per

inviare il proprio punteggio al server. Quindi effettuerò una richiesta http utilizzando gli appositi oggetti messi a disposizione dalle API di Google. Invierò al server la costante che era stata presa in ingresso dall'Activity TopWorldRecordsActivity, riguardante la modalità di gioco della quale vogliamo ottenere i vari record. Lato server attraverso un apposito script *php,* effettuerò una query sulla tabella che gestisce i 100 migliori record (per ogni modalità di gioco), e restituirò i vari dati attraverso oggetti JSON. Questi oggetti saranno poi letti lato Client, ed inseriti in un ArrayList di oggetti RecordDataWrapper (visti in precedenza) che poi l'Adapter modellerà (anche graficamente) per renderli visibili nella ListView. Il risultato visivo è pressoché identico all'Activity precedente che gestiva i record personali, come visibile in *figura 3.9.*

Figura 3.9 Lista dei migliori record mondiali

Nello Screenshot sopra riportato, è possibile osservare come l'unica differenza nella composizione della ListView, rispetto a quella vista nell'Activity MyTopTenRecordsActivity, sia dettata dalla presenza degli username in ogni oggetto della lista. Se un utente è presente nella lista dei migliori 100 record, vedrà il proprio username di colore rosso, come nel caso dello Screenshot visibile nella figura appena

riportata. Anche in questo caso cliccando su di un qualsiasi elemento della listView, sarà possibile visualizzare la frase che ha permesso a un determinato utente di effettuare un tal record.
Concludo ora questo paragrafo dedicato alle statistiche, illustrando l'Activity che si occupa di esporre tutti i record effettuati dall'utente nelle partite online, in caso di pressione dell'apposito Button presente nell'Activity RecordActivity. L' attività che si occupa di gestire quest'ultima statistica, è implementata nella classe *OnlineRecordsActivity*. Anche in questo caso per il Layout grafico, sarà utilizzata una ListView, con un Adapter personalizzato che si occupa di gestire i dati. I dati degli incontri online disputati sono in un database locale, quindi mi avvarrò come al solito di un oggetto di tipo MyDatabase per recuperarli. La tabella che contiene tali dati è ONLINE_GAMES. Il metodo che si occupa di prelevare i dati interrogando la tabella è

```
public ArrayList<OnlineGameWrapper> getOnlineGamePlayed()
```

che restituisce il solito ArrayList, questa volta però contenente oggetti di tipo *OnlieGameWrapper,* utili a gestire i dati di una partita online. Dal nome del metodo, si può intuire la particolarità di questa tabella rispetto alle altre. Infatti è indicativo come il metodo restituisca i punteggi delle partite “giocate”, perché come visto in fase di progettazione e come spiegherò nell'apposito paragrafo dedicato, la tabella ONLINE_GAMES, contiene anche le partite che l'utente deve ancora giocare e che quindi ha, come si dice in gergo, in pending. Con questo precisazione, chiudo il paragrafo dedicato alle statistiche, procedendo attraverso appositi paragrafi, con l'implementazione della sezione Online Games.

3.9 Ricerca di un avversario online, Client

L'utente attraverso il menù principale, può scegliere di effettuare una partita online. La condizione necessaria affinché l'utente possa farlo, è che abbia eseguito la registrazione/accesso. In caso contrario, quando l'utente cercherà di accedere a tale sezione, verrà reindirizzato alla sezione Accedi (vedi *paragrafo 3.2*), in modo che possa

registrarsi. Se l'utente ha invece effettuato l'accesso, gli verrà mostrata l'Activity *FindMatchActivity*, che avrà lo scopo di permettere all'utente di cercare un avversario (attraverso l'apposito pulsante) e di giocare le partite trovate (attraverso un'apposita lista di partite).

Quando un utente premerà l'apposito pulsante nell'Activity FindMatchActivity, verrà attivato il meccanismo che cerca un avversario in background. Questo vuol dire che l'utente potrà continuare a navigare nelle varie schermate dell'applicazione oppure uscire dall'applicazione, mentre il sistema cerca un avversario. Quando un avversario verrà trovato, arriverà una notifica (simile a quella degli sms) che indica la riuscita dell'operazione. Tutto questo meccanismo in background è possibile in Android grazie all'implementazione di una classe che estenda la classe Service, definita in android.app.Service. Un servizio è un componente applicativo, messo a disposizione degli sviluppatori a partire dalle API LEVEL 1, che può essere intesa come una sorta di Activity "nascosta" all'utente, che quindi non interagisce direttamente con lui, ma ne esegue un servizio richiesto.

Quindi nel momento in cui l'utente premerà il pulsante per la ricerca di un avversario, verrà lanciato il servizio apposito che si occuperà di interagire con il database del server. All'interno del servizio, grazie ad un ciclo contenuto in un apposito *Thread*, eseguirò una richiesta al server ogni tot di secondi, per cercare un avversario. La richiesta al server viene effettuata nel solito modo utilizzato nel corso di tutto il progetto, quindi attraverso richieste Http con risposte di tipo JSON. Al server verranno inviati i seguenti due parametri:

- **username**: stringa che indica il nome dell'utente.
- **id_request**: intero che indica l'identificativo della richiesta.

La prima volta che l'utente effettua una chiamata al server, l'id_request conterrà il valore "-1". In questo modo il server intenderà la richiesta come un messaggio del tipo "*voglio giocare una partita, inserisci la mia richiesta nella tabella*". Il server genererà una risposta composta da 3 parametri:

- **id_request**: l'identificativo della richiesta dell'utente nel database.
- **opponent_username**: il nome dell'avversario.
- **state**: lo stato della richiesta.

Questi tre parametri verranno salvati in apposite variabili e nel caso di id_request, verrà

sovrascritto il precedente valore della variabile che era "-1". La risposta del server è da intendere in due differenti maniere. Infatti il server può indicare attraverso quei parametri, che è stata inserita una richiesta di partita nel database, ma è anche già stato trovato un avversario. Se così non fosse i parametri indicherebbero che la richiesta di partita è stata inserita nel database, ma ancora non vi è un avversario disponibile. In quest'ultimo caso, il campo opponent_username avrà valore *null,* mentre il campo state avrà valore "-1", che è da intendersi come "in attesa". Se l'avversario non è stato trovato, il Service procederà dopo un tot di secondi ad inviare una nuova richiesta, al server, passandogli come id_request il valore che ci è stato precedentemente restituito dal server. Questo caso sarà inteso dal server come "*l'utente ha già effettuato una richiesta, vuole controllare se è stato trovato un avversario per quella richiesta*". Questo procedimento andrà avanti finché non verrà trovato un avversario. In questo caso i parametri restituiti dal server avranno il seguente significato:

- **state**: non conterrà più il valore "-1", bensì conterrà un valore intero maggiore di "-1" che indicherà l'identificativo della partita che l'utente dovrà giocare.
- **opponent_username**: conterrà il nome dell'avversario contro cui l'utente deve effettuare la partita.

Questi concetti saranno più chiari quando spiegherò la parte server dedicata alla restituzione di tali dati. Una volta che l'utente avrà quindi ottenuto un avversario contro cui giocare la partita, il servizio verrà interrotto e procederò con il creare un record nel database locale, nella tabella ONLINE_GAMES (introdotta nel *paragrafo 3.8*). Il metodo che si occupa di creare il record in tabella è il seguente

```
public void setOnlineRecord(int idGame, Object myScore, Object
opponentScore, Object opponentUsername, int played).
```

Il campo idGame che inserirò in tabella, sarà l'identificativo della partita da giocare e verrà creato grazie al valore indicato nel campo *state,* restituitomi dal server. Questo valore sarà essenziale nel momento in cui l'utente giocherà una partita e dovrà controllarne l'esito. Un campo fondamentale che compone il record del database è l'intero *played.* Infatti come anticipato in fase di progettazione questo campo (che inizialmente sarà "0"), identificherà lo stato di una partita, permettendo al sistema di

capire quali partite sono ancora da giocare, quali invece sono in attesa di un esito e quali sono state giocate. Il campo played può assumere tre valori differenti, che corrisponderanno ad altrettanti differenti stati. Questi valori sono indicati da tre costanti così definite nella classe Utils:

```
public static final int PLAYED = 1;
public static final int PLAYED_BOTH = 2;
public static final int NOT_PLAYED = 0;
```

Se il record partita online nel database locale avrà uno stato di tipo *PLAYED,* vuol dire che l'utente ha giocato la partita, ma è ancora in attesa di conoscere l'esito dell'avversario. Se il record partita avrà uno stato di tipo *PLAYED_BOTH,* allora la partita è stata giocata da entrambi gli utenti quindi può definirsi conclusa in quanto ne conosco l'esito. Infine se la partita avrà stato *NOT_PLAYED*, allora vorrà dire che l'utente deve ancora giocare la partita stessa.

Quindi nel caso in cui fosse stato trovato una avversario, la partita appena salvata nel database comparirà nell'Activity FindMatchActivity e l'utente avrà la possibilità di giocarla attraverso un tocco (come se fosse un pulsante). Come il sistema invierà i dati della partita giocata e come riuscirà ad ottenere il risultato della partita stessa, verrà spiegato nell'apposito paragrafo. Nel successivo paragrafo spiegherò invece come fa il server a gestire le richieste di partite online effettuate dai vari utenti, restituendo i parametri “opponent_username”, “id_request” e “state”, all'utente come visto in precedenza.

3.10 Ricerca di un avversario online, Server

Nel paragrafo precedente, ho descritto lo sviluppo di una richiesta di partita online lato Client. In questo paragrafo invece, mi occuperò di descrivere come il Server gestisce le richieste di partite online degli utenti. Innanzitutto la tabella che ha il compito di memorizzare le richieste degli utenti nel database del Server, sarà composta da soli tre campi:

– **id:** è l'identificativo di una determinata richiesta da parte di un utente

- **id_user:** è l'identificativo dell'utente che effettua una richiesta.
- **State:** è lo stato della richiesta. Se vale “-1”, non è stato trovato un avversario per l'utente che lo ha richiesto. Se vale “>1” allora l'utente ha trovato un avversario e la richiesta è stata soddisfatta.

Per descriverne il protocollo, partirò dal caso in cui non ci siano record nella tabella del database. Quindi il server riceve una richiesta da parte di un utente, che come visto in precedenza, avrà valore id_request “-1”. Il server esegue due operazioni, cioè inserisce in tabella la richiesta dell'utente e crea un oggetto JSON da restituire al Client. In *figura 3.10* una rappresentazione grafica per quanto appena detto.

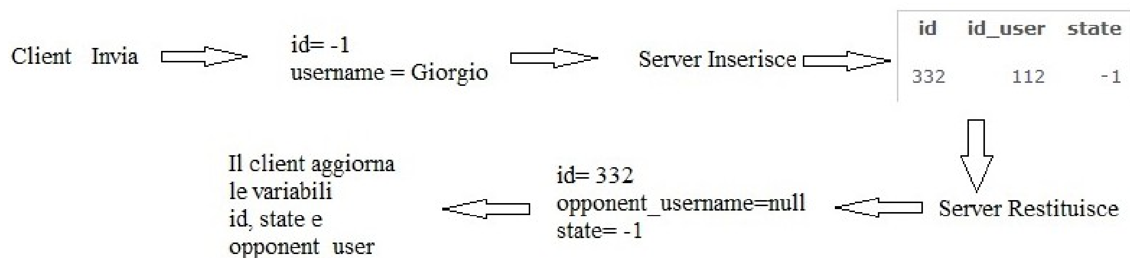


Figura 3.10 Comunicazione Client-Server per la richiesta di gioco online

Ora supponiamo che un altro utente effettui la richiesta di partita. A questo punto il server controllerà se vi sono altri utenti in tabella in attesa di una partita e se ci sono, aggiornerà il record del database e allo stesso tempo procederà a creare due record di una nuova tabella, dedita allo svolgimento delle partite trovate. Inoltre restituisce un oggetto JSON al client che ha effettuato l'ultima richiesta. In *figura 3.11* un contributo grafico per quanto appena detto.

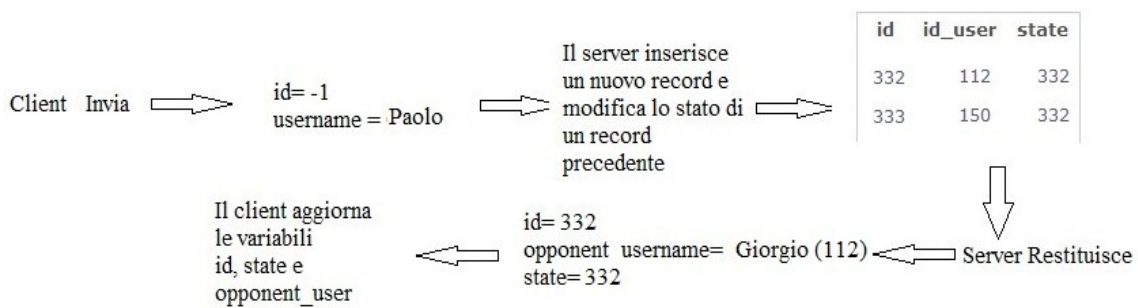


Figura 3.11 Comunicazione Client-Sever per la richiesta di gioco online Utente 2

L'utente che ha effettuato l'ultima richiesta, avrà quindi immediatamente una risposta affermativa e potrà procedere lato client a giocare la partita. E' molto importante notare il valore del campo *state.* Infatti il valore non sarà "-1", ma avrà un numero positivo, e sarà pari all'identificativo della richiesta effettuata dall'avversario. Anche l'avversario vedrà modificarsi il campo *state* da "-1" ad un valore pari al proprio identificativo. Questo nuovo valore del campo *state*, sarà una sorta di identificativo partita, ed è da intendersi come "*gli utenti Giorgio e Paolo giocano la partita 332*". In questo modo avrò la certezza che una partita sarà identificata correttamente ed esisteranno quindi solo coppie di identificativi.

Nello scenario sopra evidenziato, l'utente Paolo ha cercato la partita e ha trovato un avversario, ma l'utente Giorgio ancora non sa di averlo trovato. Lo scoprirà nel momento in cui il Client effettuerà una nuova richiesta avente parametro "id_request= 332". Il server infatti quando leggerà un valore id_request maggiore di "-1", andrà a controllare lo stato del record avente l'id_request passato in ingresso. Nel momento in cui il server troverà un valore nel campo *state* pari a 332, procederà cercando nella tabella un secondo record con il campo *state* pari a 332. Nell'esempio precedente troverà il record con id "333", appartenente all'utente Paolo e provvederà a formulare una risposta JSON, contenente i dati opportuni per permettere all'utente Giorgio di giocare una partita contro Paolo.

In precedenza ho scritto che il server, nel momento in cui trova un partita, oltre ad aggiornare lo stato delle richieste, crea due appositi record in una tabella dedicata allo sviluppo del gioco. I record creati, considerando i dati dell'esempio precedente, saranno come quelli in *figura 3.12*.

id	id_game	id_user	score
370	332	112	*NULL*
371	332	150	*NULL*

Figura 3.12 Tabella per la gestione delle partite online

Inizialmente i record avranno uno score pari a *null,* in quanto nessuno dei due giocatori avrà effettuato la partita. Nel momento in cui una partita verrà effettuata, i record verranno appositamente aggiornati. Come e quali dati l'utente invierà al database per indicare il risultato della propria partita e conoscere l'esito dell'avversario, verrà trattato nel prossimo paragrafo.

3.11 Gioco della partita online e verifica dell'esito

Attraverso l'Activity FindMatchActivity, è possibile cercare un avversario premendo l'apposito pulsante. Ma non solo. Infatti nel momento in cui verrà trovato un avversario, la partita da giocare sarà visibile sempre nella medesima Activity, con un risultato come quello in *figura 3.13.*

Figura 3.13 Schermata per la gestione delle partite online

Se l'utente premerà nuovamente il pulsante “cerca avversario”, troverà un'altra partita,

che andrà ad aggiungersi a quella precedente, creando quindi una lista di partite da giocare. Queste partite da giocare sono memorizzate nel database locale, nella tabella ONLINE_GAMES (introdotta nel *paragrafo 3.8* ed approfondita nel *paragrafo 3.9*). Le partite da giocare saranno quelle con valore del campo *played* pari alla costante (definita nella classe Utils) NOT_PLAYED. Cliccando quindi su un elemento della lista “partite da giocare” (vedi *figura 3.13*), l'utente avrà la possibilità di giocare la partita. La schermata che si occupa di permettere all'utente di giocare la partita, è definita nella classe *GameOnlineActivity*, che per sviluppo grafico e modalità di gioco, si presenta in maniera identica alla GameActivity, che si ottiene nel momento in cui si cerca di giocare una partita singola in modalità “normal”. Una volta finita la partita il controllo passerà all'Activity *GameOnlineRecapActivity,* che per composizione grafica è invece identica all'Activity GameRecapActivity, che l'utente ottiene quando finisce di giocare una partita in single player. Vi è però una differenza sostanziale (accennata nel capitolo dedicato allo sviluppo), nell'implementazione di questa Activity di recap, rispetto a quella del single player. Infatti nel momento in cui l'utente gioca una partita online, l'Activity GameOnlineRecapActivity provvederà ad inviare al server i dati della partita giocata. Il Client, più precisamente invierà al server i seguenti parametri:

- **score**: il punteggio effettuato in partita.
- **username**: il nome dell'utente che ha effettuato il punteggio
- **id_game**: l'identificativo della partita giocata.

Il server provvederà ad inserire questi dati, aggiornando il record apposito nella tabella (riportata nel *paragrafo 3.10* in *figura 3.12*). Se l'operazione avrà successo il server restituirà, tramite JSON, un valore booleano *true.* Il Client attraverso l'Activity GameOnlineRecapActivity, una volta ricevuta risposta positiva dal server, effettua due operazioni. Innanzitutto, provvede ad aggiornare il record del database locale corrispondente alla partita giocata. Il metodo (definito nella classe MyDatabase), che si occupa di questo aggiornamento è il seguente

```
public int updateMatchPlayed(int idGame, Object myScore).
```

Il metodo inserisce il punteggio nella partita corrispondente all'identificativo passato in

ingresso. Inoltre, sempre mediante questo metodo, viene modificato il campo *played* del record appartenente alla partita in questione. Il campo *played,* cambia stato e passa da *NOT_PLAYED* a *PLAYED.* In questo modo la partita non sarà più presente nella lista delle partite da giocare(e quindi non sarà "rigiocabile"), ma passerà alla lista delle partite giocate e in attesa dell'esito.
La seconda operazione che compie l'Activity GameOnlineRecapActivity, è proprio attivare il meccanismo che permette all'utente di conoscere l'esito della partita giocata. Il meccanismo sarà molto simile a quanto visto nell'ambito di ricerca di un avversario. Infatti, verrà lanciato un Service, definito in un'apposita classe, che si occuperà di chiedere al server, una volta ogni tot di secondi, l'esito di alcune partite. Ho usato il plurale in quanto l'utente potrebbe voler verificare l'esito di più partite giocate e non solo di una. Quindi il Service si occuperà di prelevare dal database la lista di partite online, che avranno il campo *played* con valore *PLAYED.* Questa lista sarà inviata quindi al server, come al solito attraverso un apposito oggetto http. Per lista di partite, intendo semplicemente la lista di identificativi delle partite del quale l'utente attende il record. Oltre a questa lista invierò anche lo *username* dell'utente che ha fatto questa richiesta al server. Il server d'altra parte provvederà ad interrogare la tabella che memorizza le partite online, per controllare se sono state giocate dall'avversario dell'utente che ne ha fatto richiesta. Il server restituirà a sua volta una lista di oggetti JSON composta da questa coppia di parametri:

- **id_game**: l'identificativo della partita del quale abbiamo ottenuto un risultato
- **score**: il punteggio effettuato dall'avversario nella determinata partita.

Quindi il server restituirà la lista delle partite che sono state giocate anche dall'avversario. A questo punto, il client farà un ultimo passo. Provvederà ad aggiornare nella tabella ONLINE_GAMES, le partite appena controllate, che sono state restituite dal server. Lo farà attraverso un altro metodo della classe MyDatabase, che avrà il compito di aggiornare i campi *opponent_score* e *played*. Il campo opponent_score conterrà ovviamente il punteggio effettuato dall'utente, mentre il campo played, passerà dall'avere un valore PLAYED, ad un valore di tipo PLAYED_BOTH. In questo modo vorrà dire, che la partita in questione è stata giocata e l'utente ne potrà conoscere l'esito.

Il Service d'altro canto continuerà a controllare gli esiti delle partite (facendone richiesta al server), finché nella tabella locale ONLINE_GAMES, ci saranno dei record con il campo played con valore PLAYED. Se non ci saranno più record di quel tipo nel database locale allora il Service verrà distrutto, fino a che l'utente non giocherà una nuova partita online.

Con quest'ultimo impegnativo paragrafo, si conclude il *capitolo 3,* dedicato allo sviluppo dell'applicazione NinjaTyping. Ho cercato di spiegare in maniera più chiara possibile, le tante meccaniche che si celano alla base di un gioco apparentemente molto semplice e banale. Ho tentato inoltre di sviluppare NinjaTyping, in modo da poter in futuro aggiungere funzionalità e modalità di gioco, senza stravolgere troppo l'architettura generale del codice. Il prossimo capitolo, sarà quindi dedicato all'esposizione degli sviluppi futuri che prevedo per la mia applicazione.

Capitolo 4
Sviluppi futuri del progetto NinjaTyping

In questo quarto e ultimo capitolo di stesura della tesi, analizzerò le possibili modifiche o aggiunte future da apportare all'applicazione NinjaTyping. Toccherò quindi vari argomenti, quali l'implementazione di nuove modalità di gioco in single player, l'implementazione di nuove funzioni per quanto riguarda la sezione online game, oltre ad un possibile eventuale sviluppo di NinjaTyping per altre piattaforme e infine qualche considerazione generale riguardo strategie commerciali.

4.1 Database e Web site

Per la memorizzazione dei dati lato server, ho utilizzato un database fornitomi dall'azienda Aruba S.p.A. della capienza di 100 MB. Per la mole di dati che verranno memorizzati attraverso NinjaTyping, è una capienza assolutamente accettabile. Infatti facendo una piccola stima, se venissero per esempio giocate ogni giorno 10000 partite (un numero assolutamente esorbitante), i dati complessivi storicizzati in un mese sarebbero poco più di 5 MB. La stima fatta è a dir poco ottimistica. Se avessi realmente un numero di partite giornaliere così elevato, avrei ben altri problemi del quale occuparmi oltre alla semplice capacità del database.

Il discorso cambia però nel caso io volessi offrire un'esperienza più ampia all'utente. In NinjaTyping, i punteggi personali effettuati dall'utente, vengono salvati solo nel database locale dell'applicazione. Se un utente accede con lo stesso account su due terminali diversi, non avrà la possibilità di visualizzare su entrambi i terminali gli stessi record locali. Per aggiungere questa caratteristica a NinjaTyping, dovrei salvare i dati personali di ogni utente, oltre che sul database locale, anche sul database nell'apposito server dedicato di Aruba. Considerando che una partita singola in NinjaTyping può essere giocata in pochissimi secondi, le partite inviate al server (e di conseguenza la

memoria occupata) potrebbero essere un numero elevatissimo, nel caso in cui l'applicazione avesse un domani un buon successo. In questo caso quindi, se volessi implementare questo tipo di funzione dovrei seriamente considerare un *Upgrade* del database a mia disposizione, in quanto 100 MB, potrebbero non bastare sul lungo periodo.

A livello di architettura del database invece, vorrei effettuare un'importante aggiunta che permetterebbe piacevoli novità nell'ambito delle classifiche di NinjaTyping. Al server dedicato acquistato su Aruba, è associato anche un dominio per un eventuale sito Web. L'idea è quella di creare un *Web Site* dedicato esclusivamente ai punteggi e alle statistiche più varie e disparate ottenute dagli utilizzatori di NinjaTyping. Per il tipo di statistiche che sono ora previste dall'architettura client/server che ho implementato, il sito dedicato in questione sarebbe un po' spoglio. La mia volontà è quella di aggiungere (inizialmente), un parametro ad ogni record effettuato giocando una partita single player dall'utente. Questo parametro mi servirà per capire con quale dispositivo mobile(inteso come marca, dimensione schermo ecc), un utente ha ottenuto un determinato risultato. Intendo quindi costruire un sito Web, con statistiche dedicate a varie tipologie di record, come per esempio ai migliori record effettuati da utilizzatori di Smartphone, oppure i migliori record dedicati agli utilizzatori di Tablet. Inoltre attraverso qualche particolare interrogazione sulle tabelle, potrei anche offrire qualche curiosità, come ad esempio far conoscere attraverso quale tipo di dispositivo vengono ottenuti i record mediamente più alti. Ho elencato solo alcune tra le tipologie di statistiche possibili implementabili, resta il fatto del concetto di fondo principale e cioè costruire un sito internet dedicato a NinjaTyping.

Nei prossimi paragrafi andrò a trattare i possibili miglioramenti dell'applicazione ad ampio raggio, quindi dai semplici miglioramenti grafici, ai più complessi miglioramenti prestazionali.

4.2 Miglioramenti prestazionali dell'applicazione

Ho cercato di sviluppare NinjaTyping, facendo attenzione alle prestazioni dell'applicazione, in termini di velocità e peso specifico. Penso di aver ottenuto ottimi

risultati in quanto l'applicazione ha una dimensione di 2,7 MB (dizionari di italiano e inglese compresi) e ha tempi d'attesa pressoché nulli. C'è però una parte dell'applicativo che non offre prestazioni accettabili, in determinate situazioni di gioco e che sicuramente in futuro subirà importanti modifiche. Sto parlando della sezione game online e nello specifico del meccanismo che permette all'utente di trovare un avversario contro cui scontrarsi e successivamente riceverne l'esito della partita. Come riportato nel terzo capitolo dedicato allo sviluppo, ho implementato il tutto utilizzando i *Service,* cioè componenti messi a disposizione degli sviluppatori per effettuare determinate azioni in *background.* Tali servizi, hanno però un grosso difetto. In caso di utilizzo prolungato infatti, il consumo di batteria è altamente dispendioso e sicuramente è qualcosa che l'utenza non apprezza.

Principalmente per questo motivo, intendo quindi sostituire in parte i *Service,* implementando le *push notification.* Le notifiche di push in android, sono possibili grazie ad un servizio gratuito offerto da Google, chiamato *Google Cloud Messaging* (*GCM*). Tale servizio, consente agli sviluppatori di inviare dati dai server alle proprie applicazioni sviluppate. Si tratta quindi del meccanismo inverso a quello attualmente implementato in NinjaTyping, in cui è l'applicazione che effettua continue richieste al server per conoscere determinate informazioni. Con questa nuova modalità quindi, potrò riuscire a rendere il meccanismo di gestione dell'online game, sicuramente più veloce e soprattutto più prestante.

4.3 Miglioramenti Grafici

In un'App mobile come NinjaTyping, è molto importante che la grafica sia per lo meno accettabile. Nel mio caso ho implementato una grafica minimalista, ma chiara e soprattutto il più “pulita” possibile. Ritengo quindi che il Layout grafico sia assolutamente accettabile. Il fatto che sia accettabile, non vuol dire che in futuro non debba subire variazioni. Infatti NinjaTyping, ha probabilmente un'unica evidente carenza di Layout e mi riferisco al fatto che non presenta alcun effetto grafico. Ad onor del vero, non sono molte le sezioni di gioco della mia applicazione, in cui vi è un impellente bisogno di effetti, ma altrettanto sicuramente, è innegabile che tali effetti

renderebbero il tutto più piacevole.

I primi miglioramenti che ho in mente, riguardano due schermate in particolare. La schermata di gioco vera e propria in cui l'utente effettua una partita e la schermata immediatamente successiva, che permette all'utente di visualizzare i dati della partita appena effettuata. Prendendo in considerazione quest'ultima schermata, pensavo semplicemente di far comparire tutte le statistiche, in sequenza con un effetto stile *Fade*. Certamente nulla di rivoluzionario, ma comunque un piccolo passo avanti. Inoltre nel caso in cui l'utente abbia appena effettuato un personal record, vorrei inserire una immagine di congratulazioni, al posto del semplice messaggio a schermo.

Ma queste sono comunque piccolezze che passano in secondo piano. Il vero effetto grafico, è sicuramente da implementare nella GameActivity, cioè la schermata di gioco vera e propria. L'idea è banale, ma al tempo stesso gradevole. Vorrei infatti inserire un'animazione, nel momento in cui l'utente digita un carattere e di conseguenza la cella della frase disegnata su schermo, cambia stato e quindi colore. La prima cosa che ho in mente, è cambiare il colore rappresentante lo stato della cella in maniera graduale, con diverse sfumature dal colore bianco al rosso (in caso di errore), o dal colore bianco al verde (in caso di tasto premuto correttamente). Inoltre nel momento in cui avviene questo cambiamento di colore, il carattere della frase premuto deve diventare un po' più grande rispetto agli altri, per poi tornare alle normali dimensioni a cambiamento di stato (e colore) acquisito.

Sempre per quanto riguarda questa schermata di gioco, vorrei anche inserire un altro effetto che non è da catalogarsi come “grafico”, ma come “acustico”. Mi piacerebbe infatti inserire un suono ad ogni carattere digitato con successo e un suono (ovviamente acusticamente opposto al precedente) in caso di carattere digitato in maniera errata.

Nulla di trascendentale insomma, ma miglioramenti abbastanza utili per alzare un po' il livello della mia App, a patto però che questi cambiamenti non vadano ad incidere troppo negativamente, sulle prestazioni in termini di fluidità e “pesantezza” dell'applicazione. Riguardo al “peso” dell'applicazione, ne approfitterò per introdurre il prossimo paragrafo in cui tratterò, tra le altre cose, la possibilità di aumentare le lingue e quindi i dizionari interni all'applicazione.

4.4 Nuova modalità di gioco e dizionari stranieri

NinjaTyping è un'applicazione multilingua, dove per multilingua non intendo la semplice composizione dei menù, bensì l'intera applicazione nella sua completezza, quindi anche per quanto riguarda le frasi casuali che l'utente digita in fase di gioco. Come spiegato nel capitolo dedicato allo sviluppo, il meccanismo che permette la creazione delle parole da digitare, prevede l'uso di un dizionario (limitato) dal quale attingere tali parole. I dizionari sono per ora due. Quello di italiano (102500 parole circa) e quello di inglese (73000 parole circa). Entrambi i dizionari saranno caricati nell'applicazione, a prescindere dal fatto che essa venga scaricata dal market italiano o da quello inglese. Ho fatto questa scelta in quanto questi due dizionari hanno una grandezza totale di 1,66 MB e quindi non appesantiscono l'applicazione. Proprio tenendo conto di quest'ultima precisazione, è mia intenzione in futuro, introdurre nuovi dizionari e di conseguenza nuove lingue, aprendo quindi NinjaTyping a più parti del mercato mondiale. L'architettura dell'applicazione, mi permette di inserire lingue di gioco di un certo tipo, semplicemente con l'aggiunta di determinati dizionari filtrati. Per "lingue di un certo tipo", intendo dire quelle lingue che utilizzano l'alfabeto latino di base, cioè quello che consiste di 26 grafi (intesi come caratteri/lettere). Per "dizionari filtrati" intendo dire, dizionari ai quali sottrarrò le parole comprensive di accenti, apostrofi o altri caratteri speciali. Se una lingua può essere catalogata in questo sottoinsieme che ho appena descritto, allora potrà essere aggiunta in NinjaTyping praticamente senza alcuna modifica di codice. Inoltre considerando che il dizionario filtrato di italiano è probabilmente il più corposo (senza filtri normalmente quello inglese avrebbe più parole), potrei aggiungere anche altre 5 o 6 lingue, con il minimo sforzo e soprattutto mantenendo l'applicazione ad una dimensione inferiore ai 10 MB.
Il discorso è differente in caso io voglia introdurre lingue come ad esempio il Russo. In questo caso il dizionario sarebbe composto da parole scritte in alfabeto cirillico. Questo comporterebbe diversi effetti collaterali, come ad esempio il dover creare un'apposita tastiera cirillica per permettere all'utente di digitare la frase random su schermo. Nulla di insormontabile, ma richiederebbe tempo e attenzione che forse, almeno inizialmente, è meglio dedicare a migliorare altri aspetti di NinjaTyping.

Uno di questi aspetti migliorabili, riguarda sicuramente l'aggiunta di (almeno) una terza modalità per quanto riguarda il single player, oltre a quelle già presenti (modalità reversed e modalità normal). Ci sono diverse idee che potrei sviluppare implementando più modalità. Potrei ad esempio creare una modalità leggermente più difficile rispetto alla *Reversed* (che ricordo, è caratterizzata da una tastiera specchiata, invertita), andando a creare una tastiera composta dai soliti caratteri dell'alfabeto latino, ma disposti su schermo non in maniera *qwerty,* bensì in maniera casuale.
Un'altra idea è invece quella di inserire nella frase da digitare, anche altri caratteri come quelli numerici, oppure quelli di punteggiatura. Ovviamente di conseguenza dovrei creare anche un'apposita tastiera.
Questi sono solo due esempi che indicano il tipo di aggiunta che vorrei effettuare in NinjaTyping. A livello di sviluppo, dovrei andare a modificare (o creare) solamente algoritmi lato Client, in quanto l'aggiunta di nuove modalità di questo tipo, che non snaturano le parti fondamentali del gioco (punteggio, errori ecc.), possono essere già supportate lato server, per quel che riguarda la memorizzazione dei dati nelle varie classifiche o altro.
Le aggiunte riguardanti le modalità di gioco, non si fermano al single player, infatti nel prossimo paragrafo mostrerò qualche idea di sviluppo, che riguarda la parte di gioco online.

4.5 Sviluppi della modalità di gioco online

Nel paragrafo precedente ho parlato di aggiunte di nuove modalità riguardanti la parte single player. Se parlo invece di nuove modalità riguardanti l'online player, non parlo solo di aggiunte, ma anche di modifiche. Infatti ritengo che a livello didattico, la parte multiplayer sia stata la più interessante, stimolante e formativa, ma nonostante questo, non posso ritenermi totalmente soddisfatto. Trovo infatti che la modalità sia un po' spoglia e per avere più successo e consenso da parte degli utenti, abbia bisogno di modifiche ed aggiunte. Per quanto riguarda le modifiche, sono orientato a rendere NinjaTyping, ancora più simile di quanto già non sia ad applicazioni che prevedono una modalità multiplayer a turni, come *Ruzzle* o *QuizDuello*. Non ho certamente la

presunzione di arrivare (per lo meno sviluppando da solo), al livello di queste due applicazioni appena citate, che sono sicuramente il massimo ad oggi per Game app, con determinate caratteristiche. L'idea è però di avvicinare, come comportamento, il multiplayer di NinjaTyping a quello di Ruzzle. Per prima cosa, voglio quindi implementare oltre alla funzione già presente "cerca un avversario", la funzione "sfida un amico", che penso sia essenziale in un typing game online a turni. Inoltre potrei modificare la sfida rendendola non più una partita "secca", ma una partita sviluppata su più livelli. In questo caso due utenti si scontrano giocando una sola partita (inteso come un solo risultato finale), che però si sviluppa su due o più round, dove ogni round è caratterizzato da una delle modalità implementate in single player (normal, reversed ecc.). In questo modo ci sarebbe probabilmente più grado di sfida, anche se non è detto che in fin dei conti, possa realmente piacere di più rispetto alla sfida secca che ho già implementato. Quindi è una fase di sviluppo da ponderare bene, tenendo in considerazione le possibilità di successo e il tempo che richiederebbe uno sviluppo di questo tipo.

Le mie idee di sviluppo riguardanti le partite online, non si fermano però qui. Ho infatti intenzione di andare a creare qualcosa di personalmente più ambizioso, dove per ambizioso intendo qualcosa che possa piacere, senza il bisogno di riproporre modalità di gioco già viste in altre applicazioni (come le già citate *Ruzzle* o *Quiz duello*). Sto pensando infatti di implementare la modalità "Live challenge". Si tratta di una modalità, in cui due utenti si sfidano in una partita con un singolo round e soprattutto in tempo reale. L'idea è quindi quella di proporre una schermata di gioco (per composizione della frase) identica per entrambi i giocatori, che dovranno cominciare la partita nello stesso istante. Questo non basta, perché la vera caratteristica dovrà essere rappresentata dal fatto che ogni utente vedrà sul proprio dispositivo, nella *View* dove è disegnata la frase, un cursore che rappresenta lo stato di avanzamento della partita dell'avversario. In questo modo penso di rendere il tutto più coinvolgente, anche se mi rendo conto di dover affrontare diversi problemi, tra i quali il ritardo di connessione che può rendere difficile una sfida realmente simultanea.

Con quest'ultima idea, si chiude il paragrafo dedicato alle implementazioni future dell'online game di NinjaTyping. Nel prossimo paragrafo andrò a considerare sviluppi riguardanti più che altro la parte commerciale, disquisendo quindi sulla possibilità di

inserire pubblicità oppure sviluppare l'applicazione anche per piattaforme diverse da quella Android.

4.6 NinjaTyping per iOS e scelte commerciali

Quando si vuole sviluppare un'applicazione, bisogna innanzitutto decidere per quale sistema operativo farlo e le scelte solitamente sono due. Farlo per dispositivi mobili che utilizzano Android, oppure dispositivi mobili Apple. Io ho scelto di sviluppare NinjaTyping per dispositivi mobili che utilizzano Android e l'ho fatto per diversi motivi. In *primis* per l'utilizzo del linguaggio di programmazione *Java*, che già conoscevo a differenza dell' *Objective-C*, con cui non avevo familiarità. Inoltre preferisco le regole più "aperte" del market di Google, rispetto a quelle rigide del market di Apple. Nonostante questo, è innegabile che le applicazioni di successo, siano al giorno d'oggi sviluppate per entrambi i sistemi operativi e solitamente tale successo, arriva prima per applicazioni scaricate dal market di Apple. Ci sono molti dati che giustificano questa affermazione, ad esempio in uno studio effettuato dalla compagnia specializzata in ricerche di mercato *Forrester Research,* viene evidenziato come su Android il 76% degli utenti utilizzi applicazioni di terze parti (quindi scaricate dal market), mentre su iOS ben l'89% degli utenti. Non posso quindi non considerare questi dati, quando penso agli sviluppi futuri di NinjaTyping e di conseguenza quasi sicuramente cercherò di riproporre la mia App anche per il market AppStore, appartenente ad Apple. Oltre al motivo precedentemente indicato, la giustificazione di questa scelta è da ricercare in realtà anche in altre considerazioni. NinjaTyping inizialmente supporterà due lingue, l'italiano e l'inglese. Presumibilmente quindi, lancerò l'applicazione per lo store Italiano e quello Americano. Android è sicuramente il sistema operativo più utilizzato al mondo, ma se limitiamo la ricerca ai soli Stati Uniti e all'Italia, la distribuzione dei due sistemi operativi è pressoché equivalente. Inoltre come riportato da diversi studi, negli USA è prevista nel 2014, una crescita di dispositivi mobili Apple del 4% a discapito dei dispositivi Android. Non posso quindi rinunciare ad una fetta di mercato così imponente, tale da aumentare le possibilità di un successo minimo dell'applicazione pari al 50%.

Un'altra considerazione importante, riguardante gli sviluppi futuri di NinjaTyping nell'ambito commerciale, è sicuramente la scelta di una strategia adatta ad un eventuale guadagno. Le strade di guadagno sono principalmente due. Permettere di scaricare l'applicazione dal market in maniera gratuita (*free* in gergo), inserendo dei banner pubblicitari (eventualmente rimovibili sotto pagamento apposito), oppure inserire l'applicazione nel market direttamente a pagamento. Mi sento di escludere assolutamente questa seconda ipotesi, per diversi motivi, ma in particolare per il semplice fatto che le applicazioni scaricate a pagamento nel Google Play Store di Android, sono un numero sensibilmente inferiore rispetto a quelle gratuite, come è possibile osservare in *figura 4.1*.

Comparison of Top 200 Apps

	Number of Reviews	Avg Rating	Average Price
Android Paid	3,382,465	4.39	$3.91
iOS Paid	6,093,675	4.22	$2.14
Android Free	61,898,436	4.34	Free
iOS Free	26,719,317	4.02	Free

	Number of Reviews	Average Rating	Average Price
Android Average	32,640,451	4.37	$1.96
iOS Average	16,406,496	4.12	$1.07

Figura 4.1 Confronto tra Android e iOS per applicazioni a pagamento e gratuite scaricate

La figura dimostra (basandosi su dati inerenti alle migliori applicazioni) come gli utenti utilizzatori di Android, preferiscano di gran lunga scaricare applicazioni gratuite. Diverso il discorso per gli utenti Apple che sono invece meno restii al pagamento delle app. Per entrambi i sistemi operativi la mia scelta ricadrà, per lo meno inizialmente, nell'inserire l'applicazione in maniera gratuita ma con pubblicità nei rispettivi market.

A tal proposito mi ricollego al terzo capitolo dedicato allo sviluppo dell'applicazione, in particolare all'implementazione grafica della GameActivity, cioè di quella schermata che permette effettivamente all'utente di giocare una partita. La schermata è composta

dalla frase da digitare che occupa sempre metà dello schermo e da una tastiera "fisica", che potrebbe occupare la seconda metà dello schermo nella sua totalità, ma non lo fa per una mia precisa scelta di implementazione, dettata proprio dallo sviluppo commerciale che l'applicazione potrebbe avere. La *figura 4.2* sottostante, aiuterà a capire meglio quel che intendo.

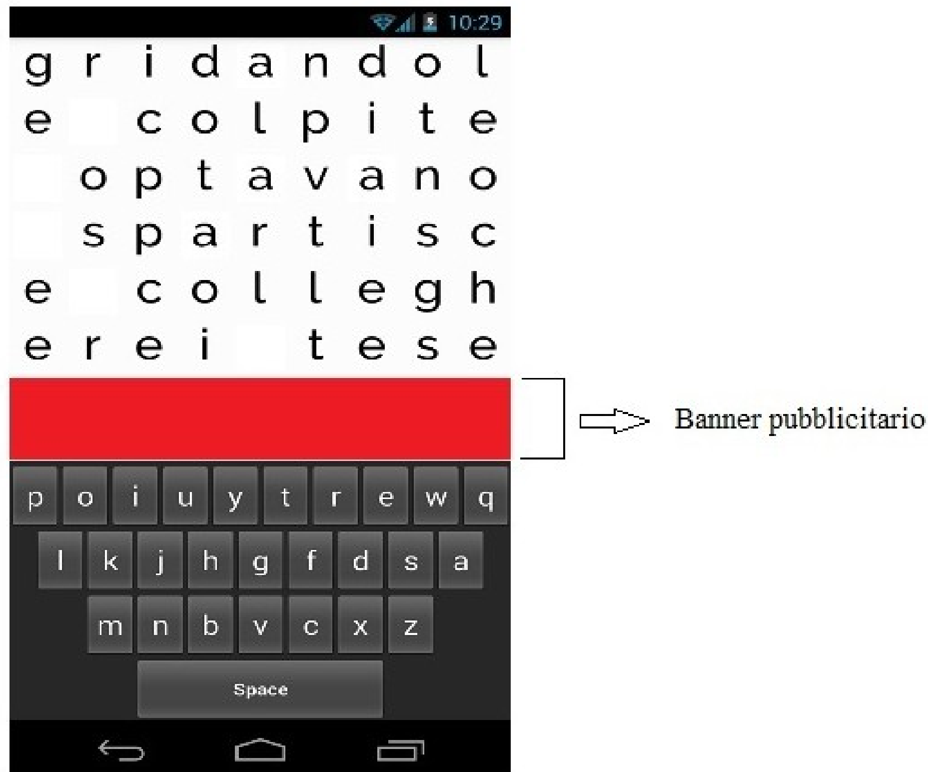


Figura 4.2 Possibile posizionamento del banner pubblicitario in NinjaTyping

Lo spazio vuoto, evidenziato da una striscia rossa tra la frase e la tastiera nella figura riportata, servirà per inserire un banner pubblicitario. Il motivo di tale scelta è dettato dalla tipologia di guadagno che si può ottenere, attraverso la partnership con Google. Infatti ad ogni *click* (o meglio *touch*) su un qualsiasi banner pubblicitario di un'applicazione, Google assegna una piccolissima parte in denaro allo sviluppatore. Quindi è compito e soprattutto interesse dello sviluppatore, inserire i banner pubblicitari in punti dell'applicazione in cui vi è più possibilità che vengano cliccati. Nello specifico in NinjaTyping, il punto che meglio risponde a questi requisiti è proprio la parte centrale

della GameActivity. L'utenza potrebbe provare fastidio nell'avere un banner pubblicitario nella schermata di gioco, quindi darò anche l'opportunità di procedere alla rimozione di tale banner, offrendo però un pagamento.
Con queste considerazioni generali riguardanti gli sviluppi commerciali di NinjaTyping, si conclude quindi il quarto e ultimo capitolo della stesura della tesi.

Conclusioni

Attraverso gli sviluppi futuri che l'applicazione NinjaTyping potrebbe avere, ho concluso il percorso di stesura della tesi. Nel corso dei vari capitoli, ho cercato di spiegare i motivi principali che mi hanno spinto a sviluppare un'applicazione per dispositivi mobili, fornendo dati concreti a supporto della mia decisione.
Ho quindi proceduto all'illustrazione della progettazione e dello sviluppo alla base di NinjaTyping, soffermandomi con più attenzione nella descrizione delle parti più caratteristiche e significative dell'applicazione.
A livello didattico, ho approfittato di questa esperienza per praticare diversi linguaggi di programmazione appresi durante il mio cammino universitario, oltre che a varie metodologie per lo sviluppo e l'ingegnerizzazione del software.
Ho cercato inoltre di esplorare nuovi tipi di implementazioni, andando a sviluppare tipologie di comportamento del software per me nuove, come ad esempio il meccanismo che permette a due utenti di giocare partire online.

Per concludere, mi preme precisare che l'applicazione è stata sviluppata non semplicemente come un progetto per una tesi di laurea da abbandonare a se stessa, una volta conseguito l'obiettivo per così dire “burocratico”, ma come un progetto da coltivare nel tempo, che possa ottenere risultati concreti e che possa aiutarmi nello sviluppo di progetti futuri simili.
Per questo motivo ho cercato quindi di sviluppare NinjaTyping costruendo una solida architettura di base, in modo da rendere possibile l'inserimento di nuovi tasselli nel tempo, atti a rendere il progetto sempre più solido e concreto.
Sarà inoltre molto utile e interessante, inserire la mia applicazione all'interno di un mercato globale in continua espansione. In questo modo, conto di riuscire a studiare e ad apprendere le varie logiche commerciali, che stanno alla base di un ambiente che con tutta probabilità farà parte del mio futuro professionale.

Bibliografia

Massimo Carli, "*Android 4. Guida per lo sviluppatore*", APOGEO

Paul J. Deitel, Harvey M. Deitel, "*How to program*", Daitel & Associates Inc.

Gatner Inc, "*Gartner Says Worldwide PC, Tablet and Mobile Phone Combined Shipments to Reach 2.4 Billion Units in 2013*", "http://www.gartner.com/newsroom/id/2408515", 2013

Androidauthority, "*Google: 900 million Android activations, 48 billion app installs*", "http://www.androidauthority.com/google-io-android-activations-210036/", 2013

Idownloadblog, "*Apple starts countdown to 50B app downloads*","http://www.idownloadblog.com/2013/05/02/apple-50b-app-downloads-countdown/", 2013

Google Inc., "*Data about the relative number of devices running a given version of the Android platform* ", "http://developer.android.com/about/dashboards/index.html ", 2014

Techcrunch, "*Despite Google's Gains, iPhone Still Edges Out Android Devices In App And Overall Smartphone Usage, Says Forrester*", "http://techcrunch.com/2013/08/01/despite-googles-gains-iphone-still-edges-out-android-devices-in-app-and-overall-smartphone-usage-says-forrester/", 2013

Comscore, "*Smartphone OEM Market Share*" , "http://www.comscore.com/Insights/Press_Releases/2013/4/comScore_Reports_February_2013_US_Smartphone_Subscriber_Market_Share", 2013

Appleinsider, “*Apps no longer differentiator in iOS vs. Android war; services next battleground*”,”http://appleinsider.com/articles/14/01/06/apps-no-longer-differentiator-in-ios-vs-android-war-services-next-battleground”, 2013

Google Inc, “*Google Cloud Messaging* (GCM)”, ”https://support.google.com/googleplay/android-developer/answer/2663268?hl=it”, 2014

RINGRAZIAMENTI

Una tesi di laurea senza ringraziamenti, sarebbe "*come un corpo senza anima",* soprattutto se il cammino universitario è stato intenso e difficile, come per il sottoscritto. Mi sento in dovere quindi di ringraziare chi mi ha aiutato nel conseguimento di questa laurea.
Innanzitutto grazie al mio relatore, il Dott. Mirko Ravaioli, uomo di grande competenza e professionalità, sempre disponibile al confronto e stimolante nello sviluppo del progetto NinjaTyping grazie a preziosi consigli.

Il ringraziamento più grande è per la mia famiglia. Grazie mamma e papà per aver permesso a vostro figlio di conseguire questo importante traguardo, senza il vostro supporto morale ancor prima che economico, non ce l'avrei sicuramente fatta. Ve ne sarò per sempre grato, spero di ripagare i vostri sforzi regalandovi altre soddisfazioni. Grazie a mia sorella, per avermi supportato durante questi anni e grazie ancor di più per averlo fatto nell'ultimo periodo, un periodo in cui hai trovato la forza per spronarmi nonostante un momento personale complicato, in cui avrei dovuto essere io a darti conforto e non viceversa. Grazie anche a suo marito per quanto fatto per lei e per il conforto nei miei confronti. Più che un cognato, sei ormai un fratello acquisito.

Grazie agli amici di una vita. Grazie Andre, Troglo, Vale e tutti gli altri. Grazie per il vostro supporto e la vostra sincera amicizia, anche in momenti in cui nel corso di questi anni, non sono stato molto presente. Grazie anche ad Ale, un amico direi ritrovato. Grazie soprattutto per quest'ultimo periodo in cui senza i nostri allenamenti sarei probabilmente impazzito!

Grazie alla mia terra e ai parenti lontani che vi abitano. Grazie per la vostra tipica allegria, spensieratezza e positività. Anche se spesso non l'ho dimostrato, sono sinceramente orgoglioso delle mie origini e di essere "uno di voi".

Last but not least, vorrei ringraziare tutti i colleghi che ho conosciuto in questi anni universitari. In particolare ringrazio "*il Canducci*", "*Maccasa*" e *Giorgio*. Abbiamo cominciato, quasi tutti insieme. Abbiamo sofferto insieme, ci siamo aiutati a vicenda, ci siamo spronati con una sincerità, oserei quasi dire *pericolosa.* Voi più di tutti, sapete quanto ho sofferto, quanto ho lavorato, ma anche quanto mi sono divertito in questi

anni. Siamo professionalmente e umanamente cresciuti insieme. Eravamo ragazzini spaesati, ora siamo uomini con idee chiare e progetti importanti. Sicuramente, per qualche tempo ancora, procederemo il nostro "*viaggio*" insieme, per conseguire ambiziosi obiettivi. Se riuscissimo a realizzare la metà delle idee che abbiamo avuto in questi anni, sarebbe un successo, ma ormai mi conoscete, sono un inguaribile sognatore, quindi non mi pongo limiti e vi prometto che ci riusciremo. Perché come diceva Michael Jordan, "*I limiti, come le paure, sono spesso soltanto illusioni*".

Grazie a tutti,
Michelangelo.