Alma Mater Studiorum - Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

APPLICAZIONE SMARTPHONE GEO-SOCIAL GAME

TESI DI LAUREA IN MOBILE WEB DESIGN

Relatore:
Dott. Mirko Ravaioli

Presentata da:
Davide Gianessi

Sessione II
Anno Accademico 2011-2012

SOMMARIO

1 – IL MOBILE

Le più grandi innovazioni tecnologiche nella storia sono quelle che entrano a far parte stabilmente della quotidianità delle persone, si fondono con le pratiche e i comportamenti e giungono a essere utilizzate in modo automatico e quasi inconscio.

È stato questo il caso della televisione che, se alla sua introduzione rappresentava quasi un bene di lusso, oggi non solo gode nel nostro paese di una penetrazione pressoché totale, ma è divenuta quasi un oggetto imprescindibile nell'ambiente domestico. Un processo simile ha interessato i dispositivi mobili, e in particolare il cellulare. Se fino a pochi anni fa, per fare un esempio, rappresentava la normalità usare apparecchi pubblici quando ci si trovava fuori casa, oggi oggetti come gettoni e tessere telefoniche appaiono obsoleti e accade frequentemente che molte telefonate vengano effettuate dal telefono cellulare anche tra le mura domestiche. Allo stesso modo, molte azioni comuni vengono sempre più spesso eseguite tramite il cellulare: è diventato la nostra sveglia, la nostra agenda degli appuntamenti, il nostro taccuino per gli appunti, il nostro registratore vocale e così via. Un'evoluzione simile non è sintomo soltanto dell'affermarsi di applicazioni tecnologiche, ma di una mutata mentalità e di un nuovo bisogno di libertà nella fruizione dei media.

Estendendo lo sguardo agli altri dispositivi mobili, è possibile notare come il concetto di mobilità sia estremamente pervasivo anche in contesti di tipo domestico. L'uomo contemporaneo mostra un sempre più spiccato desiderio di libertà dalla "schiavitù" dei "fili" e di essere wireless sia dal punto di vista della connessione, sia dell'alimentazione elettrica. È anche per questo motivo che, pur in un contesto domestico, è sempre più frequente che la musica venga

fruita dal lettore musicale piuttosto che dallo stereo; che i libri vengano letti tramite gli e-book, che la tv venga fruita dai tablet, che i videogiochi siano vissuti con nuovi strumenti come le console portatili o le nuove console con comandi senza fili e così via.

1.1 - LA MOBILITA'

Il termine *mobile* viene oggi associato a una molteplicità di concetti e utilizzato in svariati ambiti, quasi in qualità di prefisso per definire altro.

Nell'era postmoderna, espressione con cui viene denominata l'epoca storica attuale, uno dei mutamenti più pregnanti e forse più noti riguarda la sfera delle relazioni sociali. Se nell'era moderna la sfera sociale era stata caratterizzata da un sostanziale disfacimento del tessuto connettivo delle relazioni sociali e da un rifiuto delle strutture gerarchiche, come ad esempio in ambito familiare e religioso, l'era postmoderna si differenzia invece per un "rinnovato bisogno di socialità".

I legami sociali si ricompongono però secondo schemi nuovi, in aggregazioni come ad esempio *brand community* e *tribù*, aggregazioni spontanee caratterizzate rispettivamente da una passione comune per un brand o dalla condivisione di un semplice interesse; strutture sociali flessibili e non permanenti in cui è agevole entrare o uscire. Già da questi elementi è possibile ravvisare la presenza del bisogno di mobilità: l'uomo postmoderno, pur sentendo il bisogno di costruire rapporti sociali basati su un certo tipo di interscambio e condivisione, avverte parallelamente il bisogno di libertà di movimento, di abbandonare un'aggregazione sociale per abbracciarne un'altra, di muoversi senza vincoli all'interno della rete delle proprie relazioni sociali.

A un livello macro, la *mobilità fisica* si traduce nei fenomeni migratori, nel turismo a lunga tratta, nel movimento su scala globale tanto di persone quanto di merci, denaro, informazioni. Tali fenomeni hanno avuto una notevole crescita negli ultimi decenni, complice la globalizzazione, che ha fornito presupposti e condizioni importanti per questa evoluzione. Tale tendenza si traduce, a livello micro, in un aumento degli spostamenti su scala locale. Il progressivo distanziamento tra il luogo di lavoro e l'abitazione (sempre più polarizzati, rispettivamente, nelle aree urbane e in quelle periferiche) ha contribuito a intensificare gli spostamenti quotidiani.

Il fenomeno che si è invece descritto come *mobilità mentale* porta con sé altri tipi di implicazioni. Innanzitutto, sul piano funzionale, si verifica una moltiplicazione delle "destinazioni d'uso" di tempi e luoghi, creando in realtà una stratificazione di contesti. In particolare, l'accresciuta mobilità ha provocato un aumento dei cosiddetti "tempi interstiziali" e dell'importanza di tempi e luoghi transitori. Per fare un esempio, il treno diviene un luogo in cui è possibile sviluppare relazioni (magari telefonando o attraverso la connessione Wi-Fi, disponibile ormai su molti treni ad alta velocità), lavorare, studiare, intrattenersi sui social network e via dicendo. Questa prima connotazione della mobilità mentale ha effetti anche su un altro piano, quello delle relazioni sociali, che sono sempre più mobili, flessibili e caratterizzate a seconda dello specifico contesto. In particolare, agli incontri fisici realizzati anche grazie ai numerosi spostamenti, si interpongono quelli virtuali, abilitati dalla Rete e dai suoi strumenti.

In risposta all'evoluzione verso il modello dell'*homo mobilis*, si è assistito alla nascita e alla diffusione di una pletora di oggetti definiti "nomadi", con il preciso obiettivo di fornire all'individuo strumenti per permettergli di vivere la sua quotidianità completamente "wire-less". È così che nascono quindi i lettori musicali portatili, le console di videogiochi portatili, i notebook, i navigatori satellitari, i tablet e, tra tutti, il simbolo della mobilità: il telefono

cellulare. Tutti questi dispositivi si caratterizzano, seppur ciascuno in misura differente, per alcuni aspetti che contraddistinguono il loro utilizzo da parte degli individui e che contribuiscono a sviluppare un rapporto particolare con questi device. Le peculiarità dei mobile device sono:

- *Personalizzazione*: i device mobili si caratterizzano per essere estremamente personalizzabili e adattabili, tanto dal punto di vista fisico/estetico quanto da quello dei contenuti e delle funzionalità. Per quanto riguarda il primo aspetto, si pensi all'ampia varietà dei dispositivi e al numero di varianti (es. i lettori Mp3 sono disponibili in diversi colori, così come molte console di video-giochi sono differenziate per un pubblico maschile o femminile). Per molti dispositivi, inoltre, si è diffusa la possibilità di acquistare cover intercambiabili. Una crescente personalizzazione si è resa disponibile anche sul piano delle funzionalità. Per restare nell'esempio dei telefoni cellulari, non solo è possibile personalizzarne toni e sfondi, ma anche la disposizione delle icone del menu. Questa tendenza è stata resa ancora più incalzante dall'introduzione degli Application Store che hanno, in tal senso, estremizzato la componente di personalizzazione del menu attraverso l'installazione di applicativi secondo i gusti e le necessità dell'utente.
- *Interazione*: la risposta nei confronti degli stimoli provenienti dai device mobili si sviluppa sostanzialmente in tempo reale e in modo fortemente contestualizzato. La diffusione di schermi touch, non solo per quanto riguarda i cellulari, ma anche per i lettori musicali, i tablet e via dicendo, ha via via ridotto le barriere tra l'utente e il device, contribuendo a un progressivo miglioramento della user experience e a un'evoluzione sempre più marcata dei mobile device quali estensioni

digitali del corpo umano proprio in quanto rispondenti a stimoli tattili, che non interrompono la sensorialità propria dell'individuo.

- *Immediatezza*: grazie alle loro caratteristiche di portabilità, i mobile device sono sempre a disposizione dell'utente e consentono una fruizione immediata dei contenuti non appena si verifichi lo stato di necessità. Questa caratteristica contribuisce a rendere maggiormente contestuale il tipo di fruizione del mezzo: l'immediatezza dell'utilizzo consente un fortissimo legame con il luogo e il tempo dell'interazione, sviluppando quindi le condizioni necessarie per un tipo di interazione in real time e localizzata.
- *Localizzazione*: come anticipato, le possibilità offerte dai mobile device in termini di localizzazione consentono all'utente un utilizzo legato al contesto geografico in cui si trova. Ciò rappresenta un presupposto per l'integrazione della user experience con stimoli provenienti dall'esterno e secondo la tipologia e qualità delle interazioni che l'utente intrattiene in determinati luoghi. La localizzazione può essere abilitata da tecnologie specifiche (come il GPS nel caso dei telefoni cellulari o dei navigatori satellitari), oppure comunicata dall'utente stesso, in seguito a una sua interazione nello specifico contesto, e porta con sé un importante bagaglio di informazioni sul contesto di fruizione e di consumo specifici.

1.2 - NEW ECONOMY

Ormai tutti erano consapevoli di essere definitivamente entrati nella *pc economy*: tutte le nostre azioni avrebbero dovuto essere regolate dal personal computer (fisso o mobile). Nel volgere di pochi anni, siamo invece stati investiti da un cambiamento inatteso, quantomeno nella sua portata: la diffusione esplosiva degli smartphone, i cosiddetti telefonini evoluti. Una

recente ricerca ha evidenziato come nel 2013 lo stock di smartphone disponibili su scala globale (stimato in 1,8 miliardi) supererà quello dei personal computer (stimato in 1,72 miliardi).

Come spiegare questo trend? Semplice: il telefono cellulare ha progressivamente modificato la propria funzione d'uso. Da strumento di ascolto, si è progressivamente trasformato in un oggetto che viene guardato (secondo alcune ricerche per l'80% del tempo, e solo per il 20% utilizzato per effettuare chiamate), fino a diventare negli ultimi anni un dispositivo a supporto dei processi di acquisto delle persone.

Si ritiene infatti che il telefono cellulare, nelle sue forme più evolute, sia ormai diventato un *killer device* che accompagna l'individuo in ogni momento della sua vita, garantendo piena connettività in movimento: quando si è alla fermata di un mezzo pubblico per consultare notizie o confrontarsi con amici, o mentre si guarda televisione (l'86% dei navigatori di mobile internet americani ha dichiarato di farlo mentre guarda la tv). Molteplici sono i cambiamenti in atto nel contesto, che rendono questa affermazione una prospettiva quanto mai probabile. Tra gli altri:

- il *bombardamento informativo* a cui ciascuno di noi è quotidianamente esposto: il numero di messaggi veicolati nei differenti canali, tradizionali e digitali, è ormai tale da generare una vera e propria entropia comunicativa, così da rendere sempre più difficile raggiungere l'obiettivo di generare interesse rispetto a uno specifico messaggio da parte del ricevente;
- la continua crescita del *tempo trascorso in mobilità*: si tratta di un processo particolarmente rilevante nelle grandi aree metropolitane del nostro paese; la logica conseguenza è che i ritmi della vita risultano in questi casi tanto alterati da indurre nelle persone coinvolte dal fenomeno una percezione negativa circa la qualità del tempo trascorso in mobilità;

- la sempre maggiore propensione delle persone a operare in una prospettiva di tipo multitasking: in particolare, la progressiva evoluzione delle potenzialità delle tecnologie digitali (su qualsiasi device) e la loro diffusione fa sì che siano sempre più comuni i processi di fruizione simultanea di informazioni e contenuti su più canali;
- la piena affermazione di fenomeni di *convergenza multimediale*: molti degli strumenti con cui ciascuno di noi interagisce – televisione, telefono cellulare, personal computer, console di videogiochi ecc. – presentano ormai molteplici funzioni d'uso e, per questo motivo, sono in grado di veicolare tipologie di contenuti molto diverse. Se in passato, ad esempio, i contenuti video erano patrimonio esclusivo della televisione, oggi sono invece fruibili attraverso varie piattaforme: nella sostanza, aumenta la flessibilità d'uso di ciascuno degli strumenti di comunicazione con cui quotidianamente l'individuo si interfaccia.

1.3 - DEVICE : TABLET E e-BOOK

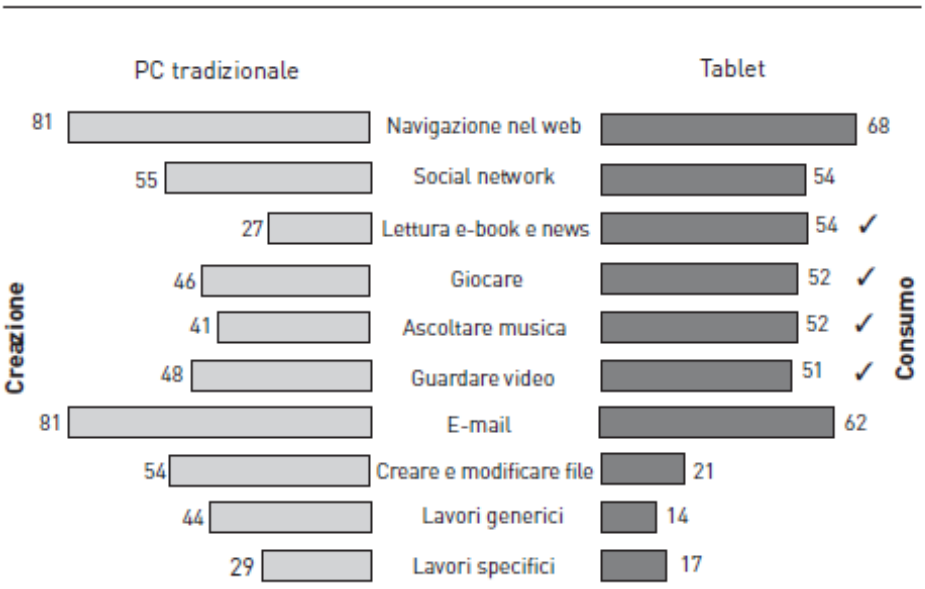
Due mobile device di recente introduzione, ma che hanno prodotto notevolissimi effetti sul mercato in termini di vendite e di paradigmi di consumo, sono i tablet pc e gli e-book reader. I tablet pc, definiti per la prima volta da Microsoft nel 2000, hanno conquistato i consumatori di tutto il mondo grazie all'iPad di Apple.

I tablet pc sono sostanzialmente assimilabili a computer portatili, ma con alcune differenze: in particolare, hanno maggiore capacità di input (es. schermo touch o penna) e maggiore portabilità (dimensioni ridotte che permettono di tenere in mano lo strumento).

Gli e-book reader sono, come il nome stesso suggerisce, lettori di libri elettronici. Si tratta di dispositivi dotati di un'unità di memoria allocata a contenere libri in formato digitale e di uno schermo per la lettura,

ottimizzato per non stancare la vista. Nonostante il processo di avvicinamento al concetto di libreria digitale sia fatto risalire da alcuni addirittura agli anni Settanta, l'anno chiave per gli e-book reader è stato il 2007, anno dell'introduzione da parte di Amazon di Kindle, il primo e-book reader che ha conquistato in maniera consistente il mercato. In seguito le due tipologie di device, tablet pc ed e-book reader, si sono progressivamente avvicinate: i primi hanno assorbito in modo sempre più massiccio le funzionalità dei secondi che, viceversa, hanno assunto caratteristiche tipiche dei tablet (es. lo schermo touch).

La linea di demarcazione tra tablet e pc si sta riducendo sempre di più, tanto da far sovrapporre molto spesso attività che prima venivano svolte prettamente sul pc. In questo panorama, quindi, i confini tra le diverse tecnologie si assottigliano, ma il consumatore ha modalità di utilizzo differenti a seconda che fruisca di un pc o di un tablet. Per distinguere questi due ambiti, bisogna focalizzare l'attenzione sulle differenze in termini di modalità di utilizzo del mezzo: i tablet si prestano maggiormente al consumo, mentre i pc alla creazione di contenuti. Oltre



1Confronto tra uso di tablet e pc (% di utenti che utilizzano il device regolarmente per ogni attività)

al confronto tra tablet e pc, è interessante studiare anche la relazione tra tablet e smartphone, e in particolare confrontare le attività che il consumatore svolge sull'iPad e sull'iPhone, nell'ipotesi che li possegga entrambi. L'iPad, grazie alla migliore qualità e ampiezza dello schermo, viene preferito all'iPhone per accedere a contenuti editoriali e a video. Le durate delle sessioni per iPhone sono molto più brevi rispetto a quelle dell'iPad per quasi tutte le categorie di contenuto, soprattutto per la lettura di news e riviste: le sessioni più frequenti sono quelle intermedie (dai 16 ai 30 minuti e dai 31 ai 60 minuti).

1.4 - DEVICE : TELEFONO CELLULARE E SMARTPHONE

Tra gli strumenti mobili, tuttavia, il ruolo principe è sicuramente giocato da cellulari e smartphone. Essi godono di una penetrazione elevatissima in tutto il mondo, dove esistono circa 5 miliardi di telefoni cellulari (pari al 72% della popolazione mondiale; dati BBc 2010).

La diffusione di questi dispositivi non interessa solamente i cosiddetti paesi industrializzati, ma anche quelli in rapida evoluzione come Cina e India, dove spesso sostituiscono le linee telefoniche fisse. L'Italia si dimostra tuttavia uno dei paesi a più alta penetrazione, con circa 45 milioni di device (The Nielsen Company, 2010c), di cui 15 milioni circa sono smartphone (dati comScore 2010) e circa 90 milioni siM card (tre ogni due abitanti: solo la Corea del Sud presenta numeri ancora più elevati; fonte: Agcom, 2009).

Ripercorrendo la storia del cellulare, sono molte le funzionalità la cui disponibilità si è avvicendata sui terminali: molte si sono consolidate nel corso del tempo e hanno contribuito a plasmare l'identità e le modalità d'uso da parte degli utenti di tali strumenti (si pensi ai messaggi di testo, che oggi

costituiscono una funzionalità irrinunciabile di cellulari e smartphone); altre funzionalità, invece, hanno fatto solo una breve apparizione, per essere successivamente abbandonate o rimpiazzate da tecnologie più evolute.

Il primo telefono commerciale fu lanciato sul mercato nel 1983: era il Motorola Dyna Tac 8000X; pesava otto etti, costava circa 4.000 dollari, disponeva di un piccolo display e permetteva esclusivamente di *inviare e ricevere chiamate*, ma non aveva ancora alcuna rubrica. Se si pensa che da allora è trascorso poco più di un quarto di secolo, è stupefacente la rapidità dell'evoluzione che ha caratterizzato tali dispositivi e il numero e la complessità delle funzionalità che oggi supportano.

Identificando le tappe più importanti che hanno portato alla nascita dei telefoni cellulari e smartphone odierni, bisogna ricordare, nel 1989, la famiglia Micro Tac realizzata da Motorola, nella quale spiccava l'8900 GSM, dotato di una memoria interna che permetteva di archiviare fino a 100 numeri telefonici, costituendo così la prima forma di rubrica.

Questi telefoni disponevano inoltre della possibilità di essere collegati a un computer attraverso cui si poteva gestire una rubrica condivisa. In seguito, con l'adozione nel 1991 del GSM come standard europeo e di sistemi di codifica digitale dei dati, fu possibile nel 1992 introdurre l'SMS (*short message service*), concepito inizialmente come sistema per le comunicazioni di servizio degli operatori di telefonia mobile, ma che divenne di uso comune anche tra gli utenti.

Un ulteriore passaggio di grande importanza fu la tecnologia WAP (Wireless Application Protocol), che ha permesso una prima integrazione dalla rete cellulare con internet. Presentata per la prima volta nel 1993 da Apple sul proprio PDA (*personal digital assistant*)

Message PAD, nel 1999 fu disponibile anche in Italia grazie all'operatore telefonico Omnitel. In quegli anni il ritmo dell'evoluzione si fece più serrato. Nel 1997 fu introdotta la rete GsM 1800, più comunemente nota come Dual

Band, che operava con frequenze più elevate e favoriva il funzionamento dei terminali anche in luoghi chiusi. Parallelamente furono sviluppate nuove funzionalità di memoria e di servizio all'interno dei terminali, come quelle introdotte in particolare da Motorola[8] e Nokia: *registro chiamate*, *calendario* e *calcolatrice*.

Con la rete GPRS e il conseguente forte impulso dato alla trasmissione dei dati, vennero introdotti gli MMS (*multimedia messaging service*), messaggi contenenti oggetti multimediali quali immagini, video e file audio.

Il GPRS diede vita a un importante cambiamento nelle modalità di fruizione del mezzo, in quanto il costo per l'utente passò dall'essere calcolato in base alla durata della connessione alla rete internet (necessaria per la trasmissione dei dati) alla quantità di dati scaricati. I cellulari, inoltre, si trasformarono sempre più in versatili lettori multimediali, integrando in primo luogo le funzionalità di *lettore musicale* (in particolare lettore Mp3) e *radio*.

Nel 1997 Nokia, con il modello 9000i, introdusse un nuovo concetto di cellulare, in qualche modo più vicino al pc: si tratta, infatti, del primo cellulare dotato di un *display con orientamento orizzontale* e di una *tastiera* QwERTY in grado, oltre alle funzionalità finora descritte, di inviare e ricevere *fax* e di supportare varie funzioni simili a quelle di un PDA, quali l'*agenda elettronica*, il *bloc notes* e la possibilità di inviare e ricevere *e-mail*. Nello stesso anno Nokia portò un'altra storica innovazione, dotando i propri dispositivi di Snake, il primo *videogioco* su telefoni cellulari.

Risale a due anni più tardi, nel 1999, l'introduzione di un'altra importantissima tecnologia, il Bluetooth, le cui applicazioni sono tutt'oggi molteplici, come ad esempio gli auricolari Wireless. WAP e GPRS congiuntamente hanno aperto la strada alla tecnologia UMTS e ai cellulari di terza generazione, che rendono possibile l'interazione con il *roaming* internazionale, i servizi sia in banda larga sia stretta con una qualità costante, lo streaming video e altro ancora.

Il 2001 fu un anno molto importante, in quanto si affacciarono nella telefonia mobile due importanti elementi che ancora oggi caratterizzano i cellulari in commercio: lo *schermo a colori*, realizzato dalla neonata joint venture Sony Ericsson con il suo primo cellulare, il T68; la prima fotocamera digitale, introdotta dalla Sharp nel suo J-SH04. Nello stesso anno fu compiuto un passo importante anche per quanto riguarda la tecnologia GPS: sebbene fossero disponibili già dalla fine degli anni Novanta, nel 2001 in Giappone vennero forniti i primi LBS (*location-based service*) e venne commercializzato il primo cellulare dotato di GPS (rispettivamente, in luglio da parte di DoCoMo e in dicembre da parte di xDDi). Nel 2003 venne realizzato da Nokia il primo telefono cellulare fortemente ibridato con una console portatile, il Nokia N-Gage.

Con l'avvento degli smartphone, il telefono cellulare tradizionale assume prevalentemente la connotazione di un prodotto di fascia più bassa. Questo ha provocato notevoli cambiamenti nel panorama dei produttori: nuove aziende hanno consolidato la propria presenza sul mercato, mentre altri attori storici (come Nokia) hanno visto mutare e in alcuni momenti ridimensionarsi il proprio ruolo. Il cambiamento che ha interessato il mercato non è solamente di natura tecnologica ma investe anche i modelli di business che sottendono le attività. La predisposizione, infatti, degli smartphone all'installazione di componenti software aggiuntive e di applicativi ha dato il via alla nascita di piattaforme sul modello di iTunes che hanno completamente rivoluzionato le dinamiche all'interno del sistema dell'offerta e l'esperienza di fruizione degli utenti.

A valle della panoramica riguardante le funzionalità dei cellulari, è importante sottolineare il ruolo che questa tipologia di device è giunta a ricoprire nella vita degli individui. A partire da un utilizzo esclusivamente professionale dei primi cellulari, infatti, si è passati a una crescente diffusione presso tutte le fasce d'età, mentre gli utilizzi hanno continuato a moltiplicarsi.

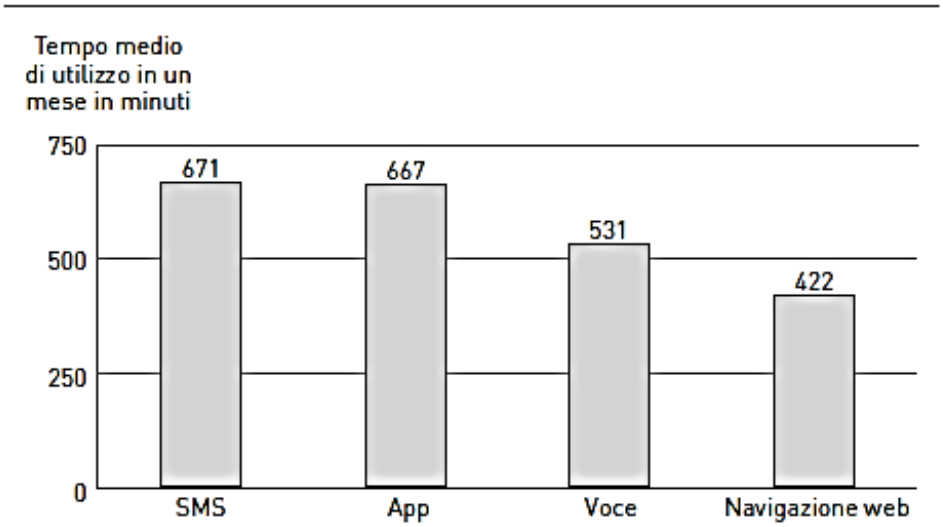
Benché sia ancora estremamente diffuso un uso per fini lavorativi, il cellulare è divenuto anche uno dei principali mezzi di comunicazione personale, sia all'interno delle famiglie (dove persino i giovanissimi ne possiedono uno), sia a livello trasversale in ogni genere di rapporto sociale. Parallelamente, tra le funzionalità hanno assunto sempre più importanza, accanto alle telefonate, l'invio di messaggi e il traffico dati (tanto la navigazione su mobile internet quanto l'utilizzo di applicativi per smartphone che richiedano un accesso al web, come per i social network).

Come conseguenza, l'utilizzo del device ha subito notevoli evoluzioni ed è progressivamente diminuita la quota di traffico imputabile alle telefonate rispetto al traffico dati. Nel 2010, ad esempio, pur a fronte di un aumento del traffico voce, il traffico dati ha avuto ritmi molto più elevati . Tale crescita straordinaria è abilitata da terminali e reti sempre più potenti ed efficienti, che permettono la fruizione di una gamma in continua crescita di contenuti in mobilità, trasformando tali device in quella che viene spesso definita come l'*estensione digitale* degli individui, quasi un arto aggiuntivo che consente loro di svolgere quasi ogni tipo di attività e che per molti rappresenta una sorta di porta sulle relazioni sociali, conquistandosi così anche l'appellativo di *social medium*.

A conferma di questo cambiamento nel modo di utilizzare il telefono cellulare, si evidenzia come, con riferimento ai mercati americano e britannico – spesso paesi anticipatori di fenomeni di adozione –, gli utenti trascorrano lo stesso tempo a effettuare il download e a usare software che scrivere e leggere SMS: il tempo dedicato dagli utenti alle applicazioni (pari a 667 minuti al mese) ha eguagliato quello riservato agli SMS (671 minuti al mese). L'impiego delle applicazioni ha inoltre superato nettamente, in termini di tempo speso, l'uso delle chiamate vocali (531 minuti al mese) e ancora di più del web browsing (422 minuti al mese).
Si può quindi affermare che il nuovo paradigma del telefono cellulare si basi su due elementi cardine: l'ambito del mobile internet e le applicazioni, denominate anche App.

1.5 - SISTEMI OPERATIVI MOBILE

In questo contesto i principali costruttori di cellulari hanno messo a disposizione degli sviluppatori i propri sistemi operativi, ciascuno con il proprio ambiente di sviluppo, i propri tools e il proprio linguaggio di



2Tempo di utilizzo di smartphone per categorie di attività

programmazione. Nessuno di questi, però, si è affermato come standard. Per esempio, per realizzare un'applicazione nativa (non web) per iPhone è necessario disporre del sistema operativo Mac OS X, su cui si basa l'iPhone, e conoscere il linguaggio Object-ive–C (un'estensione del linguaggio C, con caratteristiche Object Oriented) giunto ora alla versione 2.0. Per lo sviluppo di un'applicazione per un dispositivo Nokia, basata sul sistema operativo Symbian, è necessario utilizzare come linguaggio un dialetto del C++. Dopo l'accordo con Microsoft, sviluppare per Nokia significherà anche utilizzare gli strumenti per la creazione di applicazioni per Windows Phone 7, che avrà sicuramente un ruolo importante nei prossimi anni. Non possiamo ovviamente trascurare la piattaforma J2ME (Java 2 Mobile Edition) anche alla luce del suo utilizzo per lo sviluppo di applicazioni per Blackberry (RIM). Oltre questi, vi sono vari sistemi operativi proprietari, la cui conoscenza è spesso limitata ai soli vendor.

Ciò che si vuole sottolineare è, comunque, la varietà di ambienti e tecnologie che uno sviluppatore deve conoscere per poter realizzare un'applicazione per un particolare dispositivo mobile. A seconda del tipo di sistema operativo, è necessario acquisire la conoscenza di un ambiente, di una piattaforma e di un linguaggio. Esiste quindi la necessità di una standardizzazione, verso cui si sono diretti Google e la Open Handset Alliance con la creazione di Android.

1.5.1 - Apple iOS

Fino alla nascita dell'iPad, si è parlato di iPhone OS, ma adesso il sistema operativo della mela è denominato semplicemente iOS. Giunto alla versione 6, ha molti punti di forza: la sua stretta integrazione con il dispositivo su cui gira, secondo la filosofia monomarca di Apple; la straordinaria cura dell'interfaccia grafica divenuta proverbiale; la disponibilità di un numero

spropositato di applicazioni sull'App Store. La qualità e la perfetta integrazione iPhone+iOS si fanno decisamente pagare in termini economici (come accade sempre con Apple) ma l'esperienza utente finale vale decisamente quel prezzo.

1.5.2 - Android

Sono tanti i punti di forza di questo sistema operativo: Android pone ai produttori di hardware alcuni paletti ma lascia una certa libertà di personalizzazione che i vari produttori di telefoni possono sfruttare per perfezionare il rapporto tra OS e smartphone su cui viene montato. Se a questa libertà di personalizzazione ci aggiungiamo anche il fatto che il sistema operativo non impone royalties ai fabbricanti, facendo scendere il prezzo totale a cui vengono venduti i dispositivi, si capisce perche' Android piaccia tanto ai produttori.

Paradossalmente però, proprio questa grande libertà concessa dal sistema potrebbe rivelarsi commercialmente un boomerang per Android: anzitutto, la personalizzazione spinta effettuata da alcuni produttori rischia di far apparire il sistema molto "incostante" nelle sue diverse incarnazioni (una sorta di "interfacce dialettali" a seconda che il produttore sia Samsung, LG, HTC etc.) ed oltretutto, questa personalizzazione, rallenta il rilascio delle nuove versioni che devono, di volta in volta, essere riadattate dai produttori, prima di essere rese disponibili.

1.5.3 - Symbian OS

Come per Apple e per BlackBerry, anche il sistema operativo Symbian è oramai considerabile "monomarca", essendo disponibile, almeno nella ultima versione, solo sui telefoni Nokia.

Symbian è uno dei più antichi mobile OS ancora vivi ed è stato protagonista di una lunga serie di vicende in cui è stato inizialmente appoggiato da diversi produttori, prima di arrivare all'attuale soluzione in cui è diventato sostanzialmente il sistema di Nokia.

L'ultima versione segna una serie di sensibili miglioramenti nell'interfaccia e nelle funzioni, tali da garantire a Symbian un prolungamento dell'esistenza nell'immediato futuro.

1.5.4 - RIM BlackBerry

I dispositivi BlackBerry di Research In Motion mantengono un ruolo di tutto rispetto e anche una certa fetta di mercato conquistata negli ultimi anni specie presso l'utenza business.

La robustezza e l'essenzialità dei dispositivi RIM, la loro pertinenza a un mondo "enterprise", l'aspetto stesso dei Blackberry con la tastiera fisica, ne danno la percezione come di un vero e proprio strumento di lavoro più che di un iconico gadget tecnologico, o di uno strumento di intrattenimento. Queste caratteristiche, unite alla proverbiale sicurezza, alla gestione della posta e della messaggistica (e aiutate forse anche all'involontario endorsement fornito dal presidente Obama) hanno decretato il successo dei dispositivi BlackBerry.

È probabile che, almeno nel mercato nordamericano, BlackBerry continuerà ad avere un suo ruolo ben definito.

1.5.5 - Windows Phone 7

Seppure i risultati delle versioni precedenti di Windows Mobile non siano certo stati memorabili, Microsoft si è rilanciata con la recente introduzione della versione 7 del suo sistema operativo mobile, completamente rinnovato e ribattezzato Windows Phone 7.

Con una politica molto sensata, Microsoft ha fatto delle richieste molto precise ai produttori di hardware riguardo alle specifiche hardware minime, in termini di potenza di calcolo, memoria e schermo, nonché' di presenza di alcuni tasti "standard". In questo modo, il sistema operativo ha prestazioni di tutto rispetto sui diversi dispositivi e presenta una sostanziale uniformità di comportamento anche fra telefonini di marche diverse, con cui si integra bene. A questo vanno aggiunte una interfaccia e una esperienza di navigazione molto ben fatte e decisamente usabili.

Del resto Windows Phone 7 è l'ultima occasione per Microsoft di restare significativamente sul mercato mobile: sbagliare questa release avrebbe significato la definitiva uscita di Microsoft dal panorama mobile e l'enorme impegno, anche economico, profuso in fase di progettazione e di lancio sta a dimostrarlo.

Un punto di svantaggio del sistema è il seguente: i produttori hanno realizzato telefoni sulle specifiche richieste da Windows, ma ciò non toglie che gli stessi produttori forniscano anche modelli analoghi o molto simili che ospitano Android: molti acquirenti, dovendo scegliere tra dispositivi molto simili, sostanzialmente solo sulla base del sistema operativo o del prezzo, opteranno tendenzialmente per la versione Android, probabilmente meno cara e più appetibile anche per ragioni di "tradizione".

2 – PROGETTAZIONE

2.1 – DESCRIZIONE DEL PROGETTO

Il progetto realizzato e presentato in questa tesi prende il nome di Tamagotchi.

Tamagotchi è un geo-social Game sviluppato su Android che rivisita l'omonimo gioco elettronico portatile creato nel 1996 da Aki Maita.

L'obiettivo di questa applicazione è quello di prendere spunto dalle idee di un passato non più recentissimo e di svilupparle in un ambiente moderno e tecnologico, sfruttando a pieno tutte le caratteristiche e le funzionalità che il mondo mobile mette a disposizione.

Lo scopo del gioco originale era quello di dedicarsi alle cure di un cucciolo virtuale, cercando di crescerlo e di farlo vivere il più a lungo possibile. Il giocatore, tramite tre bottoni (A, B e C), interagiva con l'animaletto elettronico facendogli svolgere diverse attività.

Lo scopo principale del Tamagotchi rimane lo stesso ma cambiano, ovviamente, le modalità:

- Grazie all'esperienza touch del dispositivo non si è più limitati dallo scarso numero di bottoni fisici.
- Le attività svolte dal cucciolo virtuale possono diventare a loro volta dei mini-giochi svolti dall'utente che sfruttano le nuove caratteristiche dei dispositivi mobili.
- Essendo un'applicazione per Android, non si è più costretti ad acquistare un ulteriore dispositivo elettronico per iniziare a prendersi cura del proprio Tamagotchi, bensì tutto potrà essere gestito comodamente dal proprio cellulare o tablet compatibile.

- Grazie alla geolocalizzazione è possibile dare una nuova esperienza di gioco all'utente: rispetto alla stragrande maggioranza delle App in commercio, infatti, questo geo-social game ha la caratteristica di permettere lo svolgimento delle attività del cucciolo solo in prossimità di luoghi di interesse (P.O.I) a queste collegate. Ad esempio sarà possibile curare il proprio Tamagotchi solo in vicinanza di una farmacia o di un ospedale.
- L'integrazione con Facebook permette di condividere con i propri amici i progressi, le azioni ed i luoghi visitati con il proprio animaletto elettronico.

  Ormai questo tipo di integrazione è fondamentale per la diffusione di un'applicazione in quanto è una funzionalità sempre molto utilizzata dagli utenti che dona molta visibilità e pubblicità al gioco ed ai luoghi visitati.

In parallelo con la continua ricerca di mobilità e socialità che caratterizzano la società post-moderna si è cercato di offrire agli utenti un gioco che garantisse a pieno questi servizi.
Tamagotchi permette di gestire una partita alla volta: sarà infatti possibile iniziare una nuova partita in caso di morte dell'animaletto o comunque in qualsiasi altro momento cancellando i dati della partita precedente.
Il cucciolo virtuale presenta tre livelli vitali da gestire e controllare con moderata frequenza: Salute, Fame, Riposo.
Durante la giornata, anche ad applicazione chiusa, questi livelli cambieranno e sarà per esempio possibile che il livello di Salute scenda al minimo, sintomo di un malore del Tamagotchi; in questo caso, il compito del giocatore sarà quello di aprire l'applicazione per prendersi cura del suo compagno virtuale.

Le attività disponibili per questo gioco sono cinque: fare il bagno, dare da mangiare, curare, fare giocare e dormire.
Ogni attività ha delle conseguenze sui 3 livelli vitali principali; giocare, ad esempio, fa aumentare la salute ma diminuire il livello di risposo, sarà quindi raccomandabile mettere a dormire il proprio animaletto dopo averlo fatto svagare.
Come già accennato in precedenza, ogni attività può essere svolta solo in prossimità di un POI adeguato e tanto minore sarà la distanza fisica da quest'ultimo, tanto più grande sarà il bonus dell'attività ad essere assegnato.
In ogni momento sarà possibile controllare all'interno del gioco lo stato dei livelli vitali e la mappa messa a disposizione da GoogleMaps con indicati i luoghi precedentemente visitati durante la partita e le attività svoltasi.
L'applicazione gestisce inoltre un servizio di notifiche che si attiva per indicare all'utente quando un livello vitale si avvicina pericolosamente allo zero, mettendo a rischio la vita dell'animaletto.
I mini-giochi messi a disposizione del giocatore per le diverse attività sono i seguenti:

- Fare il bagno: l'utente dovrà contare quante bolle appaiono e scorrono nello schermo ed indovinarne il numero esatto.
- Dare da mangiare: il giocatore dovrà scuotere il dispositivo fino a riempire una barra di caricamento entro un tempo limite.
- Curare: tramite delle gesture, l'utente dovrà riconoscere quali medicine sono adatte alla cura del suo animaletto.
- Dormire: semplicemente il Tamagotchi si addormenta per un ora e, per questo lasso di tempo, non sarà possibile svolgere altre attività.
- Fare giocare: Il Tamagotchi scomparirà e riapparirà in posizioni diverse dello schermo molto velocemente, obiettivo del giocatore

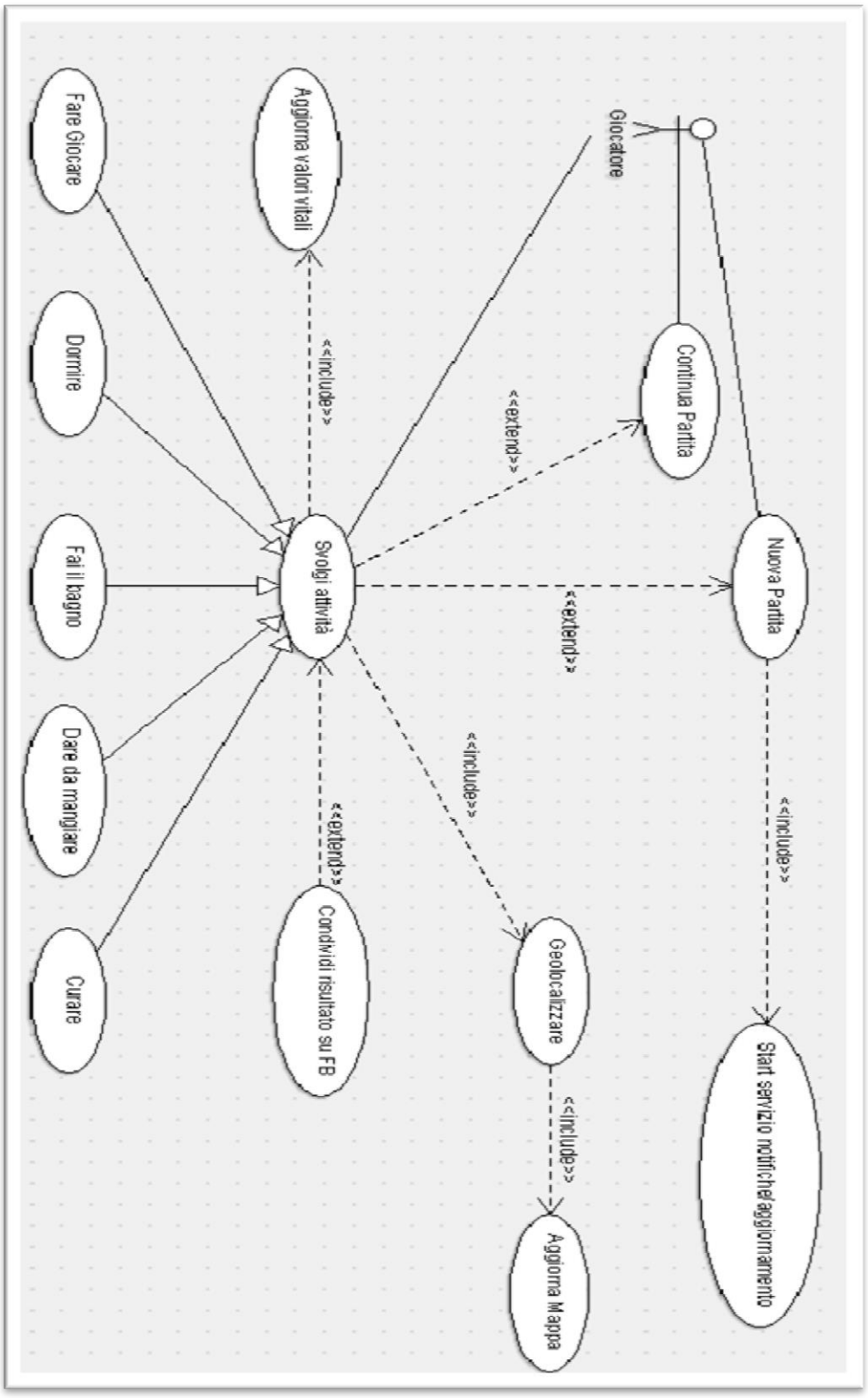
sarà quello di fare più tap possibili sul suo animaletto prima che scompaia nuovamente.

2.2 - DIAGRAMMA DEI CASI D'USO

In UML, gli Use Case Diagram (UCD o diagrammi dei casi d'uso) sono diagrammi dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. I diagrammi dei casi d'uso si possono considerare come uno strumento di rappresentazione dei requisiti funzionali di un sistema.

Nel caso di questa applicazione l'unico attore che può svolgere le diverse attività è il giocatore; come già spiegato in precedenza, il programma gestisce un utente singolo alla volta e quindi non si presentano problemi di condivisione di dati o conflitti tra diversi giocatori all'interno della stessa partita.

Il giocatore non ha bisogno di alcuna autorizzazione o azione di login per iniziare ad utilizzare l'App: appena aperto il gioco, infatti, verrà mostrato un semplice menu che farà decidere all'utente se iniziare una nuova partita o continuare quella precedente.


Giocatore
Continua Partita
Nuova Partita
Start servizio notifiche/aggiornamento
<<include>>
<<extend>>
<<extend>>
Svolgi attività
<<include>>
<<include>>
Aggiorna valori vitali
Fare Giocare
Dormire
Fai il bagno
Dare da mangiare
Curare
Condividi risultato su FB
Geolocalizzare
<<include>>
Aggiorna Mappa

In base alla scelta del giocatore si recupereranno i dati precedentemente salvati oppure si azzereranno tutti i progressi e si farà partire il servizio di notifica ed aggiornamento.
Una volta data la propria risposta, il sistema indirizzerà il giocatore verso la home; all'interno della schermata principale sarà possibile gestire e compiere praticamente tutte le azioni disponibili:

- Fare Giocare: lo scopo di questo mini-gioco è quello di riuscire a fare tap sull'immagine del Tamagotchi più volte possibili prima che questo scompaia e riappaia in un'altra posizione dello schermo.

  Ovviamente esiste un tempo limite al termine del quale l'animaletto scompare del tutto e viene chiesto al giocatore se ha intenzione di mantenere il punteggio attuale oppure ritentare la sfida. Sia nel caso della prima opzione, sia nel caso del raggiungimento del punteggio massimo prima dello scadere del tempo, saranno aggiornati i livelli vitali: in base allo score del giocatore, verrà aumentato di una certa quantità il livello di salute e decrementato quello di riposo.
- Dormire: questa attività simula una dormita dell'animaletto; per un'ora reale sarà impossibile svolgere le altre attività disponibili per il cucciolo a meno che quest'ultimo non venga svegliato con l'apposito bottone.

  Se il cucciolo riuscirà a dormire per un'ora intera senza essere disturbato sarà caricata al massimo la barra vitale del riposo.
- Curare: per aumentare il livello di salute del proprio Tamagotchi, l'utente dovrà utilizzare delle gesture a forma di cerchio o di croce per decidere se l'immagine proposta al centro dello schermo è adatta o meno alla cura dell'animaletto. Se, ad esempio, verrà raffigurata una

medicina, il giocatore dovrà cerchiare l'immagine per far capire che l'oggetto in questione è adatto allo scopo.

Verranno proposte dieci immagini ed in base al numero di scelte esatte verrà aumentata la barra vitale.

- Fare il Bagno: questo mini-gioco consiste nel testare la concentrazione del giocatore.

  Verranno fatte partire delle bolle dalla parte inferiore dello schermo ad intervalli irregolari di tempo e distanza e queste si dirigeranno verso la parte superiore intraprendendo un moto ondulatorio fino a sparire. Terminato il passaggio di tutte le bolle verrà chiesto al giocatore di indovinarne il numero esatto: in caso di risposta esatta verranno assegnati punti validi per il livello di salute altrimenti sarà necessario riprovare restando più concentrati.

- Dare da mangiare: per nutrire il proprio cucciolo si è pensato di utilizzare le funzioni dell'accelerometro.

  Questa attività consiste semplicemente nell'aumentare il livello di sazietà del proprio Tamagotchi scuotendo il proprio dispositivo al fine di riempire la barra di caricamento posta al centro dello schermo. Anche per questa attività è presente un tempo limite al termine del quale è possibile decidere se mantenere il punteggio appena raggiunto o meno.
  Il punteggio sarà ricavato dalla posizione della barra al termine del tempo massimo; se durante la partita l'utente riuscirà a riempire del tutto la barra, il cronometro si fermerà ed il punteggio assegnato sarà quello massimo disponibile.
  Per non rendere possibili riposi del giocatore durante la partita è stato deciso di decrementare automaticamente di una tacca la barra di caricamento ad intervalli regolari di due secondi, in

questo modo l'utente sarà costretto a scuotere il dispositivo per tutta la durata della sfida per evitare che la barra torni allo stato di partenza.

Per ognuna di queste attività sono preassegnati in partenza dei bonus calcolati in base alla distanza dell'utente dal POI associato; Questo perché prima di ogni mini-gioco, viene svolta in automatico un'attività di geolocalizzazione che ha come obiettivo non solo quello di calcolare bonus e punteggi massimi raggiungibili nella mini-sfida che si sta per affrontare, bensì anche quello di inserire i dati del POI in un database specifico che sarà utilizzato in seguito per gestire la mappa.
Al termine di ogni mini-gioco è inoltre possibile condividere il proprio risultato tramite un post sul proprio profilo di Facebook, così da far notare ai propri amici i propri miglioramenti o invogliarli ad utilizzare a loro volta questa applicazione.

2.3 – LOGICA PROGETTUALE

La logica progettuale dell'applicazione si basa principalmente sullo scambio di informazioni che avviene tra client e server: sia nell'ambito della geolocalizzazione per la ricerca di points of interests, sia in quello dell'integrazione con il social network per quanto riguarda le richieste fatte a Facebook.

2.3.1 – CLIENT

-Google Places

Dopo avere ottenuto le coordinate geografiche della propria posizione attuale, l'applicazione deve inviare una richiesta http verso i server di Google Places usando il metodo GET per allegare diversi parametri fondamentali:

```
https://maps.googleapis.com/maps/api/place/search/output?parameters
```

- Output: può essere scelto tra due diversi formati: JSON e XML.
- Key: API Key fornita da Google in seguito a registrazione, necessaria ad identificare l'applicazione.
- Location: latitudine e longitudine della propria posizione.
- Radius: definisce la distanza in metri entro la quale calcolare il risultato a partire dal centro di ricerche identificato dal parametro *Location.*
- Sensor: indica se la richiesta venga eseguita da un dispositivo con sensore di geolocalizzazione o meno, il suo valore può essere *true* o *false*.

-Facebook

Per potere pubblicare un post sul profilo Facebook è ovviamente necessaria l'autenticazione attraverso apposita interfaccia del social network, in questo modo si identifica l'utente e si ottiene il suo identificativo.Dopo questo primo passaggio fondamentale bisognerà preoccuparsi di inviare i parametri

necessari per la compilazione dell'oggetto che sarà visualizzato poi nella bacheca del giocatore:

- App_id: stringa alfanumerica che identifica l'applicazione, fornita da Facebook in seguito ad apposita registrazione.
- Link: indirizzo a cui viene rediretto l'utente in caso di click sul post in bacheca.
- Picture: icona che distinguerà l'oggetto facebook ed apparirà sul profilo del giocatore.
- Name: nome significativo da dare al post in via di pubblicazione.
- Caption: didascalia dell'immagine o dell'applicazione.
- Description: descrizione dettagliata del post che verrà poi visualizzata in piccolo sotto al *Name* ed al *Caption.*

Questi parametri saranno poi passati ad una richiesta per la visualizzazione di una dialog Facebook che permetta al giocatore di vedere l'anteprima del post che sta per essere pubblicato nel suo profilo e, in caso non ne sia convinto, annullare l'operazione.

2.3.2 – SERVER

-Google Places

Si è deciso di fare restituire dal server la lista di points of interest in formato JSON con i seguenti attributi:

- Status: contiene l'esito della richiesta.
- Results: contiene un array di POI e le loro descrizioni (massimo venti elementi).

- Html_attributions: contiene una serie di informazioni che devono essere inviate all'utente.
- Next_page_token: contiene un token utilizzato per ottenere le altre parti della risposta in caso l'array di POI contenga più di venti elementi


```
"html_attributions" : [
    "Listings by \u003ca
href=\"http://www.yellowpages.com.au/\"\u003eYellow
Pages\u003c/a\u003e"
  ],
  "results" : [
    {
      "formatted_address" : "529 Kent Street, Sydney NSW, Australia",
      "geometry" : {
        "location" : {
          "lat" : -33.8750460,
          "lng" : 151.2052720
        },

    ],
  "status" : "OK"
}
```


Il campo *status* può assumere diversi valori utili anche per l'azione di debug:

- OK: indica che non si sono verificati errori e la lista di luoghi è stata restituita con almeno un elemento.

- ZERO_RESULTS: indica che non si sono verificati errori di connessione ma la lista di luoghi inviata non contiene nessun elemento.
- OVER_QUERY_LIMIT: indica che si è superata la propria quota di richieste giornaliere ai server di Google Places.
- REQUEST_DENIED: indica che la richiesta è stata rifiutata probabilmente a causa di parametri non conformi alle aspettative.
- INVALID_REQUEST: la richiesta non è ritenuta valida, in genere a causa di qualche parametro mancante.

Ogni elemento dell'array di risposta conterrà un POI coerente con *location* e *radius*, ordinato tra gli altri per vicinanza e per rilevanza. I campi presenti per un oggetto di tipo Place sono svariati e spesso facoltativi ma si possono elencare i principali:

- Icon: contiene l'URL di una icona raccomandata che potrebbe essere visualizzata dall'utente al momento della visualizzazione del risultato.
- Id: contiene un identificativo univoco per il luogo in questione.
- Geometry: contiene le informazioni geografiche, in genere latitudine e longitudine.
- Name: il nome del luogo o dell'attività da fare visualizzare all'utente.
- Types[] : array che contiene informazioni sulla categoria del POI (es. "Health", "Food" ecc.).
- Formatted_address: l'indirizzo fisico del luogo o dell'attività restituita nel risultato della richiesta.

-Facebook

Per potere pubblicare un post sul profilo Facebook sono necessari due passaggi principali: La creazione dell'oggetto post e la sua successiva pubblicazione.

Dopo aver ricevuto dal client tutti i parametri necessari, l'applicazione sarà reindirizzata automaticamente dal social network verso una sua dialog dove, oltre all'anteprima del post, sarà possibile ricevere la conferma o l'annullamento da parte dell'utente. In caso positivo Facebook dovrà ricevere l'apposita richiesta di pubblicazione ed inviare all'applicazione un messaggio di conferma.

2.3.3 – DATABASE LOCALE

Una delle funzionalità di maggiore importanza di questa applicazione è senza dubbio la geolocalizzazione, motivo per cui è stato dato molto rilievo alla gestione della mappa e dello storico delle registrazioni geografiche effettuate durante la stessa partita.

Sfruttando un database locale, aggiornato ad ogni registrazione con un nuovo record, sarà possibile visualizzare all'interno della mappa un insieme di marcatori che indicano i luoghi precedentemente visitati durante lo svolgimento del gioco.

Il database conterrà due tabelle con questa struttura:

-Place:

ID	Text	PK, Id utilizzato da GooglePlace
Name	Text	Nome del luogo visitato
Latitude	Double	Latitudine del POI
Longitude	Double	Longitudine del POI

-RegistredPlace:

ID	Integer	PK, campo auto-incrementale
Place_ID	Text	FK, riferimento al luogo visitato
Date	Text	Data in cui è avvenuta la registrazione
Category	Text	Tipo di azione compiuta dal Tamagotchi per la registrazione

3 – IMPLEMENTAZIONE

3.1 - MOTIVAZIONE DELLA SCELTA DI ANDROID

Per la realizzazione di questa applicazione è stato scelto il sistema operativo Android sia per il suo notevole progresso nel mercato mobile negli ultimi anni sia per la maggiore accessibilità del suo ambiente di sviluppo.

Nel web ormai si è creata una comunità di sviluppatori di notevoli dimensioni che è spesso molto utile per risolvere alcuni inevitabili problemi durante la realizzazione del progetto.

A differenza di altri sistemi operativi, Android offre inoltre una serie di tools di sviluppo completamente open-source, utilissimi per lo sviluppo dell'applicazione.

3.2 – DATABASE LOCALE

SQLite è un leggerissimo database engine transazionale che occupa poco spazio in memoria e sul disco, pertanto è la tecnologia perfetta per creare e gestire database in un ambiente come quello degli applicativi mobile, dove le risorse sono molto limitate e dunque è molto importante ottimizzarne l'utilizzo.

A differenza della maggior parte degli altri database SQL, SQLite non ha un processo server separato ma legge e scrive direttamente su file ordinari sul disco: possiede praticamente tutti gli strumenti principali che caratterizzato i più importanti database SQL (tabelle, viste, indici, trigger)

ed il codice è distribuito gratuitamente sia per scopi commerciali che privati.

SQLite è più diffuso di quanto ci si aspetterebbe, infatti viene utilizzato in moltissimi applicativi e device che vengono utilizzati quotidianamente, come l'iPhone della Apple, Mozilla Firefox, negli smartphone Symbian, in Skype, in diversi applicativi PHP o Adobe AIR, e in molti altri progetti.

Un database SQLite è, nella pratica, un file: è possibile spostarlo, copiarlo in un altro sistema e continuerà a funzionare tutto regolarmente.

Android memorizza i file seguendo uno schema preciso; il file SQLite del database di ogni applicazione viene infatti memorizzato in una directory il cui percorso è: */data/data/packagename/databases* dove "packagename" è il nome del package del corrispodente applicativo.

Nel caso del Tamagotchi è stato necessario creare le tabelle *Place* e *RegistredPlace* per mantenere traccia dei luoghi visitati dall'utente nel corso della sua partita ed in seguito, poterli visualizzare sulla mappa. Il codice SQL per la creazione delle tabelle in questione è il seguente:

```
CREATE TABLE "main"."Place" (
   "ID" TEXT NOT NULL,
   "Name" TEXT,
   "Latitude" REAL,
   "Longitude" REAL,
   PRIMARY KEY ("ID")
);
```

```
CREATE TABLE "main"."RegistredPlace" (
   "ID" INTEGER PRIMARY KEY AUTOINCREMENT,
    "Place_ID" TEXT NOT NULL,
   "Date" TEXT,
   "Type" TEXT,
   FOREIGN KEY ("Place_ID") REFERENCES "Place"(" ID"));
```

Il codice all'interno dell'applicativo Android per inserire un nuovo record con le informazioni del luogo appena visitato assume questa forma:

```
    public boolean InsertRegistredPlace(String id,
String name,String category,java.util.Date date,double
latitude,double longitude)
    {
          String SQL;
          try{
               SQL= "INSERT INTO Place VALUES
('"+id+"','"+name+"',"+latitude+","+longitude+")";
               myDataBase.execSQL(SQL);
          }catch (SQLiteConstraintException SQLe){
               //è possibile che il luogo fosse già
presente per una precendete registrazione
          }catch (Exception e){
               return false;
          }
          try{
               SQL="INSERT INTO RegistredPlace VALUES
(null,'"+id+"','"+date.toString()+"','"+category+"')";
               myDataBase.execSQL(SQL);
          }catch (Exception e){
               return false;
          }
          return true;
    }
```

è infatti un metodo che riceve in ingresso i parametri dei record e ritorna un booleano che indica l'esito dell'inserimento nel database:

Per prima cosa si inserisce il POI nella tabella *Place*; è possibile che si generi una *SQLiteConstraintException* che indichi un Record è già presente nel database, in questo caso, sintomo che il giocatore ha già utilizzato questo luogo per una precedente registrazione, si prosegue senza problemi all'inserimento del record relativo alla registrazione e si restituisce *true* in caso di assenza di altri tipi di errori.

Più semplice è il metodo che restituisce il cursore con i dati relativi ai luoghi visitati nella partita corrente:

```
    public Cursor getRegistredPlaces()
     {
          try{
               cursorPlaces =
myDataBase.rawQuery("SELECT DISTINCT Name,
Type,Longitude, Latitude FROM RegistredPlace rp, Place
p WHERE rp.Place_ID==p.ID ", null);
          }catch(Exception e) {
               return null;
          }
          return cursorPlaces;

     }
```

3.3 – GEOLOCALIZZAZIONE

Un importante strumento messo a disposizione dai dispositivi Android è quello che va sotto il nome di *LocationManager*, con cui è possibile ricevere notifiche sulla posizione del dispositivo o lanciare un Intent qualora lo stesso entrasse in prossimità di una particolare posizione geografica. Le informazioni relative alla posizione possono essere ottenute dal LocationManager in modi diversi a seconda dei LocationProvider disponibili. Si può pensare ad un provider, con caratteristiche descritte da istanze della classe LocationProvider, come a quel meccanismo fisico in grado di acquisire la posizione del dispositivo.

Al momento, i dispositivi Android sono in grado di acquisire le informazioni relative alla posizione attraverso due tipi di provider: GPS o basato sulla rete. I primi utilizzano il segnale proveniente dai satelliti, mentre i secondi calcolano la posizione attraverso triangolazioni relative a posizioni note.

La classe LocationManager ci permette di avere indicazioni sui LocationProvider disponibili ed eventualmente ci permette di verificare quale, tra questi, soddisfa meglio particolari criteri le cui informazioni sono incapsulate in un oggetto di tipo Criteria.

Di seguito viene riportata l'implementazione del metodo che sceglie il miglior provider disponibile al momento sul dispositivo:

```
private void chooseBestProvider()
     {
          Criteria crta = new Criteria();
          crta.setAccuracy(Criteria.ACCURACY_FINE);
          crta.setAltitudeRequired(false);
          crta.setBearingRequired(false);
          crta.setCostAllowed(true);
          crta.setPowerRequirement(Criteria.POWER_LOW);
//inserisco nella stringa Provider il risultato del
metodo richiamato  //sull'oggetto LocationManager lm
          provider = lm.getBestProvider(crta, true);
     }
```

L'utilizzo del LocationManager è molto semplice, in quanto, per essere notificati delle variazioni di posizione, è sufficiente creare un'implementazione dell'interfaccia LocationListener e registrarsi a essa come ascoltatori attraverso il seguente metodo:

public void requestLocationUpdates (String provider,long minTime, float minDistance,LocationListener listener)

Si noti come il primo parametro sia l'identificativo del provider che si intende utilizzare. Questo significa che è possibile registrarsi come ricevitore degli eventi di posizione solo con un provider. I parametri *minTime* e *minDistance* permettono di specificare rispettivamente una distanza ed un tempo minimo di notifica; questo permette di non generare troppi eventi di notifica ravvicinati sia nel tempo, sia nello spazio, in modo da preservare le risorse del dispositivo.

Infine, l'ultimo parametro è il riferimento ad un'implementazione dell'interfaccia LocationListener, la quale prevede di definire una serie di operazioni successivamente descritte. La più interessante è sicuramente quella descritta dal metodo

public abstract void onLocationChanged (Location location)

il quale viene invocato in corrispondenza a una variazione nella posizione le cui informazioni sono incapsulate in un oggetto di tipo Location. Attraverso le due operazioni

public abstract void onProviderDisabled (Stringprovider)

public abstract void onProviderEnabled (String provider)

è possibile essere informati dello stato di disabilitazione o abilitazione di un determinato provider da parte dell'utente. Questo permette, per esempio, di passare al provider di rete qualora il provider GPS fosse disabilitato o viceversa.

Infine, un LocationListener riceve notifiche sulle variazioni di stato del servizio attraverso il seguente metodo:

public abstract void onStatusChanged (String provider,int status, Bundle extras)

In questo caso è interessante come lo status possa essere rappresentato da uno dei seguenti valori, di ovvio significato:

OUT_OF_SERVICE,TEMPORARILY_UNAVAILABLE o AVAILABLE.

Di seguito la registrazione al LocationListener e la sua implementazione utilizzati nell'applicazione:

```
        lm =
(LocationManager)getSystemService(Context.LOCATION_SERV
ICE);
        ll = new MyLocationListener();
        lm.requestLocationUpdates(
                provider,
                60000,
                10,
                ll);
```

```
private class MyLocationListener implements
LocationListener{
           @Override
          public void onProviderDisabled(String
provider) {
             changeProvider();
           }
           @Override
          public void onProviderEnabled(String
provider) {    }
           @Override
          public void onStatusChanged(String provider,
             int status, Bundle extras) {

if(status==LocationProvider.OUT_OF_SERVICE||
status==LocationProvider.TEMPORARILY_UNAVAILABLE )
                  changeProvider();
           }
          @Override
          public void onLocationChanged(Location arg0)
{
                  GeoPoint myGeoPoint = new GeoPoint(

(int)(arg0.getLatitude()*1000000),

(int)(arg0.getLongitude()*1000000));
//sposto il centro della mappa verso la posizione
attuale dell'utente e //chiudo la progressDialog in
attesa di una risposta dal provider
                        mc.animateTo(myGeoPoint);
                        mc.setCenter(myGeoPoint);
                        dialogWait.dismiss();
          }
     }
```

3.4 – MAPVIEW

Android permette di utilizzare gli strumenti per la visualizzazione e la gestione delle famose Google Maps: si tratta delle classi MapActivity e

MapView che, come si vedrà, devono essere sempre utilizzate insieme. Questo perché la classe MapView, specializzazione particolare di View, necessita di alcuni servizi che sono forniti dalla classe MapActivity che, come è facile intuire, è una specializzazione della classe Activity.
Il primo passo indispensabile consiste nell'includere nel progetto le API di Google attraverso pochi semplici passaggi. Una prima osservazione relativamente alla classe MapActivity riguarda il fatto che si tratta di una classe astratta che richiede l'implementazione della seguente operazione:

protected boolean isRouteDisplayed()

Si tratta di un metodo che permette di specificare se la particolare applicazione sta visualizzando informazioni di routing, ovvero relative ai diversi modi di percorrenza delle strade. Non ha una funzione tecnica, ma puramente amministrativa, nel senso che ciascuna applicazione che utilizza le Google Maps deve specificare questo tipo di informazione, perché definito dai termini di licenza. Una procedimento analogo è richiesto per il metodo

protected boolean isLocationDisplayed()

per indicare al server se la mappa sta visualizzando informazioni ottenute da un sistema di localizzazione.

Osservando invece le API della classe MapView si nota la presenza dell'importante metodo

public MapController getController()

il quale permette di ottenere il riferimento a un oggetto, descritto dalla classe MapController del package com.google.android.maps, che contiene le diverse operazioni per gestire al meglio il componente della mappa.

È utile sapere che una posizione geografica è spesso identificata da un'istanza della classe GeoPoint. Si tratta di una classe molto importante, che permette di descrivere un punto sulla mappa in termini di latitudine e

longitudine espresse in microgradi. Un microgrado è dato dal valore dei gradi moltiplicato per 1 milione, ovvero per 10^6:

public GeoPoint(int latitudeE6,int longitudeE6)

Molto interessante è la possibilità di eseguire un'animazione dal punto visualizzato attualmente a uno specificato sempre attraverso un oggetto di tipo GeoPoint. Per fare questo si utilizza uno dei seguenti overload del metodo animateTo() diMapController:

public void animateTo(GeoPoint point)

public void animateTo(GeoPoint point,android.os.Message message)

public void animateTo(GeoPoint point,java.lang.Runnable runnable).

Nel caso del Tamagotchi, si è deciso di utilizzare un metodo *startDiscover()* richiamato nell'*OnCreate* dell'Activity con il compito di centrare la mappa sfruttando le informazioni offline sull'ultima posizione conosciuta del dispositivo e facendo mostrare una ProgressDialog in attesa di dati più precisi provenienti dal provider di geolocalizzazione:

```
private void startDiscover()
     {
          try{GeoPoint initGeoPoint = new GeoPoint(
                  (int)(lm.getLastKnownLocation(
                   provider)
                   .getLatitude()*1000000),
                  (int)(lm.getLastKnownLocation(
                  provider)
                   .getLongitude()*1000000));

         mc.animateTo(initGeoPoint);
         mc.setCenter(initGeoPoint);}
          catch(Exception e){}
         showProgressDialog("IN ATTESA DELLA
GEOLOCALIZZAZIONE...");
     }
```

L'ultima considerazione possibile in relazione agli strumenti messi a disposizione per l'utilizzo dellaMapView è relativo alla possibilità o meno di utilizzare dei controlli predefiniti nella gestione dello zoom.

A tale proposito Android permette di utilizzare un componente automatico di gestione dello zoom, che compare e scompare automaticamente quando l'utente tocca il display. Si tratta di un'opzione configurabile attraverso il seguente metodo:
public void setBuiltInZoomControls(boolean on).

3.4.1 - OVERLAYITEM

Come accennato in precedenza, le Google Maps API permettono di visualizzare degli elementi aggiuntivi sulle mappe sfruttando quelli che sono chiamati overlay e sono descritti da specializzazioni dell'omonima classe. In pratica si tratta di opportuni marker caratterizzati da un'icona e selezionabili attraverso diversi tipi di eventi, tra cui quelli touch. Una volta creati degli Overlay è possibile visualizzarli sulla mappa aggiungendoli alla lista ottenuta attraverso il seguente metodo dellaMapView:
public final java.util.List<Overlay> getOverlays().
La piattaforma Android mette a disposizione due specializzazioni di Overlay descritte dalle classi ItemizedOverlay e MyLocationOverlay.
La prima è una specializzazione che ci permette di visualizzare sulla mappa un insieme di elementi mentre MyLocationOverlay permette invece di visualizzare la posizione corrente del dispositivo attraverso un'integrazione con i sistemi di localizzazione.
Come già riportato, è necessario rappresentare i marker da inserire nella mappa come particolari specializzazioni della classe Overlay.
Nel caso del Tamagotchi si è scelto di specializzare la classe ItemizedOverlay con TamaItemizedOverlay che si pone l'obbiettivo di visualizzare sulla mappa un icona del cucciolo virtuale presso ogni luogo da lui visitato e con un'icona differente in base all'azione compiuta in quel

determinato luogo. Questo è prossibile grazie all'overload del metodo *public void addOverlay(OverlayItem overlay)*

Al quale ora è possibile passare come parametro anche l'icona che il marker dovrà assumere:

```
public void addOverlay(OverlayItem overlay, Drawable
drawable) {
//l'icona necessita di un ridimensionamento prima di
essere assegnata //all'oggetto overlay
          drawable.setBounds(0 -
drawable.getIntrinsicWidth() / 2, 0 -
drawable.getIntrinsicHeight(),
                   drawable.getIntrinsicWidth() / 2,
0);

             overlay.setMarker(drawable);
          mOverlays.add(overlay);
           populate();
}
```

Da non dimenticare l'invocazione del metodo populate(), che permette all'oggetto ItemizedOverlay di organizzare al proprio interno i vari riferimenti agli item, per poterli poi visualizzare in modo ottimizzato.

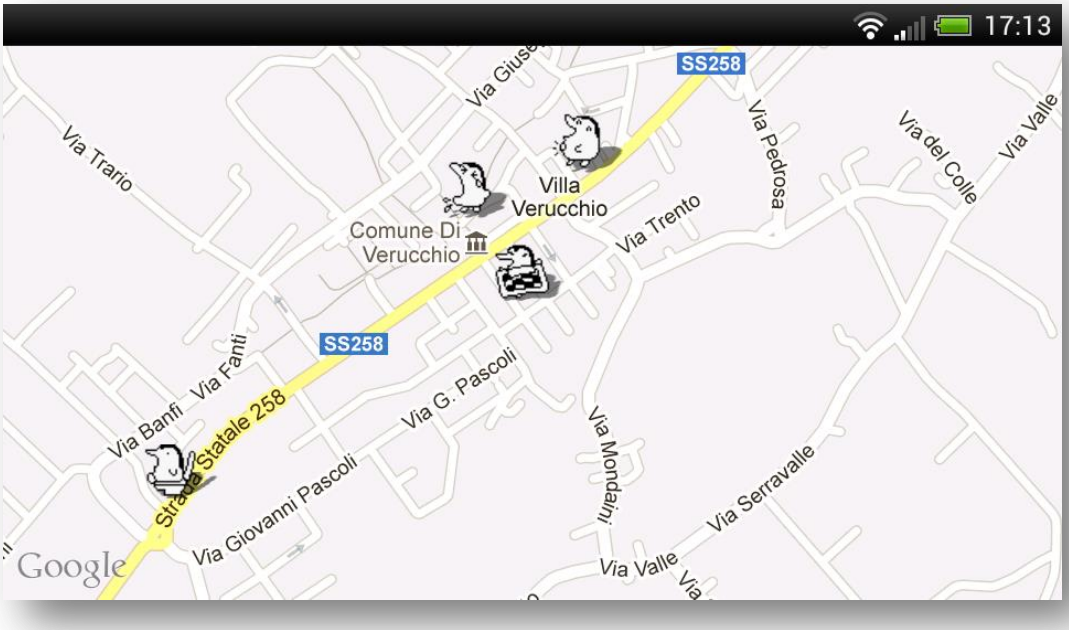
Questa funzione viene richiamata dalla MapActivity una volta ottenuti tutti i dati necessari dalla interrogazione del database locale come mostrato successivamente:

```
private void fillOverlays()
     {
          double longitude,latitude;
          Drawable drawTama =
this.getResources().getDrawable(R.drawable.tama);
          String name,types;
          mapOverlays = mapView.getOverlays();
         TamaItemizedOverlay itemizedoverlay = new
TamaItemizedOverlay(drawTama, this);
         cursorPlaces = PlacesDB.getRegistredPlaces();
```

```
          if(cursorPlaces.getCount()<1)
           return;
           cursorPlaces.moveToFirst();
           do
           {
                     Drawable drawable=null;

     types=cursorPlaces.getString(cursorPlaces.getColum
nIndex("Type"));
                     if(types.equals("food"))
                          drawable=
this.getResources().getDrawable(R.drawable.tama_feed);
                     else if(types.equals("park"))
                          drawable=
this.getResources().getDrawable(R.drawable.tama_play);
                     else
if(types.equals("beauty_salon"))
                          drawable=
this.getResources().getDrawable(R.drawable.tama_shower)
;
                     else if(types.equals("lodging"))
                          drawable=
this.getResources().getDrawable(R.drawable.tama_sleep);
                     else if(types.equals("health"))
                          drawable=
this.getResources().getDrawable(R.drawable.tama_nurse);
                     else
                          drawable=
this.getResources().getDrawable(R.drawable.tama);

     name=cursorPlaces.getString(cursorPlaces.getColumn
Index("Name"));

     longitude=cursorPlaces.getDouble(cursorPlaces.getC
olumnIndex("Longitude"));

     latitude=cursorPlaces.getDouble(cursorPlaces.getCo
lumnIndex("Latitude"));
                     GeoPoint point=new
GeoPoint((int)(latitude*1000000),(int)(longitude*100000
0));
                     OverlayItem overlayitem = new
OverlayItem(point,null, name);
```

```
    itemizedoverlay.addOverlay(overlayitem,drawable);

        }while(cursorPlaces.moveToNext());
        mapOverlays.add(itemizedoverlay);
        cursorPlaces.close();

    }
```

Ottenendo questo risultato:



3.5 – GOOGLE PLACES

Per quanto riguarda l'integrazione con le funzionalità di Google Places, non sono necessarie API specifiche, tutte le informazioni si posso facilmente ottenere tramite delle richieste http.

Dopo avere registrato l'app, indicando la volontà di utilizzare le potenzialità di Google Places, è già possibile iniziare a lavorare sul progetto Android sfruttando alcune API messe a disposizione di Google stessa per la gestione

delle richieste http e l'utilizzo della classe AsyncTask per la loro sincronizzazione all'interno dell'applicativo.

Ebbene, la classe AsyncTask mette a disposizione dei metodi di callback che vengono invocati in corrispondenza di determinati momenti di esecuzione del task. Non solo, alcuni di questi, quelli che devono interagire con la UI, vengono eseguiti nel Main Thread senza l'utilizzo degli Handler.

Il primo di questi, il metodo onPreExecute(), permette di eseguire una qualsiasi operazione sul Thread principale come, ad esempio, la visualizzazione di una ProgressDialog.

Ovviamente un task dovrà eseguire delle operazioni, questa volta in un thread separato rispetto a quello della UI. Questo viene descritto nel metodo *doInBackground* implementato nel seguente modo:

```
        @Override
          protected String doInBackground(String...
args) {
            googlePlaces = new GooglePlaces();
            double
latitude=Double.parseDouble(args[0]);
          double longitude=Double.parseDouble(args[1]);
          double radius = Double.parseDouble(args[2]);
          String types=args[3];
            try {
               nearPlaces =
googlePlaces.search(latitude,
                          longitude, radius, types);

            } catch (Exception e) {
            }
            return null;
        }
```

Il metodo permette di istanziare un oggetto della classe GooglePlaces ed utilizzare la funzione pubblica Search per inviare una richiesta http con tutti i parametri necessari tramite metodo GET ai server Google:

```
public PlacesList search(double latitude, double
longitude, double radius, String types)
            throws Exception {

        this._latitude = latitude;
        this._longitude = longitude;
        this._radius = radius;

        try {

            HttpRequestFactory httpRequestFactory =
createRequestFactory(HTTP_TRANSPORT);
            HttpRequest request = httpRequestFactory
                    .buildGetRequest(new
GenericUrl(PLACES_SEARCH_URL));
            request.getUrl().put("key", API_KEY);
            request.getUrl().put("location", _latitude
+ "," + _longitude);
            request.getUrl().put("radius", _radius); //
in meters
            request.getUrl().put("sensor", "false");
            if(types != null)
                request.getUrl().put("types", types);
            PlacesList list =
request.execute().parseAs(PlacesList.class);
            return list;

        } catch (HttpResponseException e) {
            Log.e("Error:", e.getMessage());
            return null;
        }

    }
```

Questo metodo, in caso di esito positivo, ritorna un oggetto di tipo PlacesList, creato per rendere l'utilizzo della risposta maggiormente accessibile:

```
public class PlacesList implements Serializable {

    @Key
    public String status;

    @Key
    public List<Place> results;

}
```

All'interno di *results*, infatti, sarà presente una lista di oggetti personalizzati Place in grado di mantenere tutti i dati relativi ai POI restituiti nel JSON di risposta.

Tornando all'esecuzione asincrona, infine serve un metodo che permetta di visualizzare il risultato nella UI: si tratta del metodo onPostExecute().

Nel caso di questa applicazione si è deciso di utilizzare questo metodo per chiudere la progressDialog e fare visualizzare il risultato della richiesta all'utente.

Attraverso la definizione di questa classe interna si è quindi descritto il task personalizzato e la sua interazione con i componenti della UI. Serve ora un modo per poterlo lanciare: il tutto si traduce nella semplice creazione di una sua istanza e quindi nell'invocazione del metodo execute().

```
class LoadPlaces extends AsyncTask<String, String,
String> {

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            dialogWait.show();
        }
        @Override
          protected String doInBackground(String...
args) {
```

```
            googlePlaces = new GooglePlaces();
            double
latitude=Double.parseDouble(args[0]);
          double longitude=Double.parseDouble(args[1]);
          double radius = Double.parseDouble(args[2]);
          String types=args[3];
            try {
               nearPlaces =
googlePlaces.search(latitude,
                           longitude, radius, types);

            } catch (Exception e) {
            }
            return null;
        }
        @Override
          protected void onPostExecute(String file_url)
{
          HomeActivity.this.runOnUiThread(new
Runnable(){

                    @Override
                    public void run() {
                         dialogWait.dismiss();
                         fillPlace();
                    }
          });

        }
    }
```

3.6 – MINI-GIOCHI

Ogni mini-gioco è caratterizzato da due AlertDialog: una compare all'inizio dell'attività per mostrare all'utente le istruzioni per ottenere il punteggio più alto possibile, l'altra compare invece nel finale per chiedere al giocatore se ha intenzione di ritentare la sorte, uscire o pubblicare il proprio risultato su Facebook. Esistono poi metodi privati in grado di salvare il risultato nelle

sharedPreferences o di azzerare tutte le variabili in caso l'utente voglia avere un nuovo tentativo di gioco.

SLEEPACTIVITY

Questa Activity incrementa di un unità l'ora corrente e la scrive sotto forma di stringa sulle sharedPreferences insieme ad una variabile booleana *isSleeping*; per questo lasso di tempo nessun altra azione potrà essere compiuta grazie al controllo su *isSleeping* posto prima del lancio di ogni intent. Questa variabile potrà cambiare il suo valore dopo il passaggio di un'intera ora oppure grazie al bottone di sveglia che però renderà impossibile l'aggiornamento dei livelli vitali.

FEEDACTIVITY

Questa Activity ha un'interfaccia grafica molto semplice: presenta principalmente una semplice ProgressBar che incrementa il suo livello grazie ai movimenti rilevati dall'accelerometro.

Per ottenere informazioni sulla quantità e sulla misura di questi movimenti del dispositivo è necessario implementare la classe astratta *SensorEventListener*

ed in particolar modo, il suo metodo

OnSensoriChanged(SensorEvent event).

Una volta ottenute queste informazioni sarà poi banale aggiornare il livello della barra grazie ad un contatore che si incrementerà in caso di un movimento significante del dispositivo:

```
@Override
public void onSensorChanged(SensorEvent event) {


    float x = event.values[0];
```

```
    float y = event.values[1];
    float z = event.values[2];
    float totalMovement = Math.abs(x + y + z - lastX -
lastY - lastZ);
    float XMovement = Math.abs(x- lastX);
    float YMovement = Math.abs(y- lastY);
    float ZMovement = Math.abs(z- lastZ);

    if (totalMovement > MIN_FORCE &&
XMovement>MIN_SINGLE_FORCE &&
YMovement>MIN_SINGLE_FORCE &&
ZMovement>MIN_SINGLE_FORCE) {

          mCountShake++;
          progBar.setProgress(mCountShake);
          if(mCountShake==feedMax)
          {
               crono.cancel();
               showEndDialog();
          }
      } else {
        resetShakeParameters();
      }

    }
```

Il tutto è reso più difficile da un CountDownTimer che ogni due secondi decrementerà il livello della ProgressBar obbligando il giocatore ad aumentare il suo sforzo. È presente anche una seconda barra che indica all'utente lo scorrere del tempo e che cambia il suo colore in caso manchino meno di cinque secondi alla fine della gara.

PLAYACTIVITY

Per la realizzazione di questo mini-gioco è stato necessario creare un file XML di layout particolarmente corposo con nove bottoni a forma di Tamagotchi settati ad INVISIBLE.

L'Activity comprenderà un timer ed un metodo in grado di rendere visibili i bottoni alternativamente in modo casuale, per un brevissimo lasso di tempo. Al giocatore verrà data l'impressione che lo stesso Tamagotchi scompaia e riappaia in modo frenetico in diverse posizioni dello schermo: il suo compitò sarà quello di fare tap prima che il bottone scompaia.
Il punteggio finale sarà calcolato in base al valore di un contatore, incrementato ad ogni tap, al momento dello scadere del timer.

NURSEACTIVITY

In questo mini-gioco viene mostrata al centro dello schermo un'icona rappresentate una diversa immagine contenuta in un componente di tipo ImageView; l'utente deciderà, in base all'immagine, se eseguire una gesture a forma di cerchio oppure a forma di croce.
L'insieme delle informazioni relative alle varie gesture viene rappresentato da un oggetto descritto dalla classe GestureLibrary, di cui è possibile ottenere un riferimento attraverso un insieme di metodi statici della classe GestureLibraries. È possibile, come per questo progetto, caricare da un file le informazioni relative a una GestureLibrary attraverso il seguente metodo statico:
public static GestureLibrary fromFile (String path)
Se il caricamento è andato a buon fine, il passo successivo consiste nella registrazione di un ascoltatore dell'evento di gesture, ovvero dell'interfaccia *OnGesturePerformedListener* la quale prevede la definizione della sola operazione:
public abstract void onGesturePerformed (GestureOverlayView overlay, Gesture gesture)
All'interno di questa implementazione si ottiene il riferimento al particolare GestureOverlayView sorgente dell'evento, ma anche alla gesture percepita.

In particolare si tratta di un'informazione che poi sarà passata alla GestureLibrary per il riconoscimento attraverso l'invocazione del metodo:

public ArrayList<Prediction> recognize (Gesture gesture)

il quale restituisce una lista ordinata di istanze della classe Prediction.

Si tratta di oggetti che incapsulano le informazioni del nome della gesture e del relativo valore di score, che ne indica la vicinanza rispetto al tratto eseguito.

Nel progetto è stato scelto di riconoscere solo gesture con score maggiore di due e permettere all'utente di riprovare in caso di contrario. Solo in caso la gesture venga accettata si valuterà se l'utente abbia dato la risposta esatta o errata ed in qualunque caso verrà richiamato il metodo per proseguire con l'immagine successiva.

```
        gestureView.addOnGesturePerformedListener(new
OnGesturePerformedListener(){


                @Override
                public void
onGesturePerformed(GestureOverlayView gestView, Gesture
gesture) {
                    ArrayList<Prediction> predictions =
gestureLibrary.recognize(gesture);
                    if(predictions!=null &&
predictions.size()>0 && predictions.get(0).score>2){
                        Prediction bestPrediction =
predictions.get(0);

     if((bestPrediction.name.equals("Circle")&&gestureB
ool)||(bestPrediction.name.equals("Cross")&&!gestureBoo
l))
                        {
                            score++;
                            updateScore();

     Toast.makeText(NurseActivity.this, "PERFETTO!",
Toast.LENGTH_SHORT).show();
                        }
```

```
                        else

     Toast.makeText(NurseActivity.this, "DECISIONE
ERRATA", Toast.LENGTH_SHORT).show();
                        randImg();
                    }else
                        // Non e' stata riconosciuta
alcuna Gesture

     Toast.makeText(NurseActivity.this, "NON CAPISCO!
PROVA AD ESSERE PIU' VELOCE!",
Toast.LENGTH_SHORT).show();
                }
        });
```

SHOWERACTIVITY

Per la realizzazione di questo mini-gioco è stato necessario creare un file XML di layout comprensivo di diciotto bottoni aventi come background un'immagine a forma di bolla e visibility settata ad invisible.

L'Activity, utilizzando un CountDownTimer, ad intervalli di un secondo richiamerà un metodo in grado di rendere visibile un bottone a caso e di assegnargli un'animazione personalizzata.

L'animazione è stata realizzata tramite un file XML BubbleUp.xml e, sfruttando la composizione di due diverse animazioni di tipo Translate, simulerà il movimento di una bolla di sapone:

```
<set
xmlns:android="http://schemas.android.com/apk/res/andro
id"
    android:shareInterpolator="true"
    android:fillAfter="true"
    >

    <translate
```

```
        android:toYDelta="-100%p"
        android:duration="4000"

android:interpolator="@anim/bounce_interpolator" />

     <translate
        android:toXDelta="-50%"
        android:fromXDelta="50%"
        android:repeatMode="reverse"
        android:duration="500"
        android:repeatCount="infinite"

android:interpolator="@anim/bounce_interpolator" />


</set>
```

Terminate tutte le animazioni comparirà al centro dello schermo una EditText con InputType settato a Numer che chiederà al giocatore di indicare il numero di bolle che sono comparse nello schermo.

Il mini-gioco sarà considerato superato solo in caso di risposta esatta.

CONCLUSIONI

Questo progetto aveva come obbiettivo quello di realizzare un'applicazione che rivisitasse un vecchio gioco di successo come quello del Tamagotchi in chiave moderna.

Per farlo è stata necessaria una fase di ricerca e sviluppo sia per quanto riguarda alcuni componenti esterni come Google Places sia per l'integrazione con le API fornite da Facebook. Il materiale messo a disposizione come documentazione è sempre stato molto chiaro e ricco di esempi tanto da rendere la fase di ricerca più facile del previsto. Una volta superato questo ostacolo non si sono presentati altri grossi problemi in quanto avevo già fatto diverse esperienze con Android ed il suo ambiente di sviluppo.

Per la prima volta, però, mi sono trovato ad affrontare un progetto di questa portata solo con l'ausilio del mio relatore Mirko Ravaioli e senza alcun team di sviluppo, come invece era già capitato in passato.

Questa situazione mi ha aiutato nel cercare di risolvere ogni dubbio o problema documentandomi e cercando autonomamente di trovare una soluzione plausibile; inoltre, avendo avuto discreta libertà di progettazione, è stato fondamentale per me poter prendere diverse scelte importanti per il proseguo dell'applicazione, responsabilizzandomi per la realizzazione dell'intero progetto.

BIBLIOGRAFIA

-Massimo Carli, “Andorid 3 Guida per lo sviluppatore”

- Grant Allen, “Beginnig of Android 4”

- Andrea Boaretto, “Mobile marketing”

- Ravaioli Mirko, Dispense del corso di Mobile Web Design, 2011/2012

- S. James , “Android application development for java Programmers”

- “Lezioni Android”, http://www.sauronsoftware.it/teaching/uniroma2/android/

- “Guida allo sviluppo Android “, http://www.mrwebmaster.it/mobile/guide/guida-sviluppo-appsandroid/

- “Guida allo sviluppo Android “, http://android.devapp.it/