VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

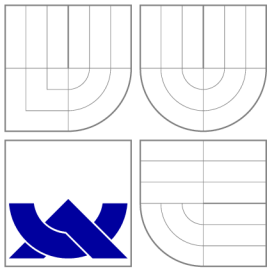
TRÉNINKOVÝ DENÍK PRO MOBILNÍ TELEFON S GPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

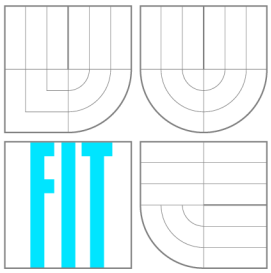
AUTOR PRÁCE
AUTHOR

MARCEL SYRŮČEK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

TRÉNINKOVÝ DENÍK PRO MOBILNÍ TELEFON S GPS

TRAINING DIARY FOR CELL PHONE WITH GPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARCEL SYRŮČEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN HORÁČEK
SUPERVISOR

BRNO 2011

Abstrakt

Bakalářská práce se zabývá vytvořením aplikace, která bude sloužit jako tréninkový deník pro mobilní platformu Android s využitím technologie GPS. Hlavním úkolem tohoto programu je zaznamenat absolvovanou trasu při jakémkoliv sportu, který je uskutečněn v dosahu signálu GPS. O této trase lze přímo v telefonu též zjistit dostupné údaje, vykreslit trasu do mapy se zobrazením podrobností v konkrétních bodech či exportovat trasu do standardních formátů pro práci se zeměpisnými údaji. Největší přínos této práce je však v podpoře běhu s takzvaným přítelem, kdy se při záznamu trasa rovnou porovnává s podobnou nebo stejnou trasou uloženou v databázi a oznamuje nám, jaké jsou mezi jednotlivými trasami výkonnostní rozdíly.

Abstract

The main task of this program is to record the completed route in any sport that is carried in the GPS range. It is possible to find out information about the route, such as all the available data, plot a route on the map to view details on specific points, or export route in standard formats for working with geographic data, directly through the phone. The greatest contribution of this work lies in "running with so-called friend"application, in which the recorded route is directly compared with the same or similar track stored in the database and notify us, what are the differences between the performance.

Klíčová slova

GPS, Android, sport, porovnávání dvou tras, GPX.

Keywords

GPS, Android, sport, comparing two tracks, GPX.

Citace

Marcel Syrůček: Tréninkový deník pro mobilní telefon
s GPS, bakalářská práce, Brno, FIT VUT v Brně, 2011

Tréninkový deník pro mobilní telefon s GPS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Horáčka

.....
Marcel Syrůček
18. května 2011

Poděkování

Na tomto místě bych rád poděkoval Ing. Janovi Horáčkovi za odborné vedení, poskytnutí věcných připomínek a vzájemnou spolupráci při tvorbě této bakalářské práce.

Obsah

Kapitola 1

Úvod

V posledních letech docházelo ve velké míře ke zlevňování GPS modulů a z tohoto důvodu i k jejich rozšíření do dalších oblastní mobilních technologií. Dnes na trhu prakticky nenajdete telefon s operačním systémem *Google Android*, který by ho neměl. Existuje i spousta aplikací, které pracují se získáváním a zpracováváním informací o poloze uživatele. Mezi nejznámější patří typické navigace. Čím dál více se však rozšiřují i aplikace, které zaznamenávají absolvovanou trasu a poté ji dále analyzují a vyhodnocují například pro sportovní účely. Obecně se jim říká *sports trackery* a právě o takové aplikaci je následující práce.

Jako každý sports tracker umí i tento zaznamenat absolvovanou trasu a exportovat ji do standardního formátu pro ukládání GPS souřadnic. Navíc přidává možnost zobrazení v *Google Maps*, kde lze například zjistit rychlost v aktuálním bodě. Dále umožňuje podporu hlasového upozornění, rozdělení do kategorií dle prováděných činností nebo vedení dlouhodobých statistik.

Největší přínos této práce však spočívá v porovnávání dvou tras v reálném čase. Odtud pochází i název aplikace **Sport with friend**. A co si lze pod tímto pojmem představit? Jde o porovnávání aktuálně zaznamenávané trasy s trasou uloženou v databázi telefonu a zjišťování, jak moc se aktuálně zaznamenávaná trasa liší od vzorové. Pokud jsou nalezeny nějaké společné úseky, aplikace nám v nich přehledným způsobem zobrazuje rozdíly aktuálního a předešlého výkonu, a my tedy víme, zda jsme se oproti minulému absolvování trasy zlepšili nebo zhoršili, případně o kolik. Předchozí trasa nám při běhu vlastně simuluje člověka, s kterým můžeme závodit.

Následující kapitola pojednává o operačním systému Google Android. Je zde zmíněn jeho vznik, specifika, možnosti vývoje a samozřejmě principy, na kterých Android funguje. Zde také zdůvodním, proč jsem si vybral právě programování aplikace pro tento operační systém.

Ve 3. kapitole je popsán návrh aplikace, ve kterém objasním, jaké problémy bylo potřeba vyřešit před samotnou implementací programu. Též jsou zde uvedeny jednotlivé kroky k jejich odstranění.

Kapitola 4 zahrnuje vlastní implementaci aplikace a problémy, které mě při ní potkaly. S tím souvisí kapitola 5, která se zabývá testování jednotlivých funkcí i celé aplikace.

V závěru 6 už pouze dojde k zhodnocení aplikace a její implementace. Navrhnu zde také případná rozšíření a směr jejího dalšího vývoje.

Kapitola 2

Operační systém Google Android

Tato kapitola byla volně převzata z [3] a [13]. V této kapitole se zmíním o historii Androidu a jeho různých verzích, detailně proberu možnosti vývoje a distribuce aplikací, architekturu operačního systému a nakonec nastíním základní principy a paradigmata programování pod touto platformou.

2.1 Historie

V nynější době už mobilní telefony neslouží pouze ke svému původnímu účelu, kterým bylo telefonování a posílání SMS, ale stále více se do nich prosazují i různé multimediální funkce, mezi které můžeme řadit focení, natáčení videí, prohlížení Internetu a v neposlední řadě i aplikace a služby pracující s polohou uživatele.

Tento trend nastal především po uvedení mobilního telefonu *iPhone* společností *Apple* v roce 2007 a dále se rozvíjí. Svůj podíl na tom má i společnost Google, která v roce 2005 koupila společnost *Android Inc.*, jenž se zabývala vývojem softwaru pro mobilní zařízení. O dva roky později spoluzakládá Google sdružení firem *Open Handset Alliance*, jehož cílem bylo zlepšení, inovace a hlavně produkce otevřených řešení v sektoru mobilních technologií. Jako svojí platformu si sdružení vybralo právě software od společnosti Android a rozhodlo se ho dále vyvíjet pod svobodnou licencí *Apache* a *GPL v2*. Nyní toto sdružení čítá přes 80 společností, jež jsou ze všech důležitých oblastí mobilních technologií. Jako příklady lze uvést softwarového giganta Google, mobilního operátora *T-Mobile* nebo výrobce hardwaru a koncových zařízení *HTC, Intel, Samsung, Motorola...*

Dalším významným bodem je den 23. září 2008, kdy byl oficiálně představen operační systém Android 1.0 a do prodeje uveden první telefon běžící na tomto systému *HTC Dream*, který se ve většině zemí prodával jako brandovaný telefon s názvem *G1* od operátora *T-Mobile*. Tento systém, a tím pádem i telefon, obsahoval prohlížeč webových stránek, podporu pro focení fotoaparátem, přístup k emailu za pomocí protokolů POP3, IMAP4 a SMTP, synchronizaci a práci s Google službami jako jsou *Kontakty, Gmail, Kalendář, Mapy, Talk, Youtube...*, podporoval Wi-Fi a přístup do Android Marketu, o kterém se zmíním později (viz 2.3.3). Postupem času přibývaly další verze, jež obsahovaly opravy chyb, přidání nových funkcí nebo zlepšení ovládání. Seznam všech verzí a jejich rozšíření lze nalézt na webu [4]. Nynější nejnovější verzí pro mobilní telefony je *2.3.3 Gingerbread* vydaná v únoru 2011, avšak nejpoužívanější verzí stále zůstává *2.2 Froyo*. Pro nasazení na tabletech je určena speciální verze *3.0 Honeycomb*, která podporuje větší obrazovky, více procesorových jader a pokročilejší akceleraci pro grafické karty.

A jak si stojí Android dnes? Během necelých tří let se stal lídrem v oblasti chytrých mobilních telefonů, tzv. *smartphones*, když je denně nově aktivováno přes 350 000 [14] přístrojů a počet aplikací v *Android Marketu* převýšil počet aplikací v ikoně tohoto způsobu distribuce aplikací *AppStore* (úložiště aplikací pro mobilní telefony *iPhone* a tablety *iPad*).

2.2 Architektura operačního systému Google Android

Architektura operačního systému Google Android je dle obrázku 2.1 rozdělena do 5 skupin.

Android platforma používá Linuxové jádro verze 2.6. **(Linux kernel)**, které zodpovídá za základní správu hardwaru, jako jsou správa ovladačů, přístup k paměti, využívání zařízení nebo správa napájení. Též slouží jako abstraktní vrstva mezi hardwarem a zbytkem architektury.

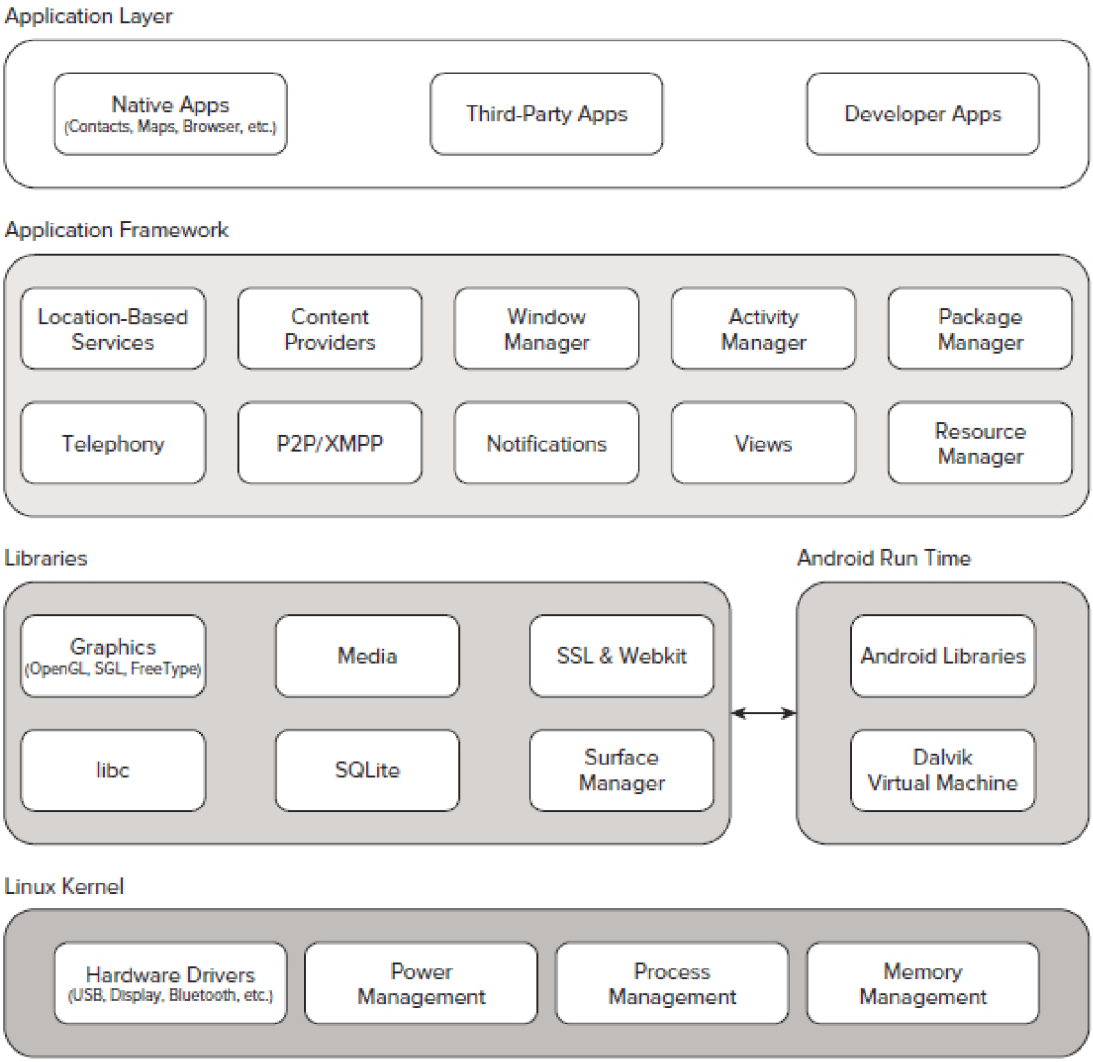
Další vrstva **Libraries** zahrnuje C/C++ knihovny, které používají komponenty OS Android. Mezi základní knihovny patří například:

- **Systémové knihovny C** - knihovny *libc* optimalizované pro vestavěná zařízení.
- **Multimediální knihovny** - postaveny na *OpenCORE*, umožňují přehrávání a záznam populárních audio a video formátů (MPEG4, MP3, ACC), stejně tak jako zobrazení obrázků (JPG, PNG).
- **Knihovny pro 2D a 3D grafiku** - umožňují i hardwarovou 3D akceleraci pomocí *OpenGL ES 1.0*.
- **LibWebCore** - knihovny pro zobrazení webů.
- **SQLite** - jednoduchá open-source relační databáze.

Vrstva **Android Runtime** obsahuje knihovny, které poskytují funkcionalitu základních knihoven jazyka Java. Každá aplikace se v Androidu spouští ve svém vlastním procesu se svou vlastní instancí virtuálního stroje. Ten se zde nazývá *Dalvik virtual machine* a byl speciálně vyvinut pro tuto platformu, jelikož standardní *Java virtual machine* nebyla pro nasazení na mobilních zařízeních dostatečně výkonná a spotřebovávala mnoho systémových prostředků. Nyní však probíhá soudní spor ohledně některých součástí *DVM*, jelikož si společnost *Oracle* nárokuje licenční práva na část použitých zdrojových kódů. *Dalvik virtual machine* spouští soubory s příponou *.dex* a zajišťuje komunikaci s jádrem pro správu vláken nebo paměti.

Application Framework poskytuje API *(Application Programming Interface)* použité pro vývoj aplikací. Největší rozdíl vzhledem k ostatním mobilním platformám je ten, že v Androidu jsou všechny aplikace rovnocenné. To znamená, že my jako vývojáři máme přístup ke stejnému API, jako mají tvůrci systému. Není pro nás tedy problémem naprogramovat vlastního správce kontaktů či kalendář. Uplatňují se zde také moderní principy programování, jako jsou jednoduchý návrh aplikací a znovupoužitelnost komponent, při kterém můžeme používat mnoho funkcí vestavěných aplikací.

Poslední vrstvou jsou **Applications**, které obsahují základní aplikace, například emailový klient, program pro správu SMS, kalendář, internetový prohlížeč atd.



Obrázek 2.1: Architektura operačního systému Google Android [13]

2.3 Možnosti vývoje

V této podkapitole se dozvíme informace o programovacím jazyce, ve kterém se aplikace pro Android vyvíjí, o *Android software development kitu (SDK)*, který je dostupný pro všechny hlavní platformy - Windows, Linux i Mac. Zmíním se zde i o možnosti distribuce vyvinuté aplikace.

2.3.1 Programovací jazyk

V Androidu se aplikace píší primárně pomocí programovacího jazyka Java, což umožňuje poměrně rychlý začátek vývoje. Je ovšem nutné si uvědomit, že se zde musíme naučit programovat i mobilně, jelikož by klasicky psané programy spotřebovaly velké množství dostupných zdrojů.

Android API [10] odpovídá zčásti *Java SE* a ve svém základu obsahuje i užitečné *Apache* knihovny pro aplikace pracující v prostředí Internetu. Tento základ samozřejmě doplňují různá API, která jsou Androidu „ušita na míru“, například práce s grafikou, Bluetooth, SQLite databází nebo GPS. Pro aplikace, typicky hry, kde potřebujeme dosáhnout vysokého výkonu, můžeme využít *Android NDK (Native development kit)*, což je balík pro vývoj aplikací v C/C++.

2.3.2 Android SDK

Android SDK nám poskytuje mnoho nástrojů, které nám pomáhají s vývojem aplikace pro platformu Android. Mezi hlavní nástroje patří:

- **Vývojové nástroje** - slouží ke kompilaci a k ladění aplikací.
- **Android Virtual Device Manager** - přes tento program se stahují jednotlivé komponenty SDK, různé verze platformy Android a jejich rozšíření.
- **Android emulator** - virtuální mobilní zařízení, které běží v počítači. Umožňuje nám vývoj a testování aplikací bez použití fyzického zařízení.
- **Dalvik Debug Monitor Server** - využívá se k ladění aplikací, získávání informací o jednotlivých vláknech nebo i simulování GPS polohy pomocí GPX souborů.
- **sqlite3** - slouží pro správu SQLite databází, do kterých ukládají aplikace data.

Společně s *Android SDK* je dodávána i rozsáhlá dokumentace zahrnující základní třídy jazyka *Java*, užitečné *Apache* třídy a samozřejmě i popis *Android API*. Mezi tuto dokumentaci lze zařadit i zdrojové kódy ukázkových aplikací, které demonstrují jednoduché využití poskytovaného rozhraní.

Při programování aplikací máme na výběr ze dvou integrovaných prostředí *Eclipse* [7] a *IntelliJ IDEA* [6] nebo můžeme použít prostředí příkazového řádku. Nejvíce doporučovaná a zároveň i nejvíce používaná je varianta vývoje pomocí *IDE Eclipse* s přídavným pluginem *ADT (Andoroid Development Tools)* [2]. Ten rozšiřuje možnosti *Eclipse* o rychlé nastavení nových Android projektů, vytvoření uživatelského rozhraní, překlad aplikací nebo usnadňuje jejich ladění.

2.3.3 Android Market

Android Market [5] je služba, která umožňuje uživatelům vyhledávání, stahování a následnou aktualizaci aplikací pro jejich zařízení, ze kterého se k Marketu i přímo přistupuje. Lze zde také zakoupit placené aplikace. Nutno však podotknout, že tato možnost byla nejdříve dostupná pouze v USA a Velké Británii. V ČR se dají aplikace legálně zakoupit teprve od podzimu 2010.

Pro vývojáře naopak poskytuje Android Market snadnější distribuci jejich aplikací, zpětné ohlasy od uživatelů nebo zjednodušené vybírání poplatků za placené programy. Abychom v něm mohli publikovat své aplikace, je nutné uhradit počáteční poplatek 25 $.

2.4 Zásady vývoje

V této části budou popsány základní komponenty pro vývoj aplikací, vysvětlena bezpečnostní opatření, která musí každá aplikace deklarovat, a nakonec bude uvedeno, jak by se

mělo v Androidu vytvářet uživatelské rozhraní. Pro další vysvětlení bude vhodné uvést, co všechno vlastně Android aplikace obsahuje. Jsou to zkompilované zdrojové soubory, které se společně s ostatními zdroji (UI, zvuk, jazykové verze, obrázky...) a takzvaným *Android Manifestem* (viz 2.4.2) zabalí do ZIP souboru s příponou `.apk`, který slouží k distribuci a instalaci na zařízení.

2.4.1 Základní komponenty

Tyto komponenty jsou základními stavebními bloky každé aplikace a umožňují její specifické chování. Každá komponenta má svůj životní cyklus a rozdílné způsoby, jak lze vytvořit. Pokud vytváříme vlastní aplikaci, nejčastěji voláme nebo dědíme z následujících komponent.

Activity

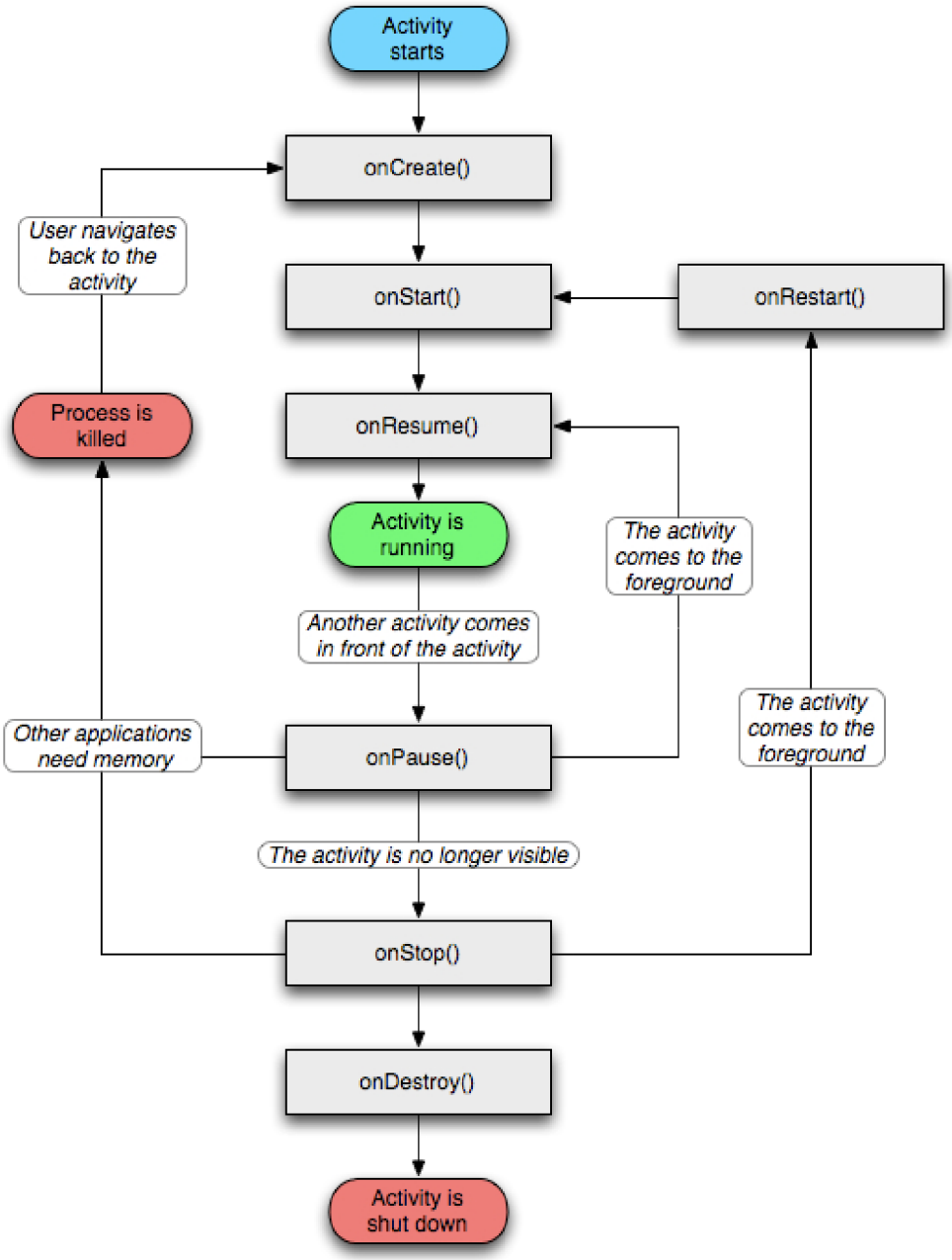
Aktivita by ve většině případů měla zobrazovat jednu obrazovku uživatelského rozhraní a využívá se právě pro komunikaci s uživatelem, kdy slouží jako prezentační vrstva. Může nám třeba volat jiné aktivity nebo zjišťovat stav služeb spuštěných na pozadí. Typický představitel může být například obrazovka znázorňující ovládací prvky pro MP3 přehrávač.

Vzhledem k tomu, že je tato komponenta nejvíce využívána a má nejširší možnosti stavů, do kterých se může během vykonávání aplikace dostat, detailně si ukážeme a popíšeme její životní cyklus. Nejprve je však nutné uvést, jak vlastně aplikace fungují. Téměř všechny aplikace se totiž skládají z více než jedné aktivity, a tak je tedy nutné nějakým způsobem řídit jejich přepínání. K tomu slouží **zásobník** *(stack)*, někdy též pro svoji funkčnost nazývaný *back stack*. Pokud se spustí nová aktivita A, vloží se na vrchol zásobníku a je dostupná pro komunikaci s uživatelem (říkáme, že má fokus). Po nějaké době uživatel zavolá jinou aktivitu B, která je opět vložena na vrchol zásobníku a předchozí aktivita A se zneviditelní. Uživatel dokončí úlohu v současné aktivitě B a zmáčkne tlačítko BACK, čímž se z vrcholu zásobníku odebere současná aktivita B a fokus dostane aktivita A. Toto je pouze zjednodušené vysvětlení, a Android samozřejmě nabízí spoustu dalších možností, jak s aktivitami na zásobníku zacházet.

Práce zásobníku a následující vysvětlení životního cyklu je nutné pro pochopení, jak Android pracuje se správou operační paměti a celkových výkonových prostředků. Pokud Android zjistí, že zásobník obsahuje aktivity, které nebyly delší dobu použity, rozhodne se je ukončit, aby uvolnil systémové prostředky. V zásobníku tedy zbude pouze viditelná aktivita, která je na jeho vrcholu. Ptáte se, proč Android standardně neukončuje všechny aktivity hned po jejich překrytí jinou aktivitou? Odpověď je jednoduchá a souvisí s rychlostí odezvy aplikací. Pokud je nějaká aktivita s různými *widgety* nahraná v paměti, není ji nutné po skončení rušit, jelikož při jejím dalším zobrazení bychom čekali na načtení grafických prvků do paměti. Zůstává tedy stále uložená v paměti až do té doby, dokud systém nepotřebuje další prostředky pro jiné aktivity nebo pokud ji programově neukončíme.

Obrázek 2.2 zobrazuje **životní cyklus** aktivity, která se může za svého života nacházet ve třech základních stavech – být na popředí a interagovat s uživatelem, být pouze viditelná nebo může být pozastavená na pozadí. Při přechodu aktivity mezi stavy jsou volány systémové metody, z nichž následující se nejčastěji překrývají:

- **onCreate()** - je volána při vytvoření aktivity, a mělo by zde docházet k inicializaci *widgetů* nebo svázání dat mezi grafikou a databází.
- **onResume()** - spouští se po umístění aktivity na vrchol zásobníku, a může dostávat uživatelský vstup.



Obrázek 2.2: Životní cyklus aktivity [1]

- **onPause()** - nastává při odebrání fokusu aktivitě, ovšem data stále zůstávají v paměti.
- **onDestroy()** - před ukončením aktivity je zavolána tato metoda, která vyčistí systémové prostředky alokované touto aktivitou.

Service

Služba běží na pozadí a využívá se pro operace a služby, které zabírají více času nebo jsou volány vzdáleně. Nemá žádné uživatelské rozhraní a je ovládána například aktivitou nebo může trvat tak dlouho, dokud nedokončí úlohu, pro kterou byla spuštěna. Jako příklad zde lze uvést samotné přehrávání hudby v MP3 přehrávači.

Content provider

Poskytovatel obsahu nám umožňuje spravovat nebo sdílet data různých aplikací, případně mohou k datům vytvořeným naším programem přistupovat i jiné aplikace. Velmi často se zde používá práce s *SQLite databází*, kde se například uchovávají informace o kontaktech nebo obrázcích uložených v zařízení.

Intent

Tato komponenta by se dala zřejmě přeložit do češtiny jako úmysl nebo záměr, avšak její funkce spíše odpovídá pojmu zpráva tak, jak ho známe z jiných programovací jazyků. *Intenty* slouží zkrátka pro komunikaci mezi jednotlivými komponentami aplikace nebo i mezi komponentami v celém systému. Posílá se v nich popis operace, která se má provést. Je možné zavolat například jinou aktivitu, která nám navrátí telefonní číslo ze seznamu kontaktů nebo přepne na displeji na námi vytvořenou aktivitu, která bude obsahovat informace z nynější obrazovky.

Broadcast receiver

Pro reakci na některé Intenty lze použít takzvaných přijímačů plošných zpráv, kdy například služba přehrávající muziku zašle při skončení seznamu písní do aktivity, která má právě zaregistrovaný příslušný *Broadcast receiver*, zprávu o konci přehrávání. Ten na toto oznámení reaguje naprogramovaným způsobem. Při předchozím příkladu reciever naslouchal na námi definovanou zprávu, avšak je možné reagovat i na systémová oznámení, třeba slabou baterii nebo příchozí SMS zprávu.

2.4.2 Android Manifest

Každý projekt vyvíjený pro operační systém Android musí obsahovat ve svém kořenovém adresáři soubor `AndroidManifest.xml`, ve kterém je definována struktura projektu a uvedena oprávnění, jež pro svůj běh aplikace potřebuje. V následujícím seznamu jsou popsány nejdůležitější vlastnosti daného souboru: (kompletní seznam lze nalézt v [11])

- **Oprávnění** - jsou zde uvedena všechna oprávnění, která aplikace pro svůj běh potřebuje. Například pro volání telefonního čísla je nutné přidat do manifestu argument `android.permission.CALL_PHONE`. Seznam oprávnění, které aplikace požaduje, musí uživatel vždy schválit při instalaci aplikace do zařízení.
- **Seznam komponent** - manifest musí též obsahovat seznam všech komponent, jichž aplikace využívá.
- **Knihovny a verze SDK** - upřesňují, pod jakou nejnižší a nejvyšší verzí SDK může aplikace běžet, pod jakou byla přeložena a případně i jaké knihovny pro svůj chod

potřebuje. Zde se často linkuje knihovna *com.google.android.maps*, která poskytuje práci s *Google Maps*.

- **Velikost displeje** - Android identifikuje displeje dle jejich rozlišení na *smallScreens, normalScreens, largeScreens* a *anyDensity*. Proto například nejdou některé hry spustit na zařízeních s malým rozlišením.

2.4.3 Aplikační prostředky

Koncepce Androidu se snaží dodržovat architekturu *MVC (Model-view-controller)*, která odděluje datový model aplikace, uživatelské rozhraní a řídící logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní. To umožňuje optimalizovat aplikaci pro různé typy zařízení, například v závislosti na jazyku, velikosti či natočení displeje.

Tohoto se často využívá při rozložení ovládacích prvků na obrazovce, různých jazykových mutacích, volbě vzhledu aplikace nebo rozlišení displeje.

Souhrnně se jedná o všechny soubory umístěné v projektovém podadresáři `res`, kde se nalézá více podadresářů, z nichž každý slouží pro dané typy souborů (definice vzhledu uživatelských obrazovek, obrázkové soubory, jazykové překlady…).

2.4.4 Uživatelské rozhraní

Pro různé prvky uživatelského rozhraní *(widgety)* nám slouží spousta tříd odvozených od třídy `View`. Patří sem například tlačítka, textová pole, mapy… Tyto widgety mohou být uspořádány do různých vrstev *(layoutů)*, kde se určuje jejich vzájemná poloha, velikost nebo viditelnost. Je samozřejmě možné si vytvořit vlastní uživatelské prvky, a to buď kompletně nové, kde musíme nadefinovat vše od vykreslení až po jeho interakci, nebo můžeme rozšířit již stávající widgety o námi požadované vlastnosti.

Při vytváření uživatelského rozhraní máme v Androidu dvě možnosti. Tou první je definice použitých widgetů přímo ve zdrojovém kódu aplikace pomocí tříd k tomu určených. Tato možnost sice umožňuje větší dynamičnost uživatelského prostředí za běhu aplikace, ale kód se stává poté velmi nepřehledným a nedodržuje se koncepce MVC.

Druhou a doporučovanou možností je definice prvků uživatelského rozhraní v souborech XML, které se poté pouze vloží do kódu aplikace a vytvoří objektovou reprezentaci uživatelského rozhraní totožnou s tou, která se vytváří přímo v Java kódu. Vlastnosti jednotlivých prvků se nastavují pomocí atributů elementů XML, kde je jejich škála stejně velká jako při vytváření ve zdrojovém kódu. Tato možnost nám však umožňuje pouze definici statických obrazovek, které s uživatelem mohou komunikovat zcela minimálně.

Výhodné je tedy kombinovat obě možnosti, a to tím způsobem, že v souborech XML jsou nadefinovány widgety uspořádané do layoutů a v kódu se na ně pouze odkazujeme. Tím můžeme měnit jejich vlastnosti za běhu programu nebo například získávat data z textových polí.

Dalším velmi důležitým prvkem pro uživatelské rozhraní je menu, které umožňuje spolehlivé zobrazení funkcí a nastavení aplikace. Nejčastěji se používá menu aplikace, které se aktivuje při kliknutí na tlačítko MENU a které slouží pro nastavení aplikace nebo vyvolání funkcí aplikace. Další nabídkou je kontextové menu, které se zobrazí při delším stisku položky na displeji a typicky rozšiřuje možnosti nabídky pro danou položku. Menu lze opět definovat přímo ve zdrojovém kódu nebo v XML, avšak vzhledem ke složitosti, respektive

k jednoduchosti jeho vytvoření a další interakci s aplikací, se spíše používá tvorba menu přímo ve zdrojovém kódu.

Pro zobrazení rozsáhlejších dat, například z databází, slouží *adaptéry*, které dokáží propojit jednotlivé hodnoty s předem danými grafickými prvky. Toho se často využívá při zobrazení dat do všemožných seznamů, kde může být v jedné položce zahrnuta spousta dalších informací.

Pro úplnost zbývá ještě zmínit, že vzhled téměř všech grafických prvků může být různým způsobem upraven. Tedy, že na ně lze aplikovat různé styly nebo témata, která se opět nacházejí ve zdrojích aplikace.

Kapitola 3

Návrh

V této kapitole bude vysvětleno, za jakým účelem aplikaci vlastně programuji, co by měla navíc oproti ostatním aplikacím ze stejné kategorie nabízet a budou uvedeny detaily jejího návrhu. Zaměřím se zejména na práci s GPS, efektivní porovnávání dvou tras a uložení samotných dat.

3.1 GPS

Global Positioning System, zkráceně GPS, je název pro globální družicový systém provozovaný Ministerstvem obrany Spojených států amerických, díky němuž lze určit polohu a přesný čas kdekoli na Zemi. Detailnější popis tohoto systému obsahující i popis zdrojů chyb lze nalézt například v [15]. Aby mohl přijímač určit svoji polohu, musí znát rozmístění viditelných satelitů na obloze a mít synchronizovány vnitřní hodiny s atomovými nacházejícími se v družicích. Teoreticky je možné zjistit zeměpisnou výšku, šířku a nadmořskou výšku na základě tří satelitů. Z důvodů zpožděných hodin je to však prakticky nemožné, takže pokud chceme zjistit naši polohu bez nadmořské výšky, musíme být v dosahu minimálně tří satelitů. Jestliže chceme znát i údaje o ní, bude potřeba přijímat signál od čtyř satelitů. Samozřejmě zde platí pravidlo, že čim více satelitů máme, tím větší je přesnost určené pozice. Vzhledem k tomu, že nemáme v aplikaci téměř žádnou možnost jak zlepšit signál GPS, zaměřím se v podkapitole 3.3 pouze na způsob odstranění špatně získaných poloh.

3.1.1 A-GPS

Naopak co můžeme velmi ovlivnit, je rychlost počátečního výpočtu pozice, takzvaný *GPS fix*, který označuje funkční spojení GPS přijímače se satelity. Pokud zapneme GPS bez tohoto rozšíření, trvá nalezení nutných tří satelitů obvykle v řádu několika minut. Naopak s *Assisted GPS* můžeme tuto dobu urychlit, jelikož pomocí mobilních datových přenosů lze oproti klasickému přenosu ze satelitů rychleji získat takzvaný *almanach* (databázi), obsahující satelity, které by v danou dobu měl přijímač vidět. V rámci almanachu se též přenáší aktuální ionosférický model Země, jenž nám umožňuje dále zpřesňovat polohu (viz opět [15]). Nutno ovšem podotknout, že se almanachy aktualizují přibližně jednou týdně, takže není nutné mít A-GPS zapnuté neustále.

3.1.2 GPX

Jako standard pro uchovávání a další zpracovávání GPS dat slouží formát *GPX (GPS eXchange Format)* [8], který je založen na XML schématu. V současné době se používá verze 1.1. Kromě metadat lze do něj vepsat následují údaje:

- **Waypoint** - bod se souřadnicemi nebo pojmenované místo na mapě.
- **Route** - cesta obsahující seznam uspořádaných waypointů, která naviguje uživatele k cíli.
- **Track** - trasa s uloženými waypointy, které uživatel absolvoval.

Na přiloženém CD se ve složce GPX nacházejí soubory, které byly exportovány aplikací Sport with friend a jsou v nich uloženy data o absolvovaných trasách.

3.2 Funkce aplikace

Při výběru bakalářské práce jsem byl rozhodnutý, že chci naprogramovat nějakou užitečnou aplikaci pro Android. Po přečtení loňských zadání, která se věnovala mobilním platformám, mě velmi zaujala aplikace sloužící jako tréninkový deník pro *OS Symbian* a *Windows Mobile* porovnávající dvě trasy. Tato možnost však probíhala pouze zpětně. Když jsem navíc vyzkoušel podobně zaměřené aplikace v Android Marketu, zjistil jsem, že neexistuje žádná aplikace, která by uměla porovnávat dvě trasy online. Tudíž, že by nám oznamovala, jestli jsme ve stejném bodě jako při minulé trase pomalejší nebo rychlejší, případně jaký je rozdíl času nebo vzdálenosti mezi oběma trasami. Zkrátka by nám simulovala jakéhosi sparing partnera.

Hlavní požadavky na funkčnost byly tedy následující:

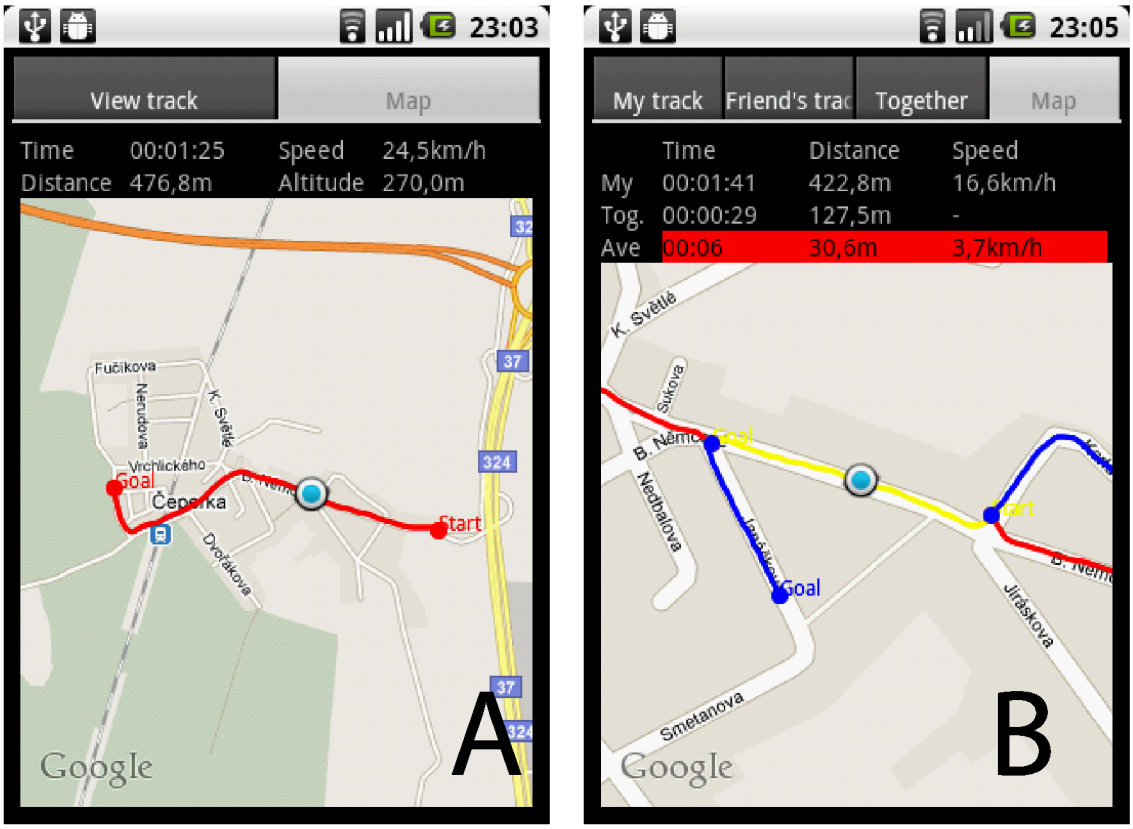
- záznam trasy se zobrazením podrobných statistik o ní a jejich přehledné vykreslení,
- při začátku nové trasy možnost porovnání se zvolenou trasou a zobrazení jejich výkonnostních rozdílů.

Vedlejší požadavky nejsou tolik důležité a mají za úkol pouze zjednodušení práce s aplikací nebo její rozšíření (zde jsou uvedeny implementované doplňky, možnosti dalších rozšíření najdete na konci práce v kapitole 6):

- zvukové upozornění o aktuální rychlosti, čase a vzdálenosti nebo jejich rozdíly oproti zvolené trase,
- export trasy do standardizovaného formátu,
- záznam trasy podle různých kategorií, které umožňují nastavení času pro obnovu signálu GPS, přesnost zaznamenané trasy...

Zobrazení statistik je rozděleno do dvou obrazovek, kde se v první ukazují kompletní statistiky o celé trase a ve druhé, která je popsána níže, lze vidět pouze vybrané údaje o jednotlivých částech trasy.

Při návrhu zobrazení statistik pro jednotlivou trasu jsem usoudil, že bude nejlepší ji vykreslit do mapy a při kliknutí na každý zaznamenaný bod oznámit, jaké v něm byly



Obrázek 3.1: Zobrazení statistik pro jednotlivé body v trase

jednotlivé údaje. Pro přehlednost a smysluplnost jsem se rozhodl uvést aktuální čas, vzdálenost, rychlost a nadmořskou výšku v daném bodě, jak lze ostatně vidět na obrázku 3.1 část A, kde modrý bod znázorňuje aktuální pozici.

Pro zobrazení statistik u dvou tras se předchozí koncept hodil ještě více, jelikož nám umožňuje přehledně zobrazit společné úseky tras a v nich jednotlivé rozdíly. Pokud jsou pozadí textových polí v červené barvě, znamená to, že jsme v naší aktuální trase o uvedený čas pomalejší, danou vzdálenost pozadu a máme o zobrazenou hodnotu nižší rychlost. Jak je vidět na obrázku 3.1 část B, trasa přítele se zobrazuje modrou barvou, naše trasa barvou červenou a společné úseky jsou vykresleny barvou žlutou. Ještě zde přibyly statistiky pro délku a čas společné trasy.

3.3 Záznam trasy

Při zpracování této funkčnosti bylo potřeba zohlednit mnoho faktorů. Jednalo se především o řešení problému s přesností zaznamenané trasy, výdrží baterie a výpočtu a zobrazení jednotlivých statistik u trasy.

3.3.1 Přesnost zaznamenané trasy

Jak již bylo v části o GPS uvedeno, data o poloze nejsou úplně přesná a v závislosti na okolních podmínkách se mohou oproti skutečné poloze lišit o 2–150 metrů. Mezi podmínky

ovlivňující přesnost určené polohy patří zejména všechny situace, které mají vliv na přímý výhled k satelitům. Jedná se například o hustotu zástavby nebo tloušťku střechy automobilu. Mezi další faktory můžeme též zahrnout typ použité GPS a její firmware. Pro představu bych rád uvedl následující příklad. Když jsem začal vyvíjet tuto aplikaci s v té době nejvybavenějším telefonem na trhu *Samsung Galyxy S*, pohybovala se nejlepší přesnost okolo patnácti metrů, avšak doba k prvnímu určení polohy dosahovala minimálně 50 sekund. Postupem času však vznikly od uživatelů různé patche, které danou přesnost i rychlost GPS fixu značně zlepšily.

Problém nepřesnosti je tedy určitě potřeba řešit. Když si představíme, že bychom třeba při běhu získali polohu našeho místa o 100 metrů jinde, určitě by nám to značně zkreslilo výslednou cestu. Dalším problémem i při slušné přesnosti dat z GPS může být jejich neustálá změna, například necháme-li telefon ležet po dobu 10 minut na jednom místě a budeme zaznamenávat trasu, kterou urazil, dostaneme velmi nepřesné výsledky. Jako logickou hodnotu vzdálenosti bychom zřejmě očekávali nula metrů, avšak právě díky datům z GPS, která mění polohu třeba i pouze o centimetry v každou chvíli jejich přijetí, získáváme nakonec hodnoty v řádech až desítek metrů. Toto je určitě chování, které se nedá akceptovat a v následujících odstavcích bude uvedeno, jak mu lze zabránit.

V bakalářské práci [12] zabývající se záznamem trasy byl tento problém řešen pomocí *rádiusu*, což je označení pro poloměr kružnice se středem v naposledy zaznamenané poloze. Algoritmus funguje následujícím způsobem: od posledně zaznamenaného bodu změří vzdálenost k aktuálnímu bodu z GPS, a ta pokud je menší nežli předem určená vzdálenost `R (radius)`, považuje aktuální bod za shodný s posledně zaznamenaným. Jak autor práce uvedl a jak z popsaného postupu vyplývá, velmi důležitá je hodnota optimálního `R`.

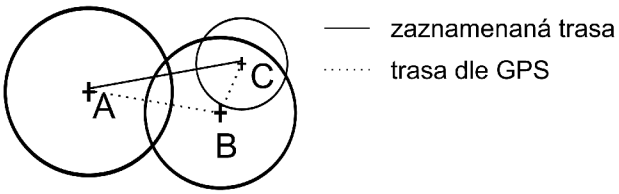
„Pokud je hodnota příliš malá, může se stát, že výsledná trasa bude stále delší než reálná. Pokud naopak bude tato hodnota příliš vysoká, lze očekávat vyfiltrování i efektivních trasových bodů a zaznamenaná trasa bude v konečném důsledku kratší než trasa reálná. Zkrácení trasy se nejvíce projeví zejména v ostrých zatáčkách, které mohou být vyhlazeny či dokonce úplně vyfiltrovány.“ [12]

Autor v práci dospěl po experimentálním testování k optimální hodnotě 10 metrů. Ukázalo se však, že dlouhodobé ideální nastavení této hodnoty není možné, jelikož závisí opět na okolních podmínkách a typu přijímače. Proto jsem se tedy rozhodl zmíněný algoritmus zčásti modifikovat. Tato úprava není nikterak složitá a její hlavní výhodou je to, že umožňuje automatické nastavení radiusu v závislosti na okolí.

Jak tedy algoritmus nyní funguje? S každou získanou polohou obdržíme i údaj o její přesnosti, který se může pohybovat přibližně v rozsahu 1–200 metrů. Tato přesnost nám určuje poloměr kružnice, ve kterém se může naše skutečná poloha nacházet. Na obrázku 3.2 jsou naše obdržené pozice, jež jsou středy kružnic, které zobrazují přesnost dané polohy. Pokud je vzdálenost posledně zaznamenaného bodu **A** a současného bodu z GPS **B** menší nežli součet jejich přesností, nastává velká pravděpodobnost, že by si body mohly být podobné, tudíž nedochází k záznamu aktuálního bodu. Naopak, když vezmeme v úvahu opět posledně zaznamenaný bod **A** a jeho vzdálenost k aktuálně obdrženému bodu z GPS **C**, zjistíme, že je větší nežli součet jejich přesností, a je tedy jisté, že jsme se pohnuli z místa a změna polohy není způsobena chybou v GPS.

3.3.2 Výdrž baterie a přesnost získané polohy

Činnost GPS pro svůj běh potřebuje velmi mnoho energie, a když si uvědomíme, že telefony s dotykovými displeji a operačními systémy nemají obecně závratnou výdrž baterie, máme



Obrázek 3.2: Filtrování trasových bodů

tu velký problém. Při testování bylo zjištěno, že při obnově signálu z GPS každou vteřinu nám baterie i se zhasnutým displejem vydrží přibližně necelých 6 hodin (testováno na zařízení G1), což je velmi málo.

Toto lze řešit různou dobou obnovy GPS, která je nastavitelná v kategoriích naší aplikace. Její hodnota se liší především podle prováděné činnosti a také podle toho, jakou kvalitu záznamu vyžadujeme. Uvedu zde malý příklad: pokud si budeme zaznamenávat trasu při chůzi, je určitě obnova každou sekundu zbytečná, jelikož se za tu dobu nepohneme o tolik metrů, aby nám to trasu značně zkreslilo. Naopak při jízdě autem v serpentinách a obnově signálu po 60-ti sekundách by se zaznamenaná trasa od absolvované zřejmě velmi lišila.

Jak bylo při testech zjištěno, s obnovou signálu souvisí často také přesnost získané polohy, a to vztahem, čím více obnovujeme signál, tím je obdržená poloha přesnější. Vzhledem k tomu, že si přejeme z GPS dostávat co nejpřesnější informace a zároveň šetřit baterii, je nutné zvolit mezi těmito dvěma cíli určitý kompromis. Ten spočívá v tom, že obnova signálu bude probíhat co nejčastěji až do té doby, než se získá požadovaná přesnost (opět nastavitelná v kategoriích naší aplikace). Poté se obnova zastaví, s pozicí se bude dále pracovat a po uplynutí doby pro obnovu GPS se začne opět získávat GPS signál.

3.3.3 Výpadky signálu GPS a další funkce

Ve výše zmíněné bakalářské práci autor řešil výpadky GPS signálu a dlouhodobé stání na místě. Já jsem se však rozhodl po otestování většiny podobných aplikací tyto funkce navrhnout jinak, a to hlavně z důvodů problémového nastavení parametrů pro očekávané chování daných funkcí a předpokladu přirozené inteligence uživatele. Například při neočekávaném stání na místě (semafory) by se trasa mohla předčasně ukončit. Proto je z mého pohledu jednodušší, když si uživatel záznam trasy sám ukončí.

Výpadky GPS signálu vznikají především uvnitř nebo v blízkosti budov, v údolích, v lese nebo třeba i v některých dopravních prostředcích. Uživatel může nastavit dobu, po které se při dlouhodobějším výpadku zobrazí upozornění, že nejsou dostupná data o jeho pozici a záleží pouze na uživateli, jestli se rozhodne zaznamenání trasy vypnout nebo naopak tuší, že se podmínky pro příjem signálu za chvíli zlepší a nechá záznam dále pokračovat.

Do návrhu byla zahrnuta i možnost pozastavení zaznamenávání trasy, což znamenalo, že uživatel mohl záznam trasy přerušit a později na něj navázat. Užitečnost této funkce se nejvíce projeví při běhu ve městě, kdy uživatel může například při stání na semaforech pozastavit záznam a při dalším rozběhnutí na něj opět navázat. V případě jízdy autem lze odstranit dlouhé pauzy při tankování paliva atd. Ve výsledné trase se tedy zobrazí její čistý čas, což je rozhodně užitečné. Tato funkce však způsobovala zbytečnou složitost při porovnávání dvou tras, takže do výsledné aplikace nebyla zahrnuta. Jednalo se však pouze

o upřednostnění důležitější funkce a je pravděpodobné, že se v dalších verzích aplikace opět objeví.

3.3.4 Statistiky

Při záznamu trasy nebo jejím zpětném prohlížení máme na výběr spoustu statistik. Zde si uvedeme jejich přehled, co přesně znamenají a u některých i jejich způsob výpočtu.

- **Čas** - při zaznamenávání trasy se zobrazuje aktuální čas a při jejím zpětném prohlížení celková doba trvání.
- **Vzdálenost** - počet metrů od začátku trasy. Při překročení hranice 10 000 metrů se začne zobrazovat v kilometrech s jedním desetinným místem.
- **Aktuální rychlost** - zobrazuje aktuální rychlost v kilometrech za hodinu. Bylo možné využít argument `speed` z GPS, ale ten nebere v potaz filtrování bodů. Počítá se tedy jako $(\Delta d/\Delta t)$, kde je Δ vždy rozdílem aktuální hodnoty a naposledy zaznamenané hodnoty.
- **Aktuální nadmořská výška**
- **Průměrná rychlost** - je udávána opět v km/h a počítá se jako (d/t), kde d je celková vzdálenost trasy a t znamená celkový čas trasy.
- **Průměrná nadmořská výška** - $\sum_{i=0}^{n} alt_i/n$, kde alt je nadmořská výška.
- **Maximální/minimální nadmořská výška**
- **Spálené kalorie** - čas trasy násobený koeficientem pro určitou fyzickou zátěž. Hodnota koeficientu je uložena v nastavení kategorie.
- **Maximální rychlost**
- **Délka klesání/stoupání** - $\sum_{i=0}^{n} \Delta d$, kde je nadmořská výška aktuálního bodu menší/větší nežli předchozího bodu.
- **Čas klesání/stoupání** - $\sum_{i=0}^{n} \Delta t$, kde je nadmořská výška aktuálního bodu menší/větší nežli předchozího bodu.

3.4 Porovnávání dvou tras

Tato funkce by měla především sloužit pro porovnání našeho aktuálního výkonu s nějakým výkonem předchozím, abychom zjistili, zda jsme se na dané trase zlepšili, či naopak zhoršili. Podpora online porovnání je zde dostupná hlavně z toho důvodu, abychom při absolvování trasy věděli, jaký je náš aktuální rozdíl oproti předešlé trase a mohli na něj odpovídajícím způsobem reagovat. Jako relevantní statistiky pro srovnání výkonů se jeví především rozdíl vzdálenosti, času a průměrné rychlosti ve společném bodě. Bylo by též vhodné nějakým přehledným způsobem zobrazit, jestli se náš výkon aktuálně zlepšuje nebo naopak

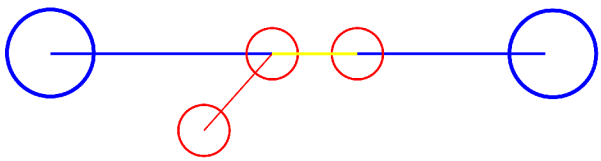
zhoršuje. K tomu nám dopomůže rozdíl aktuálních rychlostí obou tras. Pro naši představu o podobnosti společné trasy by se hodila i informace o času a vzdálenosti zaznamenaných ve společných úsecích.

Důležité zde bude, jak si vymezíme pojem společná trasa, respektive úsek. Pro společný úsek by mělo platit, že body z jedné trasy (aktuální, naše) budou ležet přibližně na stejné cestě jako body z jiné trasy (předchozí, přítelova). Slovem přibližně je myšlena největší vzdálenost v metrech nastavitelná v kategorii, o kterou se mohou jednotlivé trasy vzájemně od sebe nacházet. U definice společné trasy můžeme vyjít ze dvou situací, které je opět možné nastavit:

- společná trasa bude poslední společný úsek - hodí se pro téměř identické trasy,
- společná trasa se bude skládat ze všech předchozích společných úseků - umožňuje uživateli od společné trasy odbočovat, a tím měnit její úseky.

Pokud si vezmeme nejjednodušší způsob porovnání dvou tras, což je měření vzdáleností mezi body z jedné trasy (aktuální) a body z druhé trasy (předchozí), zjistíme, že má pro online porovnávání mnoho nevýhod, které budou později řešeny:

- Velká náročnost - pokud se při získání nového aktuálního bodu bude muset porovnávat jeho vzdálenost se všemi ostatními body z předchozí trasy, zabere to značný čas, zvláště pokud si uvědomíme, že trasy mohou obsahovat tisíce bodů.
- Různá hustota záznamů - v případě, že předchozí trasa bude mít zaznamenávání bodů po 500 metrech, aktuální po 20 metrech a maximální možná odchylka bude 15 metrů, tak ačkoliv jsou trasy stejné, nebudou za ně po záznamu prvních 24 bodů považovány. Obrázek 3.3 nám ukazuje podobnou situaci. Modrá barva značí přítelovu, červená naši trasu a společný úsek je znázorněn žlutě. V tomto případě však nebude označen, jelikož naše body neleží v okruhu tolerance (modré kružnice) přítelových bodů (červené kružnice určují přesnost zaznamenané polohy).



Obrázek 3.3: Různá hustota záznamů

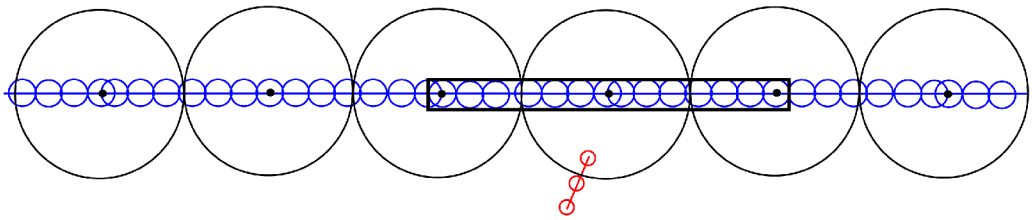
3.4.1 Velká náročnost

Snížení náročnosti porovnávání vzdáleností mezi všemi body v přítelově trase a naším aktuálním bodem dosáhneme tím, že si z přítelovy trasy vybereme několik základních bodů `basePoints` a budeme měřit a porovnávat vzdálenost těchto bodů k aktuálně zaznamenanému bodu z GPS. Výběr `basePoints` provedeme následujícím způsobem. Nejdříve určíme pomocnou vzdálenost `helpDistance`, která slouží k rozeznání, které body budeme z přítelovy trasy vybírat, tím způsobem, že vydělíme celkovou vzdálenost trasy deseti. Další vzdáleností kterou potřebujeme zjistit, je nová maximální odchylka `radiusBasePoint` od aktuálního bodu z GPS k bodům z `basePoints`. První pozici trasy uložíme do `basePoints` a poté postupujeme dle následujícího algoritmu:

```
1. čti další pozici v přítelově trase
2. zjisti actualDistance, což je vzdálenost od posledně zaznamenaného
      basePoints k aktuální načtené pozici
3. if (actualDistance < helpDistance)
      // bodů je stále hodně
      jdi na 1.
   else
      ulož přítelovu načtenou pozici do basePoints
      if (radiusBasePoint < actualDistance)
         radiusBasePoint = actualDistance
      jdi na 1.
```

Do seznamu `basePoints` ještě uložíme přítelův poslední bod. Nyní máme přibližně desetinu kamarádových bodů, pro které budeme zjišťovat, jestli je jejich vzdálenost od aktuálního zaznamenaného bodu z GPS menší nežli `radiusBasePoint/2` (nová maximální odchylka společné trasy). Pokud tato situace nastane, znamená to, že jsme se přiblížili k přítelově trase (viz obrázek 3.4).

Algoritmus pokračuje následovně: při zjištění přiblížení našeho aktuálního bodu (červená barva) k základnímu bodu přítele (černá barva) načte všechny reálné pozice (černý obdélník) od předchozího základního bodu až po další základní bod. V tomto intervalu máme tedy všechny reálné polohy, ke kterým se může následující aktuální pozice z GPS přiblížit. Pro rozpoznání začátku stejné trasy nám bude stačit již pouze to, aby vzdálenost aktuálně zaznamenaného bodu s jakoukoliv pozicí z předchozího intervalu byla menší nežli nastavená odchylka (červený bod musí ležet uvnitř modré kružnice).



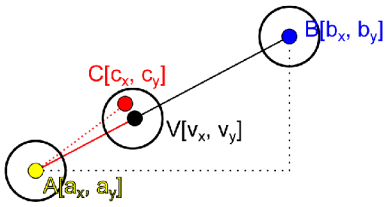
Obrázek 3.4: Přiblížení k základním bodům

3.4.2 Různá hustota záznamů

Předpokládejme, že jsme pomocí předchozího algoritmu zjistili první společný bod a že záznam trasy přítele je hustější nežli náš. Nyní získáme naši novou pozici z GPS a hledáme k ní odpovídající bod z přítelovy trasy, kterou procházíme od posledního společného bodu do té doby, dokud bude vzdálenost pozice z GPS a porovnávaného bodu stále klesat. Při skončení tohoto cyklu by měl být vybrán bod z přítelovy trasy, který je aktuální poloze nejblíže. Avšak vzdálenost těchto dvou bodů nemusí být v závislosti na hustotě záznamu menší nežli vyžadovaná odchylka, a přesto mohou ležet na stejné trase. Obdobná situace může nastat u opačné hustoty záznamů, to jest pokud bude přítelova trasa obsahovat méně bodů.

Tuto situaci jsem se rozhodl řešit pomocí takzvaných **virtuálních bodů**, kdy je vždy v přítelově trase vytvořena pozice, která na dané cestě leží, a pokud je aktuálně zaznamenaný bod z GPS také na stejné trase, je od něj vzdálen méně, než požaduje odchylka.

Názorné sestavení jednoho virtuálního bodu `V` je zjednodušeně zobrazeno na obrázku 3.5, kde představují body `A` poslední společný bod, `B` nejbližší bod v přítelově trase a `C` aktuálně zaznamenaný bod z GPS.



Obrázek 3.5: Vytvoření virtuálního bodu

Pro vytvoření nového bodu je nutné nejprve zjistit poměr (*coeficient*) vzdálenosti `AC` ku `AB`. Pro nový bod `V` platí následující souřadnice

$$v_x = a_x + (b_x - a_x) \cdot coeficient$$

$$v_y = a_y + (b_y - a_y) \cdot coeficient$$

Hodnoty pro délku a čas v bodě se určují stejným způsobem.

Nyní zjistíme vzdálenost aktuálně získané pozice k virtuálnímu bodu, a pokud je menší nežli nastavená odchylka (černá kružnice), porovnáme jejich hodnoty a uživateli zobrazíme statistiky.

3.4.3 Statistiky dvou tras

Při běhu s přítelem se nám kromě zaznamenávaných a zobrazovaných statistik pro jednotlivou trasu porovnávají naše vzájemné statistiky a zobrazuje se vzdálenost a čas ve společných bodech. Hodnoty, které budou počítány, závisí na nastavení společné trasy. Jestli za ni budeme považovat pouze poslední společný úsek nebo všechny společné úseky. Zde je přehled celkových statistik:

- **Společný čas** - doba od prvního společného bodu k nynějšímu společnému bodu.
- **Společná vzdálenost** - vzdálenost od prvního společného bodu k aktuálnímu společnému bodu.

Nyní si ukážeme statistiky, které nám zobrazí, o kolik sekund nebo metrů jsme v daném bodě pozadu (napřed) oproti přítelovi. Zjistíme i průměrnou a aktuální rychlost, kterou na přítele ztrácíme (získáváme). Pokud jsme v daném místě lepší nežli přítel, zobrazí se pole zelenou barvou, v opačném případě barvou červenou.

- **Průměrná rychlost** - rozdíl průměrných rychlostí počítaných od prvních společných bodů k mému (přítelovu) aktuálnímu společnému bodu.
- **Vzdálenost** - rozdíl společné vzdálenosti a vzdálenosti přítele, kterou získáme vynásobením jeho průměrné rychlosti v jeho posledním společném bodě se společným časem.

- **Čas** - rozdíl společného času a času přítele, který vypočítáme vydělením společné vzdálenosti přítelovou průměrnou rychlostí získanou v jeho posledním společném bodě.
- **Aktuální rychlost** - slouží především k aktuálnímu zobrazení, jestli se oproti příteli zlepšujeme nebo zhoršujeme. Počítá se od předposledních k posledním společným bodům.

Statistiky aktuálního času a vzdálenosti nebyly po testování zahrnuty z toho důvodu, že počet zobrazovaných údajů byl velmi nepřehledný. To, že se aktuálně zhoršujeme (zlepšujeme) lze rozpoznat podle klesajících (stoupajících) společných rozdílových hodnot průměrné rychlosti, celkového času a vzdálenosti. Abychom nemuseli displej sledovat delší dobu, je zde právě uvedena hodnota rozdílu aktuální rychlosti, která nám oznámí náš aktuální stav.

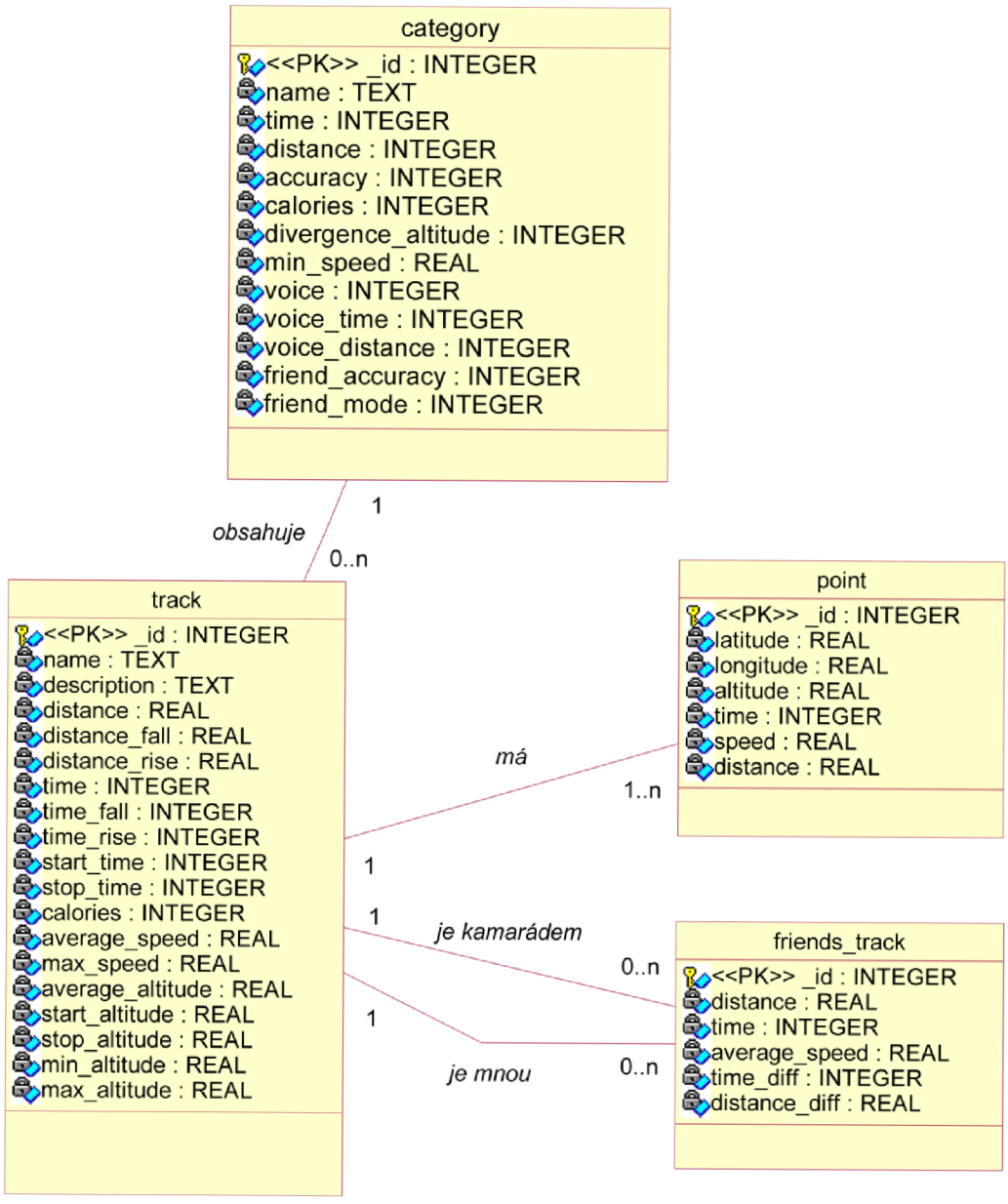
3.5 Uložení dat

Pokud chceme v Androidu uložit dlouhodobá data, máme na výběr z uložení do databáze nebo uložení do samostatných souborů. Při ukládání záznamů trasy do souborů by byla výhoda v tom, že bychom mohli data rovnou ukládat ve formátu *GPX*, a ty dále zpracovávat v PC, avšak jejich následná analýza v telefonu by zabrala značný čas, jelikož by se musely soubory parsovat.

Proto jsem se rozhodl data o trasách, ale i kategoriích ukládat do databáze, jejíž schéma popisuje uvedený *ER diagram* 3.6. Entita `track` obsahuje záznamy o jednotlivých trasách, které jsou nahrány dle nastavení určité kategorie `category` a obsahují jednotlivé body `point`. Pokud absolvujeme trasu podle jiné trasy, hodnoty naší trasy se uloží předchozím způsobem a společné hodnoty budou uloženy do entity `friends_track`. Zde se nacházejí také odkazy na obě absolvované trasy.

U entity `point` bych ještě zmínil, že by se v ní nemusela nacházet data o rychlosti vzdálenosti, jelikož se dají vypočítat. Zde jsou však uložena z toho důvodu, abychom při následné analýze trasy zbytečně neztráceli čas.

Jediná nevýhoda tohoto způsobu uložení tras spočívá v tom, že data jsou dostupná pouze v zařízení. Pokud s nimi budeme chtít dále pracovat, například v PC, je nutné je exportovat do souboru formátu GPX, jehož název bude časové razítko zaznamenané trasy ve formátu `yyyy-MM-dd_hh_mm`.



Obrázek 3.6: ER diagram

Kapitola 4

Implementace

V této kapitole bude popsána implementační část a problémy, které mě při ní potkaly. Zmíním se o komponentách a způsobu jejich využití, o rozhraních k práci s GPS a mapách a o práci s hlasovým upozorněním. Veškeré popisy vytvořených tříd obsahuje programová dokumentace k aplikaci, kterou naleznete na přiloženém CD.

4.1 Uživatelské rozhraní

Jak bylo v kapitole o uživatelském rozhraní v Androidu popsáno, tvoří ho zejména Aktivity a programuje se pomocí XML layoutů, které obsahují uspořádané widgety. V této aplikaci je hlavní obrazovkou aktivita `SportWithFriend` obsahující pouze čtyři tlačítka, která slouží jako rozcestník pro další funkce.

Z důvodu potřeby rozdílné implementace získávání dat při absolvování trasy a jejího následného zobrazení se v celém grafickém rozhraní uplatňuje systém dědičnosti. Tu bylo také vhodné použít pro rozšíření některých nastavení a zobrazení trasy pro běh s přítelem. V následujícím textu, pokud nebude uvedeno jinak, jsou popsány rodičovské třídy.

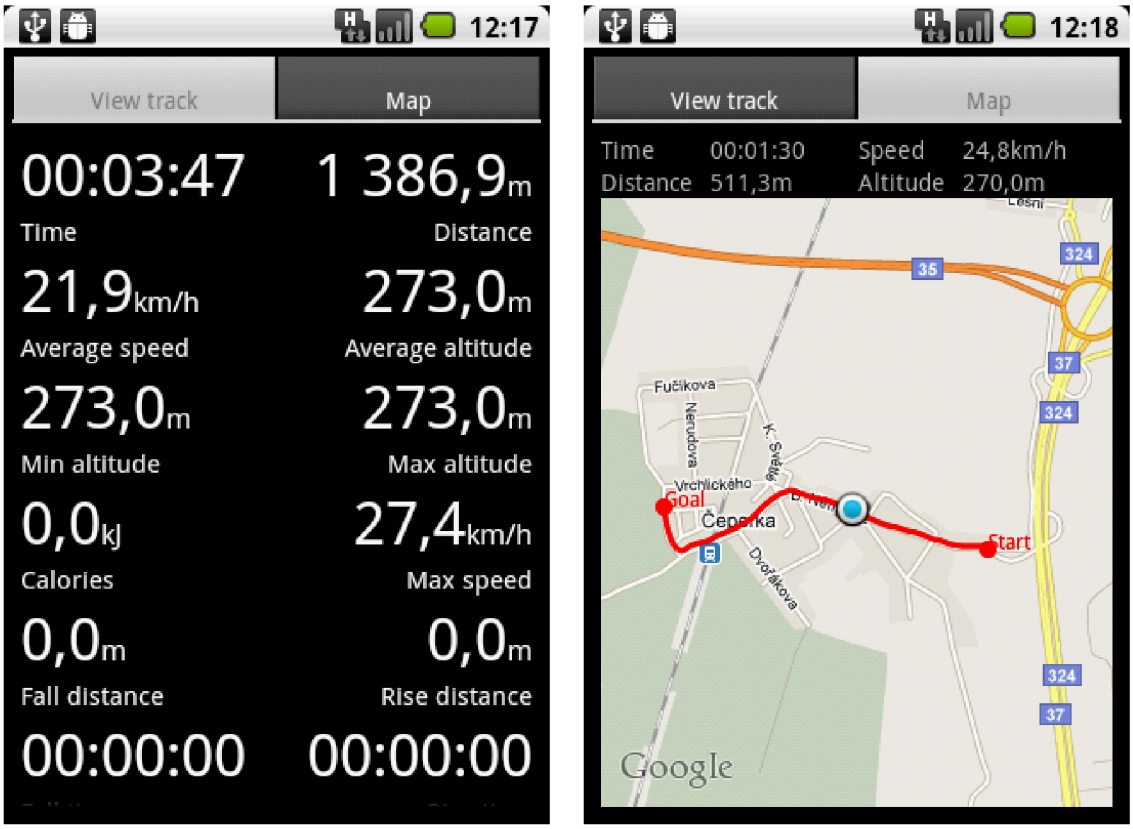
Pro nastavení informací o trase slouží třída `Preparation`, kde je možné zadat název zaznamenávané trasy, její popis a především kategorii, podle které bude záznam probíhat. Třída `Category` pomocí třídy z Android API `ListAdapter` zobrazuje z databáze seznam kategorií, které můžeme vybírat, mazat nebo dále editovat. K úpravě jejich hodnot slouží třída `CategoryPreferences`, jež zobrazuje pro Android standardizovanou obrazovku nastavení.

Po spuštění záznamu trasy nebo jejím následném zobrazení máme k dispozici dvě záložky, jak ukazuje obrázek 4.1. Jedná se o třídu `TrackClass` pro vypsání celkových statistik a třídu `MapClass` pro zobrazení informací v mapě 4.6.

Pokud si budeme chtít prohlédnout absolvované trasy, poslouží nám k tomu třída `TrackTab`, která zobrazí v jedné záložce seznam všech námi absolvovaných tras a v druhé trasy absolvované společně s přítelem.

4.2 Databáze

V podkapitole 3 bylo uvedeno, že ukládání dat bude zprostředkovávat databáze v zařízení. Z tohoto důvodu je nutné naprogramovat vlastní Content Provider (viz. 2.4.1), který nejdříve vytvoří celou databázi i s její strukturou a poté nám umožní přidávat, odebírat, měnit nebo dotazovat se na data. Ve třídě `SportWithFriendProviderMetaData` se nacházejí názvy tabulek, jejich *URI* (což jsou vlastně jejich identifikátory pro Content Provider)



Obrázek 4.1: Zobrazení informací o trase

a názvy jednotlivých sloupců tabulek, jež později budeme využívat při dotazování nebo zápisu záznamů do databáze. Třída `SportWithFriendProvider` obsahuje samotnou implementaci Content Providera. Je též nutné přidat informaci o vlastním Content Providerovi do manifestu.

Při vkládání dat do databáze je tedy vždy nutné uvést, přes jakého Content Providera se vložení provede, a poté pouze zavolat metodu `insert(PointTableMetaData.CONTENT_URI, pointValues)` s příslušnými argumenty. První argument je právě výše zmíněné URI, které nám jednoznačně identifikuje cílovou tabulku. Druhý argument obsahuje objekt typu `ContentValues`, což je třída obsahující data určená k uložení v tabulce. V tomto případě se jedná o data pro jednu polohu.

Dotazování dat probíhá opět skrze Content Providera pomocí příkazu `query (uri, projection, selection, selectionArgs, sortOrder)`, kde `uri` identifikuje tabulku, `projection` obsahuje seznam navrácených sloupců, `selection` nahrazuje *SQL* příkaz `WHERE`, `selectionArgs` jeho argumenty a `sortOrder` určuje, jak budou záznamy seřazeny.

Metoda `query()` navrací objekt typu `Cursor`, který obsahuje všechny požadované záznamy, v nichž se pomocí různých funkcí můžeme pohybovat a z jednotlivých záznamů získávat data.

4.3 Služby a vzájemná komunikace

Služby poskytují v OS Android prostředek pro dlouhotrvající operace, které běží na pozadí. Už podle této definice by sem určitě měl být zařazen dlouhodobý záznam bodů z GPS. Tuto službu implementuje třída `TrackRecording`, která podle nastaveného intervalu periodicky získává údaje o poloze a následně je i vyhodnocuje a ukládá do databáze. Služba se spouští hned po začátku záznamu trasy a má nejdříve za úkol pouze zjistit dostupnost GPS signálu. Pokud je signál dostupný, informuje o tom pomocí odeslané zprávy, kterou přes potomka třídy `BroadcastReceiver` `GPSFixReceiver` zachytí třída `Preparation`, a až poté dojde k zobrazení statistik o trase a jejich zápisu do databáze. Nyní služba `TrackRecording` pravidelně získává, vyhodnocuje a v případě správné přesnosti obdržených zeměpisných poloh i zaznamenává, odesílá a zapisuje data do databáze. Jak probíhá práce s GPS, je vysvětleno v následující části.

4.4 Location API

OS Android poskytuje v základním API tzv. *Location API*, které umožňuje snadné získávání poloh z GPS, převody jejich souřadnic na reálné adresy nebo lokalizaci podle telefonní sítě. V dalších odstavcích si ukážeme, jak se jednotlivé třídy používají a jak byly využity přímo v aplikaci.

4.4.1 LocationManager

U této třídy máme na výběr, zda chceme údaje o poloze získávat pomocí GPS nebo pomocí vysílaného signálu základních stanic mobilních operátorů. Na tomto nastavení také záleží, jaké povolení je nutné přidat do manifestu. Pro práci s GPS musí aplikace disponovat oprávněním `android.permission.ACCESS_FINE_LOCATION`. Následující kód zpřístupňuje získání polohy z GPS každých 5 vteřin:

```
LocationManager locationManager = (LocationManager)
   getSystemService(LOCATION_SERVICE);
GPSListener gpsListener = new GPSListener();
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000,
   0, gpsListener);
```

Pokud se poloha změní, je zavolána přetížená funkce `onLocationChanged(Location location)` třídy `GPSListener`, která rozšiřuje třídu `LocationListener`. Parametr `location` obsahuje právě získanou pozici, o které vlastní metoda `isLocationOK()` dle návrhu 3.3.1 zjišťuje, zda má být zaznamenána. Pokud ano, volají se postupně další funkce, které vypočítají statistiky a následně je uloží do databáze a zašlou ve formě zprávy aktivitám, jež zobrazují aktuální stav.

Při testování však bylo zjištěno, že se metoda `requestLocationUpdates()` chová při hodnotě argumentu času obnovy větší než 1 sekunda velmi zvláštně. Zjevné to bylo při nastavení této hodnoty například na 30 vteřin, kdy se po uplynutí této doby zavolala metoda `onLocationChanged(Location location)` desetkrát, a teprve poté došlo k přerušení získávání polohy. Po nahlédnutí do zdrojových kódů API jsem zjistil, proč se tato funkce takto chová. Jak již bylo uvedeno v podkapitole 3.3.2, čas obnovy nejspíše souvisí s přesností získané polohy. Proto pokud GPS přijímač začne určovat polohu po delším čase, snaží se ji získat s co nejlepší přesností, a tak signál obnoví desetkrát. Jelikož toto chování dělalo

značné zmatky v zaznamenaném souboru, rozhodl jsem se implementovat vlastní vlákno pro získávání signálu.

Při spuštění služby `TrackRecording` se zároveň spustí vlákno `UpdateGPS`, které zapne obnovu GPS signálu v co nejkratší možné době a uspí se na zadaný čas. Při změně polohy se ověří její přesnost, a pokud je větší nežli v kategorii nastavená, obnova GPS signálu se pozastaví až do opětovného vyvolání vláknem `UpdateGPS`.

4.5 Porovnání dvou tras

Porovnání dvou tras řeší třída `ComparsionPoints`, která jako parametry konstruktoru přijímá ID trasy přítele, odchylku vzdálenosti dvou tras, hodnotu určující způsob záznamu společné trasy a nakonec odkaz na Content Providera. Hned při vytvoření instance třídy jsou vytvořeny základní body trasy. Pro zjištění podobnosti dvou tras a jejich rozdílů slouží metoda `isSameTrace(Location actualLocation)`, která jako argument přijímá aktuální pozici nynější trasy. Dle míry jejich podobnosti vrací následující stavy:

- `IS_FIRST` - nalezen první společný bod.
- `IS_SAME` - naše trasa je stejná jako přítelova, můžeme tedy získávat rozdíly jednotlivých hodnot.
- `NOT_SAME` - trasy nejsou prozatím stejné.

4.6 Google Maps API

Práci s mapovými podklady umožňuje v OS Android externí knihovna s balíčkem `com.google.android.maps.MapView`. K té je však nutné získat z důvodů potvrzení licenčních podmínek klíč (viz [9]). Hlavním prvkem pro zobrazení a manipulaci s mapou je třída `MapView`, která získává data od služby Google Mapy a umožňuje měnit mapové a satelitní pohledy. Pro vykreslení jakéhokoliv prvku do mapy musíme rozšířit třídu `Overlay` a přetížit její metodu `draw()`. Ta převádí objekty typu `GeoPoint`, které obsahují geografickou polohu místa, na objekty typu `Point`, jež můžou být vykresleny. Všechny takto vytvořené objekty se přidávají jako vrstvy s průhledným pozadí do třídy `MapView` a po kliknutí na ně se volá metoda `onTap()`.

Pro vykreslení složitějších objektů a jejich dynamickou změnu slouží třída `ItemizedOverlay<OverlayItem>`, kde právě `OverlayItem` představuje jednu položku pro vykreslení. Zde je nutné oproti metodám z `Overlay` přetížit i funkce pro přidání, odebrání nebo zjištění počtu vykreslovaných bodů.

Třída `MapPoint` reprezentuje jeden bod na mapě a rozšiřuje výše uvedenou třídu `GeoPoint` o uložení informací o čase, vzdálenosti, nadmořské výšce a aktuální rychlosti současného bodu. Abychom však mohli daný bod vykreslit, musíme ještě rozšířit třídu `OverlayItem`. Nyní tuto třídu `MapOverlayItem` použijeme jako generický typ pro rozšíření třídy `ItemizedOverlay`, kde přetížíme výše uvedené metody, čímž získáme třídu `MapOverlay`. Objekt této třídy tedy reprezentuje aktuální trasu, do které podle druhu zobrazení pouze přidáváme jednotlivé body z databáze nebo z přijímaných zpráv.

4.6.1 Vykreslení dvou tras

Při sportu s přítelem se celá jeho trasa nejprve vykreslí do objektu `pointsOverlayList`, zatímco naše trasa je postupně vykreslována do objektu `myPointsOverlayList`. Pokud však nastane protnutí obou tras, body se přidávají místo do `myPointsOverlay` do `togetherOverlayList`, což je cesta, která představuje společný úsek. Z `pointsOverlayList` se také musí odebírat body, které byly označeny za společné, což se určuje pomocí funkcí z třídy `ComparsionPoints`.

4.7 Export GPX

Aby byly informace o trase dále přenositelné, musí aplikace umožňovat export do standardizovaného formátu GPX. Ten zajišťuje třída `ExportGPS`, která v parametru konstruktoru obdrží id exportované trasy. Poté přistoupí do databáze a zjistí o ní celkové údaje - čas a datum jejího záznamu, název a popis. Na základě data a času začátku zaznamenání trasy vytvoří ve složce `\sdcard\GPX` soubor s názvem ve formátu `yyyy-MM-dd_hh_mm.gpx`. Do něho se následně zapíše obecná hlavička pro tento typ formátu, kde se uvedou i souhrnné informace o trase. Poté se v cyklu čtou pro danou trasu všechny body z databáze a pomocí funkce `writePoint()` se do souboru zapisují zeměpisná šířka, zeměpisná délka, nadmořská výška a čas zaznamenání bodu. Po jejich zapsání se přidají pouze uzavírající elementy značící konec trasy a celého souboru, který následně zavřeme. Nyní máme ve složce GPX čitelné soubory, které můžeme pomocí správce souborů zkopírovat do PC nebo odeslat emailem.

4.8 Hlasová signalizace

Pokud jsou v telefonu nainstalována hlasová data, umožní nám Android funkce *Text-to-Speech* přehrávání napsaného textu v několika jazycích. Toho můžeme využít při pravidelném oznamování informací o právě zaznamenávané trase. Ve službě `TrackRecording` je tedy vytvořen objekt `voiceSpeech` typu `TextToSpeech`, který nám tuto funkčnost zpřístupňuje. Před jeho použitím však musíme nastavit jazyk, ve kterém bude syntéza řeči probíhat. Vzhledem k tomu, že čeština ještě není bohužel dostupná, musíme použít slovníky pro anglický jazyk. Nyní již skrze vlákno `voice` voláme funkci `sayInfo()`, která připraví textový řetězec obsahující informace o čase, délce a rychlosti trasy. Následně tento text pomocí objektu `voiceSpeech` i přehraje.

Kapitola 5

Testování

V této kapitole shrneme testování jednotlivých důležitých funkcí, které byly popsány v předchozích kapitolách. Uvedu zde i způsob a výsledky testování celé aplikace. Důležitým bodem je výčet zařízení a jejich verze OS, na kterých byla aplikace testována a vyvíjena: Android Emulator (2.1, 2.2), G1 (2.1, 2.2) a Samsung Galaxy S (2.1).

5.1 Testování jednotlivých funkcí

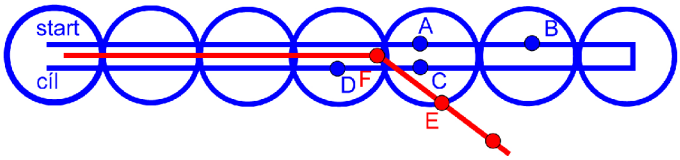
Po otestování důležitých funkcí bylo zjištěno, že všechny nefungují zrovna takovým způsobem, jaký byl očekáván. Některé z těchto problémů byly již popsány v kapitole návrhu, proto je zde pouze shrnu a poté se zaměřím na nově objevené chyby.

Největší čas při testování zabrala celková práce s GPS. Jak se ukázalo, *Location API* se při obnově signálu po delší době nechovala podle předpokladů, tudíž bylo implementováno vlastní vlákno pro periodické získávání pozice. Dalším nedostatkem je u třídy `Location` absence funkce pro měření vzdálenosti mezi dvěma pozicemi, které by zahrnovalo i jejich nadmořské výšky. Při měření krátkých vzdáleností mezi dvěma polohami totiž hraje větší roli rozdíl jejich nadmořských výšek nežli zakřivení Země. Vzhledem k tomu, že je pro většinu terénu rozdíl takto získaných vzdáleností minimální, nebyla vlastní funkce pro zjištění vzdálenosti dvou zeměpisných pozic zahrnující i nadmořskou výšku implementována. Dalším důvodem byl i fakt, že funkce `distanceTo` z třídy `Location` je značně optimalizována.

Při testování vlastních funkcí na porovnání dvou tras ve třídě `ComparsionPoints` se objevila situace, se kterou nebylo v návrhu počítáno. Jednalo se o části jedné trasy, které mají společné body, tj. trasy vzniklé při absolvování cesty tam a zpět. Pokud tuto trasu absolvujeme znovu a budeme si ji chtít porovnat s předchozí cestou, algoritmus bude fungovat v pořádku, jelikož začne body porovnávat od začátku trasy. Problém však nastává v případě, že bychom se k porovnávané cestě připojili v její druhé části. Vše zobrazuje obrázek 5.1, kde jsou modré kružnice opět maximální možné odchylky, od kterých se může společná trasa odvrátit. Zde totiž při zaznamenání bodu `E` dochází k procházení bodů přítelovy cesty od začátku a jako první společný bod v přítelově trase je označen bod `A` místo bodu `C`. Avšak při záznamu naší další pozice `F` už je vzdálenost mezi bodem `B` a `F` větší nežli maximální odchylka, a tudíž dochází k přerušení společné trasy. Tento problém aplikace neřeší a při zachování online porovnávání ani řešit nemůže, jelikož předem neví, jakým směrem bude naše cesta po bodu `E` pokračovat. Jako způsob řešení zde lze uvést:

- připojení se k přítelově trase ještě v její první půli,

- při doběhnutí do půli trasy záznam ukončit a při zpáteční cestě provést nový záznam,
- implementovat funkci, která by při online porovnávání používala dočasně offline porovnávání.

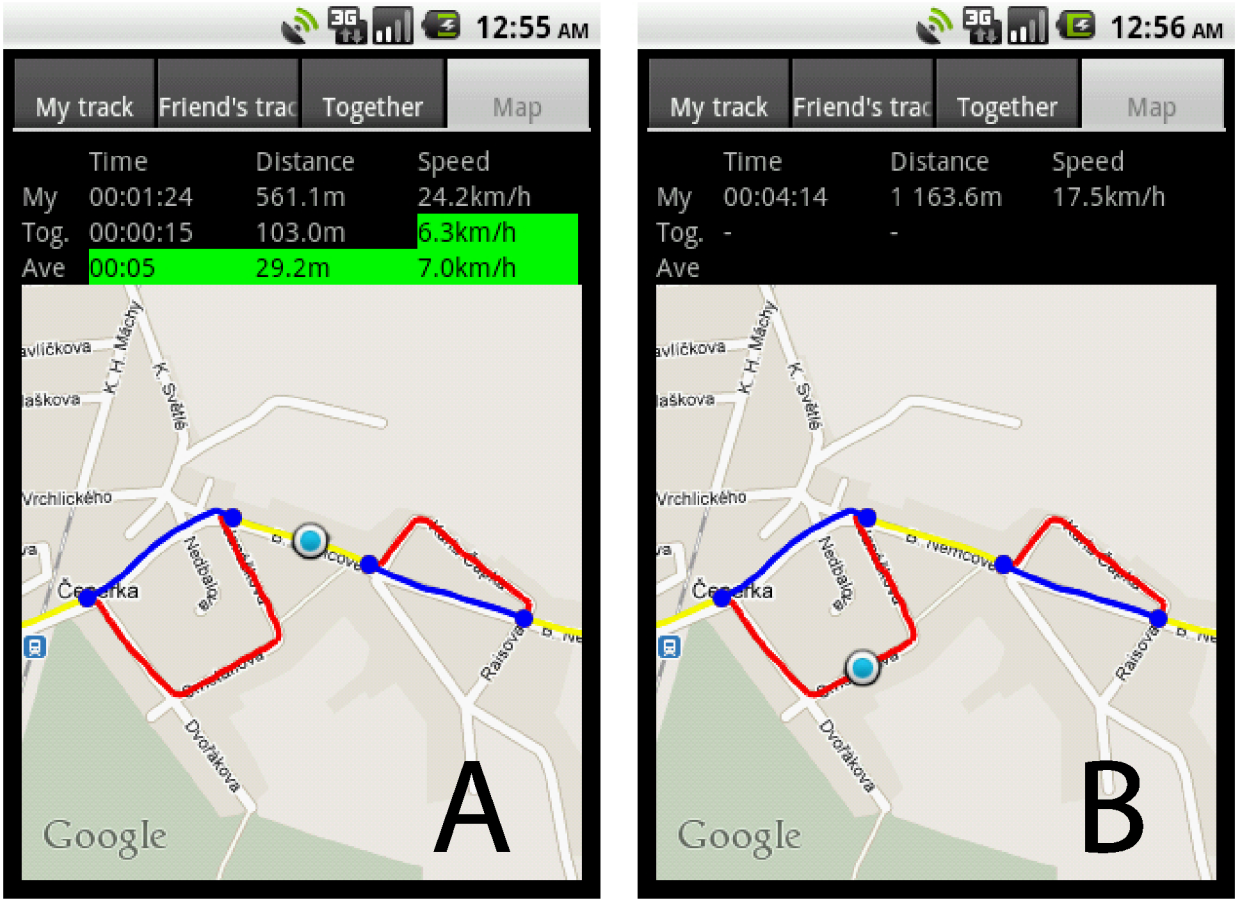


Obrázek 5.1: Problém „tam a zpět“

5.2 Testování celé aplikace

Testování záznamu jednotlivých tras probíhalo takřka od začátku implementace, a to od pouhého stání na místě až po stakilometrové cesty. Tento způsob testování spočíval v nahrávání trasy, při kterém docházelo k různým výjimečným situacím, jako byly výpadky signálu nebo uspání telefonu. Při těchto zkouškách nebyly až na přesnost zaznamenané trasy, která byla řešena v návrhové a implementační části, zjištěny žádné chyby.

Desítky pokusů byly také provedeny pro porovnávání dvou tras, avšak zde uvedu pouze jedno porovnání, kde jsou ve výsledcích zřetelné rozdíly. Vše nejvíce osvětlí obrázek 5.2. Červená cesta zde zobrazuje trasu přítele, která má body zaznamenány přibližně po dvou sekundách a průměrná rychlost je 16,6 km/h. Naopak naše cesta je barvy modré, zaznamenána byla po pěti sekundách a průměrná rychlost se blíží k 23 km/h. Společné úseky mají maximální možnou odchylku 15 metrů, vykresluje je žlutá barva a na obrázku v části A vidíme náš odpovídající náskok, což indikuje zelená barva pozadí. Pokud srovnáme průměrné rychlosti obou tras, jejich nynější rozdíl nám přibližně vychází na odpovídající hodnotu 7 km/h. Hodnota 6,3 km/h znamená, že se nyní ani oproti přítelovi nezhoršujeme, tudíž nemusíme mít obavu, že by nás prozatím dohnal. Část B obrázku zobrazuje pouze přítelův samostatný úsek, kdy nejsou statistiky pro společnou trasu zobrazeny. To samé platí při zobrazení detailů o naší aktuální trase. GPX soubory, které byly zaznamenány v tomto testu, najdete na přiloženém CD ve složce GPX.



Obrázek 5.2: Testovací trasy

Kapitola 6

Závěr

6.1 Přínosy práce

Cílem této práce bylo vytvoření aplikace pro mobilní operační systém Android, která by sloužila jako tréninkový deník zaznamenávající data z GPS a umožňovala jejich následnou analýzu. Aplikace Sport with friend toto zadání splnila a přidala navíc i spoustu užitečných funkcí, které se podle mého názoru doposud v mobilních telefonech na této platformě nevyskytovaly. K nejdůležitějším určitě patří podpora online porovnávání dvou tras, která zajišťuje uživatelům přehledné porovnání jejich aktuálního výkonu oproti výkonu předchozímu. Zde je jednoduché vysvětlení: pokud jsme oproti předchozí trase v určitém bodě pomalejší, aplikace nám zobrazí, jaký čas, vzdálenost nebo průměrnou rychlost ztrácíme oproti své předchozí trase, a umožní nám tak na to reagovat případným zrychlením.

Pro vypovídající zobrazení informací o různých částech trasy byla namísto tradičních grafů zobrazena mapa, která pouze neukazuje různé závislosti například času na výkonu, ale dokáže detailně popsat hodnotu našeho výkonu v určitém bodě. To je dle mého názoru užitečnější, jelikož si většina lidí spíše pamatuje konkrétní místo nežli čas. Jednou z nevýhod použití mapového zobrazení však může být nutné připojení na Internet, což se ale v dnešní době dostatku Wi-Fi Hotspotů a mobilního Internetu stává čím dál menším problémem. Aplikaci lze samozřejmě využívat i bez mapových podkladů, při tomto použití však slouží spíše jako obyčejné GPS zařízení zaznamenávající naši absolvovanou trasu.

6.2 Rozšíření aplikace

Samotná aplikace po otestování funguje spolehlivě, avšak k dokonalosti ji oproti dalším aplikacím schází především další propojení s různými sociálními sítěmi. V nejlepším případě přímo s nějakou oficiální webovou aplikací, která by umožňovala sdílení tras s dalšími uživateli, upload z telefonu přímo na web nebo zobrazení porovnaných tras v prohlížeči.

K samotným funkcím v mobilním zařízení by mohlo být rozhodně užitečné přidání podpory pro běh nebo jízdu na oválu, kde by se zaznamenávala jednotlivá kola a dále se analyzovala. Vzhledem k tomu, že je uživatelské rozhraní navrženo především pro funkčnost, mohla by některým uživatelům přijít vhod i možnost stylizace.

Příloha A

Obsah CD

- application - instalační soubory pro aplikaci Sport with friend
- GPX - složka obsahující vzorové trasy ve formátu gpx
- source - projekt pro IDE Eclipse
- text - zdrojové soubory pro LaTeX
- poster.pdf - plakát reprezentující aplikaci
- README - soubor podobný této příloze
- xsyruc00.pdf - text práce ve formátu pdf

Literatura

[1] *Activities – Android Developers* [online]. 2011-01-05 [cit. 2011-05-14]. Dostupné z WWW: <http://developer.android.com/guide/topics/fundamentals/activities.html>.

[2] *ADT Plugin for Eclipse – Android Developers* [online]. 2011-03-14 [cit. 2011-05-14]. Dostupné z WWW: <http://developer.android.com/sdk/eclipse-adt.html>.

[3] *Android Developers* [online]. 2011-05-13 [cit. 2011-05-17]. Dostupné z WWW: <http://developer.android.com/index.html>.

[4] *Android SDK – Android Developers* [online]. 2011-05-13 [cit. 2011-05-17]. Dostupné z WWW: <http://developer.android.com/sdk/index.html>.

[5] *Aplikace – Android Market* [online]. 2011-05-10 [cit. 2011-05-14]. Dostupné z WWW: <https://market.android.com>.

[6] *Best Java IDE :: Do more high-quality code in less time with IntelliJ IDEA.* [online]. 2010-02-19 [cit. 2011-05-14]. Dostupné z WWW: <http://www.jetbrains.com/idea>.

[7] *Eclipse – The Eclipse Foundation open source community website* [online]. 2010-08-20 [cit. 2011-05-14]. Dostupné z WWW: <http://www.eclipse.org>.

[8] *GPX: the GPS Exchange Format* [online]. 2011-05-10 [cit. 2011-05-14]. Dostupné z WWW: <http://www.topografix.com/gpx.asp>.

[9] *Obtaining a Maps API Key – Google Projects for Android* [online]. 2011-05-18 [cit. 2011-05-17]. Dostupné z WWW: <http://code.google.com/intl/cs/android/add-ons/google-apis/mapkey.html>.

[10] *Package Index – Android Developers* [online]. 2011-05-13 [cit. 2011-05-17]. Dostupné z WWW: <http://developer.android.com/reference/packages.html>.

[11] *The AndroidManifest.xml File – Android Developers* [online]. 2011-05-02 [cit. 2011-05-17]. Dostupné z WWW: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

[12] Klubal, O.: *Aplikace pro mobilní telefon zpracovávající data z GPS (Symbian).* Brno, 2010. 36 s. Bakalářská práce. VUT v Brně, Fakulta informačních technologií. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=9846>.

[13] Komatineni, S.; Hashimi, S.; MacLean, D.; aj.: *Pro Android 2.* Apress, 2010, iSBN 978-1-4302-2659-8.

[14] Rao, L.: *TechCrunch* [online].2011-04-14 [cit. 2011-05-09]. Google: 3 Billion Android Apps Installed; Downloads Up 50 Percent From Last Quarte. Dostupné z WWW: <http://techcrunch.com/2011/04/14/google-3-billion-android-apps-installed-up-50-percent-from-last-quarter>.

[15] Rapant, P.: *Družicové polohové systémy [online].* VŠB - TU Ostrava, 2002, iSBN 80-248-0124-8.