Tartalomjegyzék

1. Bevezetés

Korunk legdinamikusabban fejlődő ágazataihoz tartozik a mobiltechnológia, illetve az internet. Az emberek generációtól függetlenül egyre több időt töltenek az internetezéssel. Mindenkinek van már mobiltelefonja, melyen egyre több funkciót használ. Főként az okostelefonok megjelenésével, valamint a piacokon található mobilcsomagokból kiindulva látható az is, ahogy a felhasználói szektorban a mobilkészülékek egyre inkább képesek átvenni a számítógépek szerepét. A fejlett telefonok nagy része rendelkezik mind beépített navigációs rendszerrel, mind fényképezővel. Szintén erőteljes jelenség a játékoknál az online, egyszerre több felhasználó által játszható játékok megjelenése és terjedése. Ebben az esetben a felhasználó nem egymaga játszik, hanem más felhasználókkal csoportot alkotva együtt küzdenek a cél eléréséért egy másik felhasználói vagy gépi csoport ellen a győzelem érdekében.

Napjainkban nagyon sok időt töltünk zárt térben főleg akkor, ha számítógépes vagy telefonos játékokat játszunk és egyre kevesebbet a szabadban vagy a természetben, ezáltal sokszor nem is igazán ismerjük még a közvetlen környezetünket sem. Ebből kifolyólag hasznos volna egy olyan játék megvalósítása, melyet a szabadban lehet játszani, sétálni, mozgásra ösztönöz, és amelynek segítségével jobban megismerhetjük a környezetünket. Fontos szempont az, hogy jól játszható legyen mind városi, mind városon kívüli környezetben és mégis kellő izgalmat biztosítson.

A szakdolgozat célja egy régi nagy népszerűségnek örvendő játéknak, a fogócskának a modernizálása. A fogócskánál mindig vannak fogók(üldözők) és elfogandók(üldözöttek). Az üldözők célja, hogy elkapják az üldözötteket akik így, kiesnek a játékból. Az üldözöttek célja az, hogy ne tudják őket elkapni és ezért ők egyfajta taktikus menekülést végeznek a játék folyamán.

A megvalósított programban az egymáshoz pozícióban közeli felhasználók játszhatnak egymással a telefonjaik segítségével. A játékhoz legalább két, egymáshoz kellő közelségben lévő játékos szükséges. Minden játékos telefonja folyamatosan küldi a saját pozícióját és információt kap az ellenfélről pozíció formájában. A játék

végessége érdekében az üldözőknek egy órájuk van az összes üldözött elkapására a játék megnyeréséhez, különben az üldözöttek nyernek. Az elkapás feltétele az, hogy egy üldöző az üldözötthöz megfelelő közelségbe kerüljön. Egy pontrendszer bevezetésével elérhető, hogy a játékon belüli egyéni teljesítmények is megfelelően jutalmazva legyenek.

A játék rendszere a következőképpen kell, hogy felépüljön.

Az első rész az adatbázis megvalósítása. Ennek a résznek a szerver oldali megvalósítása biztosítja az adatok megfelelő és biztonságos kezelését. Fontos, hogy ez egy jól védett, csak a megfelelő programok által hozzáférhető helyen legyen található. Az adatbázisban letárolódnak az egyes felhasználók adatai. Itt jelennek meg a megfelelő jogosultsági szintek is, melyeknek megfelelően az egyes felhasználótípusok hozzáférhetnek a számukra elérhető funkciókhoz. Az adatbázis kezeli, tárolja és biztosítja a játék folyamatához nélkülözhetetlen adatokat. Ezeknek az adatoknak gyorsan hozzáférhetőnek kell lenniük a játékok szerver általi levezényléséhez és a teljes webes megjelenítéséhez.

Lényeges az, hogy legyen egy webes felület, melyen keresztül mind a felhasználói, mind az adminisztrációs részek elérhetőek. Egy játékos ezen a felületen keresztül tudja magát regisztrálni, majd a későbbiekben bejelentkezve megtekinteni saját statisztikáit és a különböző ranglistákat, módosítani tudja személyes adatait, amennyiben bármilyen változás állna be azokban, valamint itt található egy felület, amin keresztül betekintést nyerhet az éppen folyamatban lévő játékok menetébe nézőként.

A webszerver feladata a játékok megszervezése és lebonyolítása, összeköttetést hoz létre az egyes játékban lévő felhasználók között és elvégzi a megfelelő adminisztrálást az adatbázisban.

A kliens feladata a játék felületének a biztosítása. A weboldalon regisztrált felhasználók a kliens programon keresztül léphetnek be a játékba. Sikeres bejelentkezés után az alkalmazáson belül nyomon követhető a játék alakulása. A különböző felületeken a játékos látja saját, és ellenfelei pozícióit is, szerepe szerint készíthet vagy lekérhet fényképet, látja a számára fontos adatokat, mint például: ellenfél elkapása, új

fénykép, és a játék végén a végeredmény.

A szakdolgozat feladata egy fentiekben vázolt programrendszer megvalósítása, melyet másodmagammal készítek.

Ennek megfelelően a dolgozat csak a feladat egy részét dokumentálja. Az én feladataim az alábbiak:

- Kliens alkalmazás megvalósítása *Android* platformon.
- Játék felület felépítése.
- Webszolgáltatással való kommunikáció megvalósítása.

2. Technológiai háttér

Ez a fejezet a szakdolgozatomhoz felhasznált technológiákat taglalja. A legfontosabb természetesen az *Android* operációs rendszer, felépítése, szerkezete és működése (2.1. fejezet). Ezen kívül a programban felhasználásra került térképfelület is, ami a *Google* által fejlesztett *Google Maps*-t (2.3. fejezet) használja. A webszolgáltatással való kommunikáció SOAP alapú, ami a (2.4. fejezet) alatt kerül bemutatásra.

2.1. Android OS

Az *Android* egy *Linux* alapú nyílt forráskódú operációs rendszer, amit az *OHA* (*Open Handset Alliance*) fejleszt. Az *OHA* egy olyan cégcsoportosulás, amiben megtalálhatók a legnagyobb készülékgyártók (*HTC, LG, Samsung*...), a jelentős szolgáltatók, (pl. *T-mobile*, *Vodafone*) és rengeteg szoftveres cég, többek közt maga a *Google* is. Az, hogy a köztudatban az *Android* és a *Google* összekapcsolódik, annak köszönhető, hogy a *Google* vásárolta fel az *Android Incorporated*-et 2005-ben. [1]

Az operációs rendszer egy *Linux* kernelre épül, ami egy réteg a hardver és a szoftver között (1. ábra). Itt találhatóak az adott eszközhöz a gyártó által megírt meghajtók, illetve a kernel feladatai közé tartozik még a folyamatok ütemezése, és a memória kezelése is. Részben erre épül a DVM, azaz a *Dalvik Virtual Machine.* Fontos megemlíteni, hogy a DVM sokban különbözik a Java virtuális géptől. Fordítás után a fájlok egy **`dex`** kiterjesztésű fájlba kerülnek, ami optimalizáltabb a JVM-nél (Java Virtual Machine) megszokott **`class`** kiterjesztésű fájloknál.[2]

1. ábra: Az Android OS felépítése

2.1.1. Androidról felhasználói szemszögből

Ha felhasználói szempontból vizsgáljuk az *Androidot,* akkor először is érdemes szót ejteni a fő konkurenciákról, hogy lássuk, miben is áll az *Androidos* telefonok népszerűsége. A történet az *iPhone* 2007-es megjelenésével kezdődött. A mobilpiac addigi készülékeitől az *Apple* új telefonja jelentősen eltért, megalkotva ezzel egy új kategóriát, a „smart phone”-t. Jellemzői alapvetően az érintőképernyő, a teljes értékű internetezési lehetőségek és ezen kívül a rengeteg letölthető alkalmazás, aminek csak egy része ingyenes. Az év végéig meginghatatlannak tűnt az *iPhone* térnyerése, de még ugyanabban az évben bejelentette a *Google* az *Androidot*, majd nem sokkal később piacra dobta az első ilyen készüléket, a *HTC Dream*-et (2. ábra), amivel megjelent az *iPhone* egyik legfőbb konkurenciája.

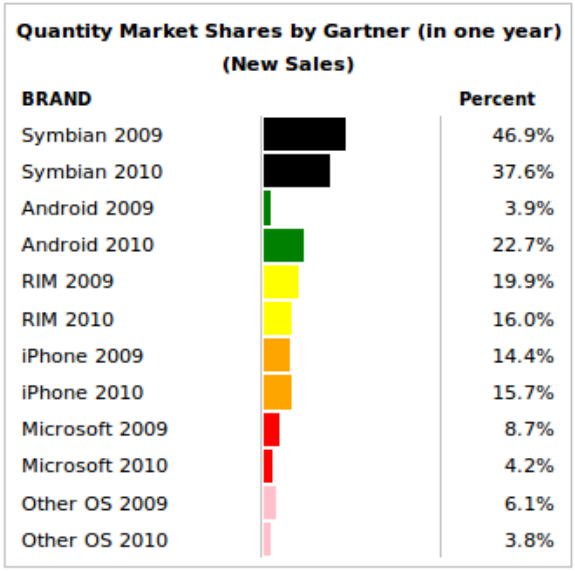
2. ábra: HTC Dream

Jelentős különbség ugyanis, hogy míg az *iOS* zárt, addig az *Android* nyílt forráskódú, tehát bárki fejleszthet rá, és kedve szerint alakíthatja. Ez egy átlagos felhasználó számára annyiban fontos, hogy napról napra jelennek meg új alkalmazások, widget-ek, amiből kedvére válogathat. Az alkalmazások között vannak fizetősek és ingyenesek egyaránt, ezek elérhetősége az adott országtól függ. Magyarországon például az *Android Marketnek* „csak” az ingyenes felületéről tölthetnek le a felhasználók különböző programokat.

A másik lényeges tulajdonsága az *Androidnak*, amivel maga felé billenti mérleget, az az, hogy nem fonódik össze kizárólagosan egy gyártó készülékével. Ennek köszönhetően már szinte az összes nagy mobilgyártó cég repertoárjában találhatunk

Androidos telefont, tehát aki vásárlásra adja a fejét, könnyen találhat az igényeinek megfelelőt. Ezen felül még érdemes megemlíteni, hogy az *Apple iPhone* készülékei felső kategóriások, ezért sokan nem engedhetik meg maguknak, így ha valaki mégis okos telefonra vágyik, akkor *Androidos* telefonokkal a középkategóriában is bőven találkozhat. Ezzel egy új felhasználói réteget sikerült bevonni a piacra, ami meg is látszik a részesedések alakulásában.

Látható, hogy a közel 23%-ra való növekedés, főleg a *Symbian* kárára történt (3. ábra), míg az *iPhone* stagnál. Szinte minden országban, az *iPhone4* megjelenése ellenére is több *Androidos* telefont vásároltak. Ezek az arányok 2011 végére valószínűleg további jelentős változásokat mutatnak majd. [3]



3. ábra: Statisztika

2.1.2. Android SDK

Az *Android SDK (Software Develelopement Kit)* egy szoftverfejlesztő készlet, ami átfogó fejlesztési eszközöket tartalmaz, mint például a könyvtárak, az emulátor, a minta kódok, illetve a nyomkövető.

Az emulátor egy virtuális eszköz, amelynek segítségével lehet tesztelni a megírt programokat anélkül, hogy szükség lenne egy tényleges fizikai eszközre. Megjelenésében és felépítésében is olyan, mint egy teljes értékű *Androidos* készülék, amelynek a tulajdonságai az *AVD (Android Virtual Device) Managerben* könnyen megadhatóak, kezdve az SD kártya méretétől egészen az GPS vevőig.

Az *SDK* hivatalosan támogatott integrált fejlesztői környezete az *Eclipse* (legalább 3.5-ös) és hozzá az *ADT (Android Development Tools) Plugin*. Az *ADT* kibővítve az *Eclipse* lehetőségeit, segít abban, hogy gyorsabban lehessen létrehozni *Android* projektet, és kényelmesebben a felhasználói felületet. Segítségével a nyomkövetés egyszerűbbé válik, hiszen már az első verzióktól kezdve rendelkezik beépített *DDMS-el (Dalvik Debug Monitor Server).* [4]

A fejlesztés a legtöbb platformon lehetséges, úgymint:

- 8.04 vagy újabb *Ubuntu*
- *Mac OS X* (minimum 10.4.9)
- *Windows XP* vagy újabb

2.2. Java

Az *Android* platformra Java nyelven lehet programozni, így érdemes erről is pár szót ejteni. A Java egy objektumorientált nyelv, amelyet *Sun Microsystems* fejlesztett egészen addig, amíg az *Oracle* fel nem vásárolta. A nyelv szintaxisa C++ -hoz hasonló, de annál jóval egyszerűbb objektumokat tartalmaz. A nyelv egyik fő sajátossága, hogy

platformfüggetlen, azaz a program minden hardveren ugyanúgy fog futni. Ezt úgy oldották meg, hogy a program egy virtuális gépen fut, az úgynevezett *Java Virtual Machine-on (JVM)*. A fordítóprogram a programot byte-kódra fordítja le, amit aztán a JVM az adott hardver gépi kódjára, így válik a program hordozhatóvá. [5]

2.3. Google Maps

A *Google Maps* egy ingyenes online térképszolgáltatás, amelyet a *Google* fejleszt. 2005-ben indult el, akkor még *Google Local* néven, és mára már szinte az egész Földről rendelkeznek nagyfelbontású képekkel, amik részben műhold felvételek, részben repülőről készült fényképek. Az egyszerű térképfunkció mellett még sok egyéb szolgáltatást is nyújt, ilyen például az útvonaltervezés, webkamerák képei, utcanézet, vagy akár aktuális forgalmi információk. Nagy előnye a könnyű kezelhetőség, és hatékonyság, ugyanis egyszerre mindig csak akkora területet tölt be, amekkora az adott nézet. Ez különösen akkor lényeges, ha egy mobil alkalmazásban használjuk, ahol törekedni kell a minimális adatforgalomra és memóriahasználatra. [6]

2.3.1. Google Maps API

A *Google Maps API* egy ingyenes szolgáltatás, ami lehetővé teszi a *Google Maps* használatát egy weboldalon vagy egy alkalmazásban. Használatához mindössze egy API kulcsot kell igényelni, ami szintén ingyenesen megtehető.

2.4. SOAP

A SOAP egy XML-alapú protokoll, ami különböző strukturált adatok küldését teszi lehetővé, elsősorban webszolgáltatásoknál használják. Egy ilyen SOAP formátumú üzenet továbbítása általában más protokollokra támaszkodik, ezek leggyakrabban az *RPC* (*Remote Procedure Call*) és a *HTTP* (*Hypertext Transfer Protocol*). Egy SOAP üzenet funkcióját tekintve háromféle lehet: [7]

- kérés (request)
- válasz (response),
- hiba (fault).

Ezeket az alábbi példák mutatják be:

2.4.1. Request

Az alábbi példa egy egyszerű kérés, ami meghívja a webszolgáltatás *SignIn* metódusát, ami vár egy „string” típusú felhasználónevet és jelszót, illetve az aktuális pozíció szélességi és hosszúsági koordinátáit, ami egy-egy „double” típusú érték. A fejlécben látható, hogy a kérés HTTP POST metódust használ. A „host” mezőben a webszolgáltatás helye található, ahova a kérést küldjük, a „SOAPAction” mezőben pedig a webszolgáltatás adott függvényének a neve található.

```
POST /Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/SignIn"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SignIn xmlns="http://tempuri.org/">
      <username>string</username>
      <password>string</password>
      <lat>double</lat>
      <lng>double</lng>
    </SignIn>
  </soap:Body>
</soap:Envelope>
```

2.4.2. Response

Az alábbi példa a válasz a kérésünkre, amennyiben az helyes volt. Jelen esetben ez egy logikai érték, ami jelzi, hogy indulhat-e a játék, illetve az aktuális dátum. Látható, hogy a válaszüzenet fejléce hasonló, mint a kérés fejléce, de különbség az első sor. A

„**HTTP/1.1 200 OK**” itt azt jelenti, hogy a kérést sikerült végrehajtani.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <SignInResponse xmlns="http://tempuri.org/">
      <SignInResult>
        <Indulhate>boolean</Indulhate>
        <Pontosido>dateTime</Pontosido>
      </SignInResult>
    </SignInResponse>
  </soap:Body>
</soap:Envelope>
```

2.4.3. Fault

A harmadik típusú üzenet, a hibaüzenet. Ilyet akkor küld vissza a webszolgáltatás, ha probléma lépett fel a kérés teljesítése közben. Ilyen hiba lehet például, ha nem sikerült csatlakozni az adatbázishoz, helytelen a paraméterezés, vagy rossz a felhasználónév-jelszó páros. A hibaüzenet ekkor az üzenet törzsén belül egy **<faultstring>** tagban olvasható.

```
HTTP/1.0 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: 525

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
   <soap:Fault>
     <faultcode>soap:Server</faultcode>
     <faultstring>Server was unable to process
request</faultstring>
     <detail/>
   </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

2.4.4. kSOAP 2

A *kSOAP2* egy egyszerű SOAP könyvtárcsomag *Java* alapú mobil alkalmazásokhoz. Tartalmazza az alaposztályokat, amelyek a SOAP kommunikációhoz és a szerializációhoz szükségesek. Sajnos, a hozzá tartozó dokumentáció egyelőre meglehetősen szűkszavú.

3. Felhasználói dokumentáció

A felhasználói dokumentáció bemutatja a programhoz szükséges tudnivalókat, úgymint pl.: telepítés lépései, technikai követelmények, illetve az esetlegesen fellépő hibák. Ezen információkon túl ismerteti a játék szabályait, menetét és pontrendszerét a felhasználó számára.

3.1. Követelmények

A játékot mobiltelefonon („okostelefonon”) keresztül lehet játszani, aminek az alábbi követelményeknek kell megfelelnie: 2.1-es Android operációs rendszer, GPS vevő, internet kapcsolat, fényképező. Ha a készülék eleget tesz a fentieknek, akkor megfelelő az alkalmazás futtatásához.

3.2. A játék leírása

A játék menete a weboldalra való regisztrációval kezdődik. Ez teljesen szokványos módon történik. A webes felületen meg kell adni egy választott felhasználónevet, illetve a teljes nevet, e-mail címet, és a jelszót. Ezek után lehet belépni a játékba. Ha egy játékos bejelentkezett, akkor az aktuális helyzete elküldésre kerül a webszolgáltatásnak, ami ezt eltárolja, és ez alapján megállapítja, hogy van-e már elegendő ember a játék elindításához. Ez úgy történik, hogy megvizsgálja az bejelentkezett felhasználó pozícióját, és ennek egy adott sugarú környezetét. Ha ezen a területen összesen legalább 4-5-en vannak, akik szintén várakoznak, akkor elindulhat a játék. A webszolgáltatás kiosztja, hogy ki milyen „szerepet” kap. Ehhez figyelembe veszi az aktuális koordinátákat, és a résztvevő játékosok számát. Kétféle szerep van: üldöző és üldözött. Ezek alapján válnak szét az aktuális játékosok két csapatra.

Az üldözők feladata, hogy az egy óra hosszán át tartó játék során az összes üldözöttet elkapják, míg az üldözötteknek a célja, hogy minél tovább életben

maradjanak. Az egyenlő esélyek biztosítása érdekében a csapatok nem ugyanazokat az információkat kapják meg az ellenségről. Az üldözők folyamatosan értesülnek a hozzájuk képest két legközelebbi ellenfélről, megkapják azok koordinátáit, ami az alkalmazás térképfelületén nyomon követhető. Ezen felül extra információ gyanánt tízpercenként fényképet is megtekinthetnek az üldözöttek jelenlegi tartózkodási helyéről.

Ezzel ellentétben az üldözöttek csak két, véletlenszerű üldözőjük helyzetét kapják meg, így elkerülhető, hogy az üldözöttek könnyen elmenekülhessenek, hiszen így bármikor belefuthatnak egy olyan ellenfélbe, akinek a helyzete addig ismeretlen volt számukra. A játék menete során tíz percenként kapnak egy értesítést, hogy készítsenek képet, amit az alkalmazás fényképfelületén tehetnek meg. A fénykép ezután elküldésre kerül, és az üldözők megkaphatják, ha kérik. Egy üldözött akkor esik ki, ha egy üldöző adott sugarú környezetébe kerül.

A játéknak több kimenetele lehet aszerint, hogy meddig tartott, illetve ki nyert. Ha az üldözők minden üldözöttet kiejtenek az egy órás időkereten belül, akkor a játék véget ért, és a pontjuk jóváírásra kerül, ilyenkor a vesztesek semmit se kapnak. Ha lejárt az egy óra, és még van olyan üldözött a játékban, aki életben van, akkor az üldözöttek csapata nyert. Ekkor minden üldözött kap pontot, akár túlélte az üldözést, akár nem, de természetesen, aki életben maradt, az arányosan több pontok kap. Ilyenkor az üldözőknek is jár a pont minden kiejtett ellenfél után. Ezeket az eredményeket és az összesített statisztikát is a webes felületen lehet megtekinteni.

3.3. Pontrendszer

	Üldöző
Elkap egy üldözöttet	+20 pont
Az üldözők csapata nyer	+10 pont

	Üldözött
A játék végéig életben marad	+20 pont
Az üldözöttek csapata nyer	+10 pont

3.4. Telepítés

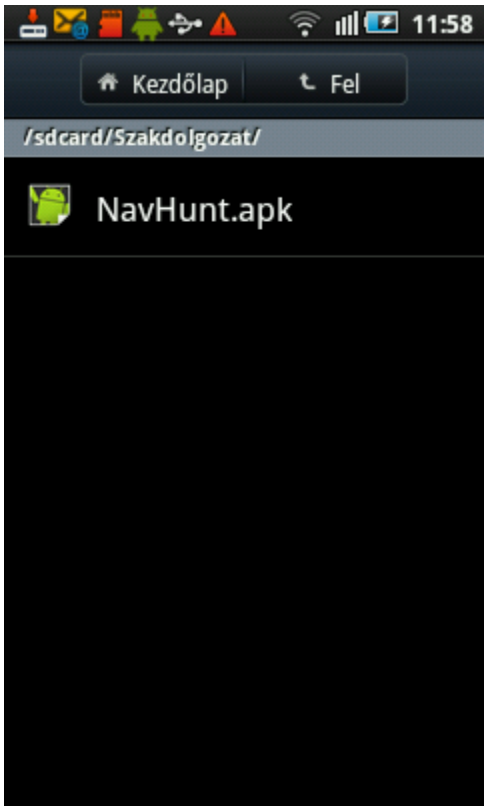
A játék telepítése a mobiltelefonra meglehetősen egyszerű. A lefordult program egy APK fájlba kerül, amelynek helye a következő:

```
../<project name>/bin/<project name>.apk
```

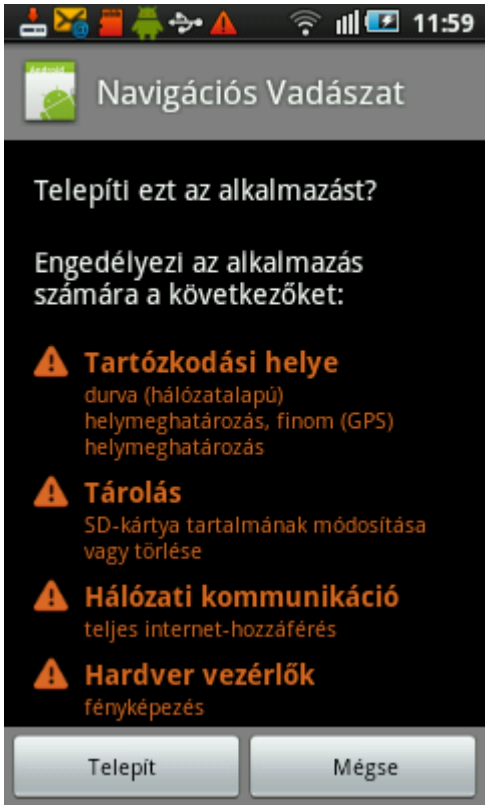
A telepítéshez erre a fájlra van csak szükségünk. A telepítés lépései Windows 7 operációs rendszeren az alábbiak:

1) Csatlakoztassuk a mobileszköz a számítógéphez a hozzá való USB kábellel.
2) A Computer/Sajátgép menüben válasszuk ki a telefon memória kártyáját.
3) Másoljuk az előbbi APK fájlt egy tetszőleges mappába a telefon SD kártyájára.
4) Ha ez sikeresen megtörtént, leválaszthatjuk az eszközt a gépről.
5) A telefonon lépjünk be a fájlkezelőbe (4. ábra).
6) Keressük meg, hogy hova mentettük a fájlt, majd kattintsunk rá (5. ábra).
7) Ezt követően a megjelenik egy képernyő, amelyen látható, hogy a telepíteni kívánt programnak milyen hozzáférhetőségekre van szüksége, amelyeket engedélyeznünk kell (6. ábra).
8) Ezután már csak rá kell kattintani a telepítés gombra, és pár másodperc múlva már használhatjuk is az alkalmazást. A program ezután a menüből elérhető, de akár a telepítés után az „Megnyitás” gombbal rögtön ki is próbálhatjuk (7. ábra).

4. ábra: Fájlkezelő megnyitása



5. ábra: Telepítendő fájl megkeresése



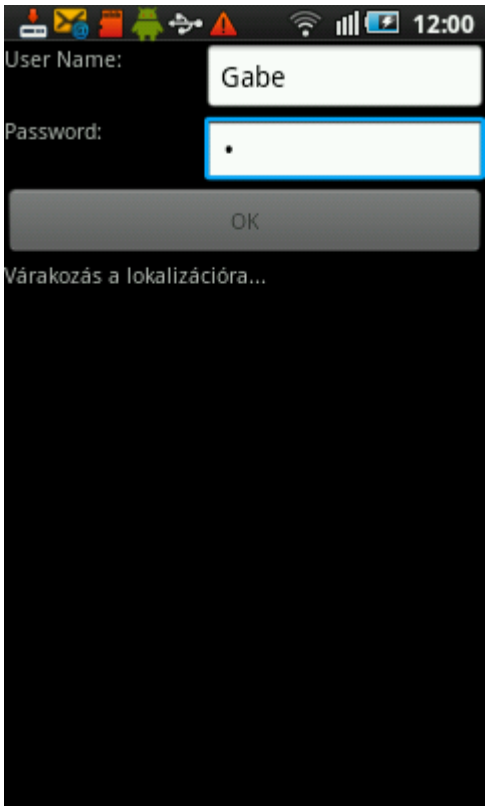
6. ábra: Hozzáférések engedélyezése és telepítés

7. ábra: Telepítés befejezése

3.5. A játék folyamata

3.5.1. Bejelentkező felület

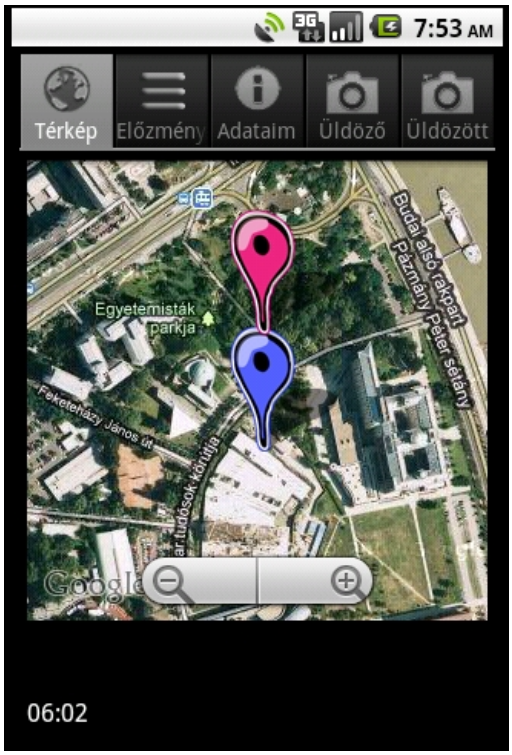
A játék elindítása után a 8. ábrán látható képernyőkép jelenik meg. A belépéshez meg kell adnunk az előzőleg a webes felületen regisztrált felhasználónevet és jelszót. Ez még nem elegendő a bejelentkezéshez, ugyanis a programnak szüksége van az aktuális pozícióra is. A helymeghatározás a készülékben található GPS vevővel történik. Ez igénybe vehet egy kis időt, de mindenképp szükség van rá, hiszen a webszolgáltatás ez alapján osztja be a játékosokat az egyes játékmenetekbe. Amíg ez nem sikerül, addig az „OK” gomb inaktív. Viszont amint sikerült megfelelő GPS jelet fogni, rögtön aktívvá válik, így lehetővé téve a belépést a játékba.



8. ábra: Bejelentkező felület

3.5.2. Térkép nézet

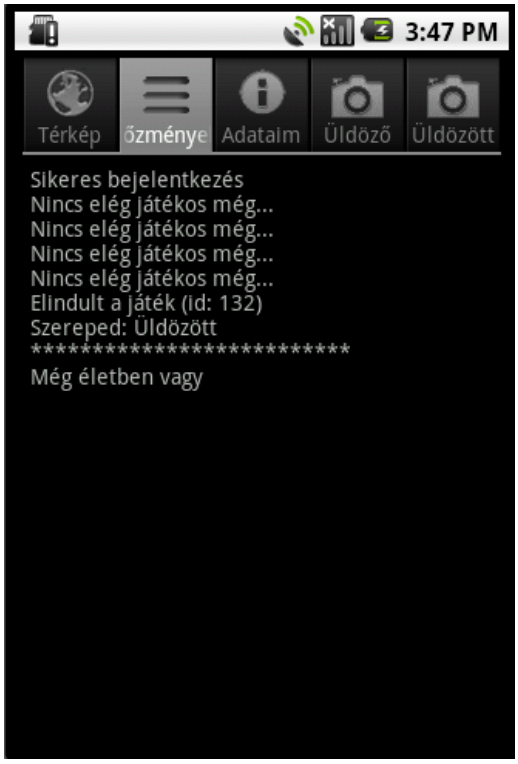
Ha az OK gombra kattintva sikerült a bejelentkezés, akkor egy lapokra osztott felület töltődik be, amin az első fül az aktív. (9. ábra) Ezen a fülön egy *Google Maps* térkép látható, amin egy kék buborék jelzi a legutóbbi ismert pozíciót, ami ilyenkor még az, amivel beléptünk. Ez a koordináta percenként frissül, olyankor már az új adat jelenik meg a térképen. A felületen van még egy számláló is, ami a belépéskor elindul, így jelzi mennyi idő telt el a játék indítása óta.



9. ábra: Térkép nézet

3.5.3. Előzmények nézet

Az *Előzmények* felületen jelennek meg az aktuális információk. Bejelentkezés után a játékos aktuális helyzete továbbítódik a webszolgáltatásnak, ami percenként megnézi, hogy van-e elegendő játékos a térségben a játék indításához. Ez úgy történik, hogy a legfrissebb pozíció 1 kilométeres sugarú környezetét vizsgálja, további játékra várakozókat keresve. Az eredménynek megfelelően olvashatóak az üzenetek. A 10. ábrán látható, hogy a játékos be van lépve, de további játékosokra kell várakoznia.



10. ábra: Előzmények nézet

3.5.4. Adatok nézet

A harmadik fül az alkalmazásban az *Adataim*. Ennek két fő része van, egyik a játékosra vonatkozó információkat tartalmazza, amik állandóak a játék folyamán. Ezek a következőek: felhasználó név, a játékos azonosítója és a szerepe. Amíg nem indult el a játék, addig természetesen az utóbbiak üresek, egészen addig, amíg a webszolgáltatás beosztja egy új játékmenetbe. Ekkor kapja meg a kiosztott azonosítót, és hogy melyik csapathoz tartozik.

A másik része ennek a felületnek csak a játékhoz kapcsolódó dinamikus információkat mutatja. Ilyen például a legközelebbi ellenfél, ahol a játékos neve olvasható, illetve rögtön utána az, hogy milyen messze van. Ez különösen akkor hasznos, ha a térképen nem látszik egyértelműen, hogy ki a legközelebbi játékos. (11. ábra)

11. ábra: Adatok nézet

3.5.5. Üldöző nézet

Ezen a felületen az üldöző megtekintheti a legfrissebb képeket a két legközelebbi üldözöttről. Amíg nincsenek még kiosztva a szerepek, addig a gombok alatt a „Nincs elérhető kép a játékosról” szöveg olvasható. Amennyiben a játékos megkapta a szerepét, ami üldözött, akkor bármelyik gomb megnyomása után a szövegdobozban erre vonatkozó figyelmeztetés lesz látható. Így nem áll módjában egy üldözöttnek képet lekérnie.

Ha a játékos üldöző lett, akkor a két gombbal tudja váltogatni, hogy melyikük fényképét szeretné megtekinteni. Ha elérhető már kép, akkor a szövegdobozban látható az ellenfél neve, akitől a kép származik. (12. ábra)

12. ábra: Üldöző nézet

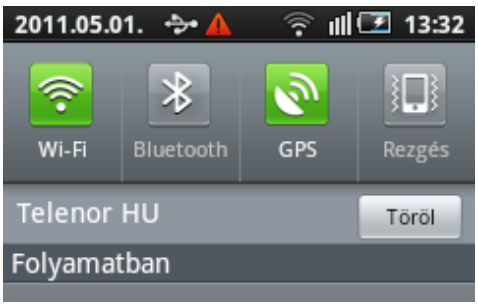
3.5.6. Üldözött nézet

Az üldözött nézeten fényképeket lehet készíteni. Ez úgy történik, hogy az üldözöttnek az *Előzmények* felületen tízpercenként megjelenik egy figyelmeztetés, miszerint le kell fotóznia az aktuális helyét, majd ezt kell elküldenie. Fontos, hogy olyan képet kell készíteni, amin értékelhető tartalom van. Amennyiben ezt nem teljesíti, és később reklamáció érkezik emiatt, akkor a játékban szerzett pontjai törlődnek a játékosnak, így neki is érdekében áll, hogy használható fényképpel szolgáljon. Ha egy olyan játékos szeretné ezt a felületet használni, akinek a szerepe még nem kiosztott, vagy pedig üldöző, akkor természetesen nem tudja a funkciókat használni, amire a szövegdobozban megjelenített szöveg is figyelmezteti.

3.6. A program futtatása során felléphető problémák:

- **A bejelentkező képernyőn sose válik az OK gomb aktívvá**

  Lehetséges okok:



13. ábra: GPS bekapcsolása

  - GPS vevő kikapcsolt állapotban van. Ezt könnyen ellenőrizhetjük, ha a telefon felső menüsávját lejjebb húzzuk. Ha a GPS ikon nem aktív, akkor kattintsunk rá, ezzel bekapcsolva azt. (13. ábra)

  - Amennyiben azt találjuk, hogy a GPS be van kapcsolva, akkor valószínűleg nem kap jelet. Ez jellemző, ha belső térben tartózkodunk, vagy leárnyékolt helyen.

- **Az OK gomb aktív, mégsem sikerült a bejelentkezés**

  Lehetséges okok:

  - Nincs internet kapcsolat. Ellenőrizzük, hogy van-e mobilinternet kapcsolatunk és/vagy be van-e kapcsolva a WIFI a telefonon. (13. ábra)

  - A webszolgáltatás valamilyen oknál fogva éppen nem elérhető. Ilyenkor az OK gomb alatt szövegdobozban a „Connection Timed Out” felirat jelenik meg, ez esetben próbáljuk meg később újra a bejelentkezést.

  - Rossz felhasználónév és/vagy jelszó. Ebben az esetben a „Sikertelen bejelentkezés” szöveg olvasható a gomb alatt.

- **Megszakadt játékmenet**
    - Ez abban az esetben fordulhat elő, ha a játékos több mint két percig „tétlen”, azaz nem küld semmiféle koordinátát a webszolgáltatásnak. Ezt az internet kapcsolat elvesztése okozhatja. Két perc tétlenség utána a játékos nem térhet vissza a játékba, mert a webszolgáltatás kidobta az aktuális játékmenetből, tehát már csak egy új játékba csatlakozhat be.

4. Fejlesztői dokumentáció

Az alkalmazás fejlesztése objektumorientált programozási paradigma segítségével történt, vízesésmodellnek megfelelően. Modellezőkörnyezetként a *Sparx Enterprise Architect*, míg fejlesztőkörnyezetként az *Eclipse 3.6*-os verziója szolgált. A fejlesztés, valamint a tesztelés jelentős része mobil emulátor segítségével történt, míg az integrációs tesztek futtatása *Samsung Galaxy 3* mobilkészülékkel történt.

4.1. Mérföldkövek

A program fejlesztése az alábbi mérföldkövekkel került megvalósításra:

1) A webszolgáltatáson keresztül történő kommunikáció megvalósítása.

   Ehhez a már korábban említett *kSOAP* könyvtár került felhasználásra, amelyhez sajnos nem találtam kellően részletes dokumentációt.

2) A játék felületének felépítése.

   Itt a fontosabb a részfeladatok a térkép felület integrálása, földrajzi helyzet lekérése (GPS vevő használata) és a fényképező nézet megírása volt.

3) Adatbázis használata.

   Addig ugyanis csak a webszolgáltatás bejelentkező, és *canPlay* függvényét használtam, ami nem igényelt adatbázis elérést, mert a tesztelésnél a felhasználó/jelszó páros a programba bele volt drótozva.

4) A játék végleges „összeállítása”.

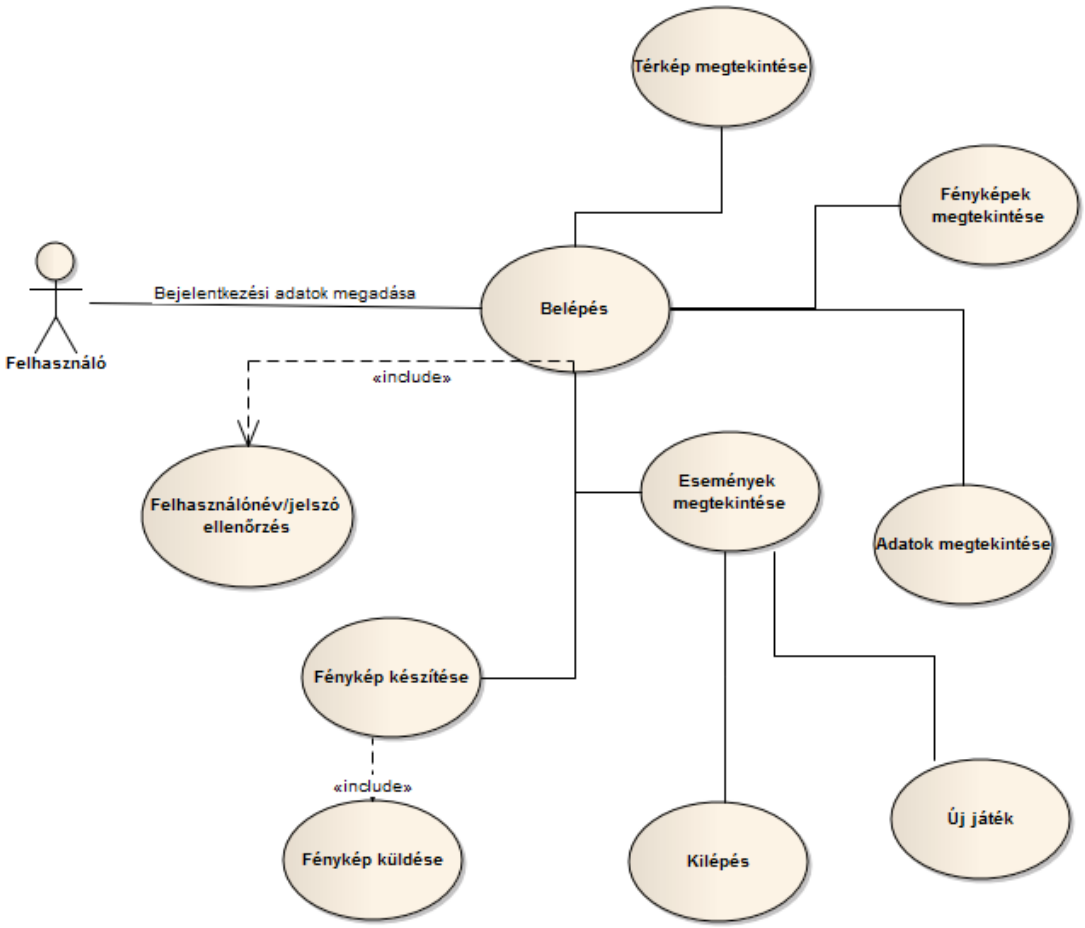
   Ekkor már az egyes részek (kliens, webszolgáltatás, adatbázis) külön-külön jól működtek, úgyhogy már csak az együttes, helyes működés elérése volt a cél.

4.2. A program használati esetei

- **Bejelentkezés**
  - Felhasználó név megadása
  - Jelszó megadása
  - OK gombra kattintva bejelentkezés
- **Térkép fül**
  - Ellenfél nevének és távolságának megtekintése a térképen, az azt jelző markerre kattintva
  - Saját koordináta megtekintése a térképen a kék markerre kattintva
- **Előzmények fül**
  - Előzmények és aktuális események megtekintése
- **Adatok fül**
  - A játékosra vonatkozó adatok megtekintése
- **Üldöző fül**
  - „Kép1” gombra kattintva az egyik üldözött fényképének megtekintése
  - „Kép2” gombra kattintva a másik üldözött fényképének megtekintése
- **Üldözött fül**
  - A képernyőre kattintva kép készítése
  - „Új kép” gombra kattintva lehetőség új kép készítésére
  - „Küldés” gombra kattintva a fénykép elküldése a webszolgáltatásnak
- **„Játék vége” felugró ablak**

- „Új játék” gombra kattintva új játék indítása
- „Kilépek” gombra kattintva játék bezárása

A program használati eseteit a (14. ábra) mutatja be:



14. ábra: A program használati esetei

4.3. A fejlesztőkörnyezet telepítése

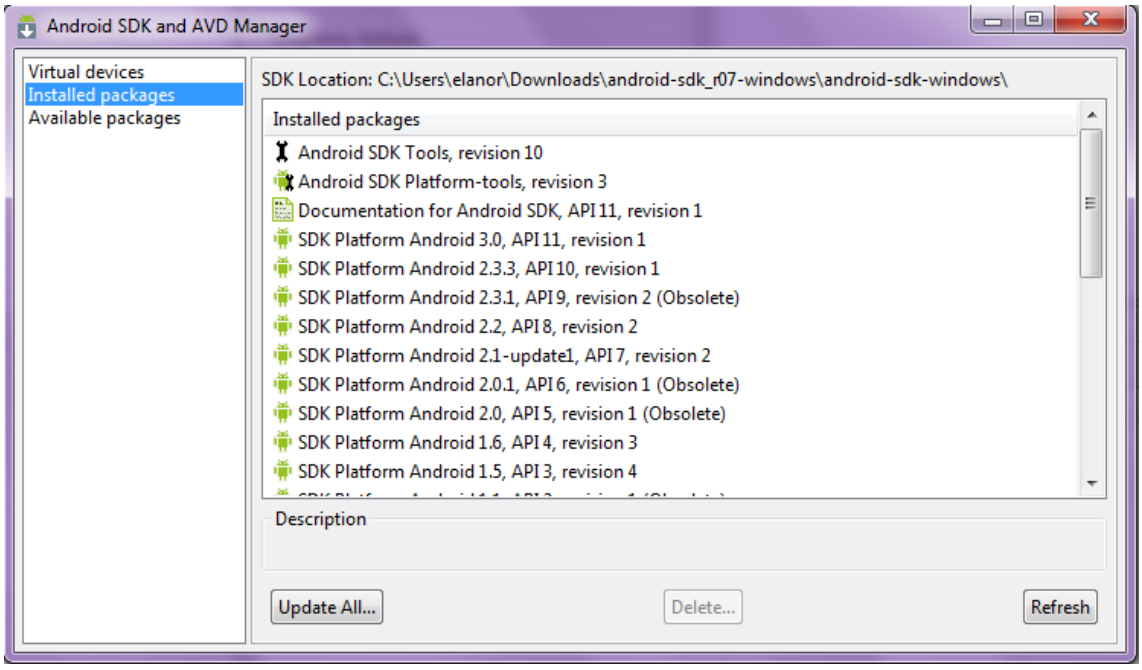
Ahhoz, hogy *Android* platformra fejleszthessünk, elő kell készíteni a fejlesztőkörnyezetet, ami az első alkalommal általában még elég időigényes. Erről olvasható egy útmutató a http://developer.android.com/sdk/installing.html oldalon.

Telepítés folyamata

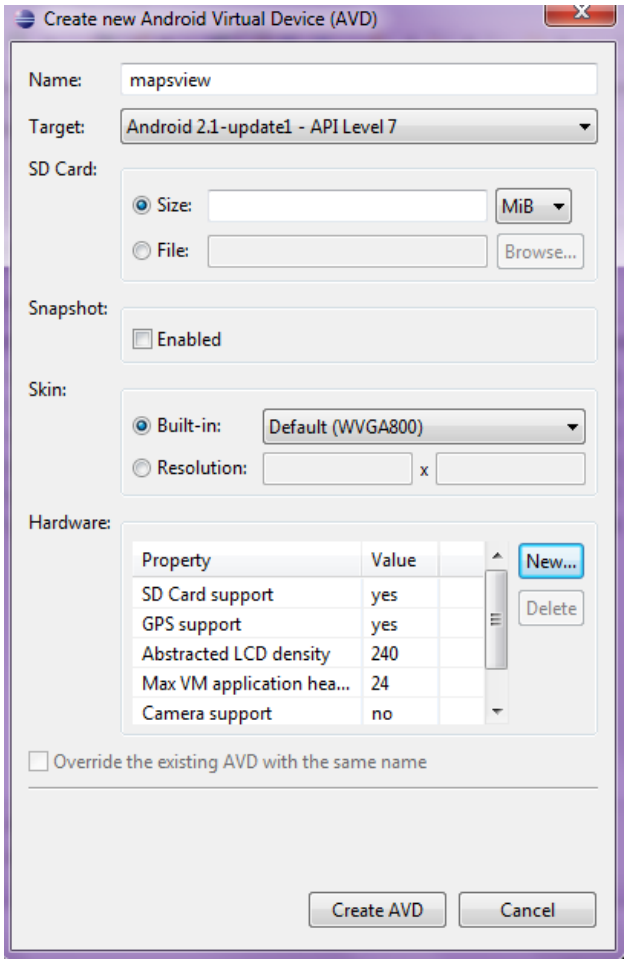
Először vessünk egy pillantást a rendszerkövetelményekre. Itt ellenőrizhetjük, hogy az általunk használt operációs rendszer megfelelő-e a telepítéshez. Ezenkívül itt látható, hogy mik szükségesek még az SDK telepítéséhez:

3.5 (vagy újabb) *Eclipse*, *Java Development Kit (JDK)* 5 vagy 6

Ezután letölthetjük az operációsrendszerünknek megfelelő *SDK*-t a http://developer.android.com/sdk/index.html oldalról. Kitömörítés után indítsuk el a `/tools` mappába található *Android* programot. Itt az *Available packages* menüpont alatt kiválaszthatjuk és telepíthetjük a platformhoz szükséges összetevőket. (15. ábra)



15. ábra: Android SDK and AVD Manager



16. ábra: Virtuális eszköz létrehozása

Következő lépésként létre kell hozni egy virtuális eszközt, amit szintén az *Android SDK and AVD Manager*-ben tehetünk meg, a *Virtual Device* menüpont alatt. (16. ábra)

Kattintsunk a *New* gombra, melyet követően a felugró ablakban létrehozhatjuk az új virtuális eszközünket. Fontos, hogy mindenképpen meg kell adni, milyen verziójú *Androidra* szeretnénk fejleszteni ennek az eszköznek a segítségével, illetve hogy milyen hardvertámogatás legyen benne, hiszen ettől függ, hogy tudjuk-e rajta tesztelni a programunkat.

Végül le kell tölteni még a legfrissebb *ADT Plugint* a http://developer.android.com/sdk/eclipse-adt.html oldalról. Ezt azután *Eclipse*-ben telepítsük fel (*Help/Install New Software...*) a szokásos módon. Ha befejeződött a

telepítés, az *Eclipse* újraindul, így már csak egy dolgunk maradt. Menjünk a *Window/Preference* menüpontba, és az oldalsó listából az *Androidot* kiválasztva adjuk meg a korábban már letöltött *SDK* elérési útvonalát. Ha mindez sikeresen megvan, akkor létrehozhatunk egy *Android* projectet a *File/New* menüpont alól. [8]

4.4. Kommunikáció webszolgáltatáson keresztül

A webszolgáltatással való kommunikáció SOAP alapú, a programban egy *kSOAP* nevű könyvtárcsomagot használunk ehhez, amint az a 2. fejezetben olvasható.

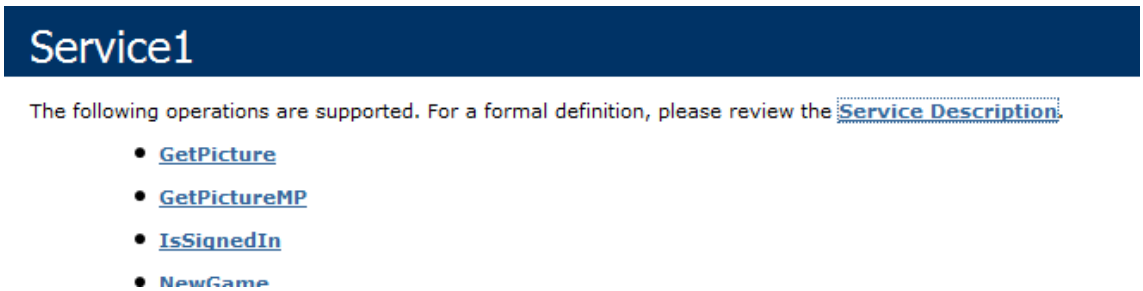
A kommunikáció felépítését a webszolgáltatás *canPlay* függvényén keresztül mutatom be:

```
String NAMESPACE = "http://tempuri.org/";
String URL =
"http://46.107.183.231:8080/Service1.asmx";
String METHOD_NAME = "canPlay";
String SOAP_ACTION = "http://tempuri.org/canPlay";
```

A kapcsolathoz szükséges szövegek helyes megadása:

- **NAMESPACE:** A webszolgáltatás névtere, alapértelmezetten ez a "http://tempuri.org/"
- **METHOD_NAME:** A függvény neve, amit szeretnénk meghívni. Fontos, hogy kis- és nagybetű érzékeny.
- **SOAP_ACTION:** A névtér és a függvény neve egy szövegben összefűzve.

A fentiek mind megtalálhatóak a webszolgáltatás WSDL (Web Services Description Language) leírásában, ami elérhető a webes felületről, a *Service Description* pont alatt. (17. ábra)



17. ábra: A webszolgáltatás leírásának megtekintése

- **`URL:`** A webszolgáltatás kívülről is elérhető címe

Input paraméterek beállítása, amennyiben szükséges:

```
SoapObject request = new SoapObject(NAMESPACE, method);
request.addProperty("username", username);
request.addProperty("password", passwd);
```

Létrehozzuk a kérést, ami egy *SoapObject,* majd hozzáadjuk a megfelelő paramétereket, amit a függvény vár. Az addProperty első paramétere a hívott függvény megfelelő paraméterének a neve, a második pedig a várt típusnak megfelelő változó.

Ezután létrehozunk egy szerializált *soapEnvelope*-t, aminek segítségével küldhetjük és fogadhatjuk az üzeneteket.

```
SoapSerializationEnvelope envelope = new
     SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.setOutputSoapObject(request);
```

Majd utána, ha .NET-es webszolgáltatáshoz kapcsolódunk, akkor a **`dotNet`** flaget igazra kell állítani, hogy a *SoapEnvelope* kódolása kompatibilis legyen a .NET-es alapértelmezett kódolással.

```
envelope.dotNet = true;
```

Ezt követően kapcsolódunk a webszolgáltatáshoz, majd meghívjuk a függvényt.

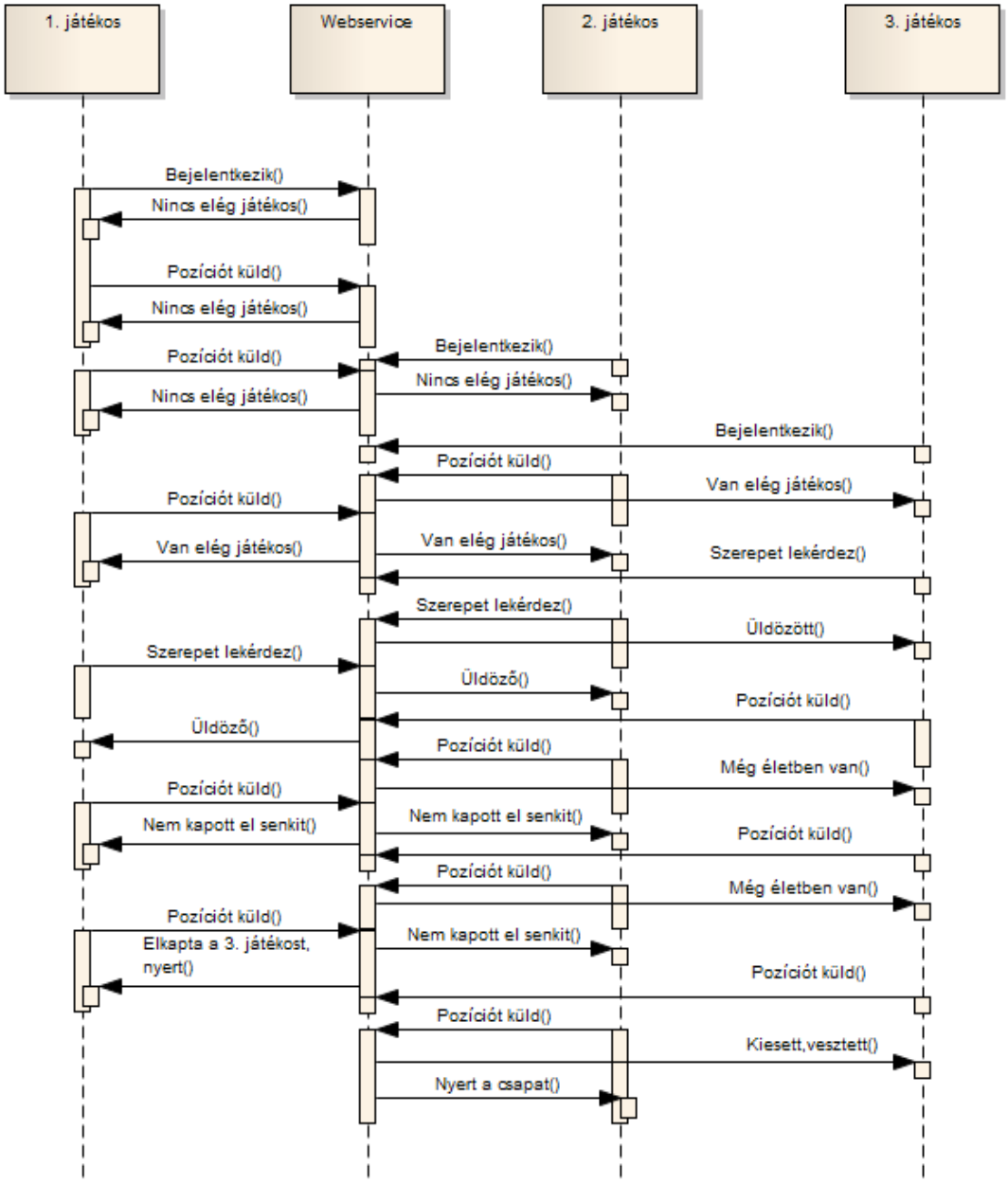
```
HttpTransportSE transport = new HttpTransportSE(URL);
transport.call(soap_action, envelope);
```

Végül a függvény visszatér egy *SoapObject*-tel, ami primitív típusokat tartalmazó

lista. Ez szétbontható, ha ismerjük előre a várt értékek típusát, illetve nevét:

```
SoapObject result = (SoapObject) envelope.getResponse();
indulhate =
Boolean.parseBoolean(result.getProperty("Canplay")
      .toString());
my_player_id =
Integer.parseInt(result.getProperty("PlayerId")
      .toString());
my_game_id =
Integer.parseInt(result.getProperty("GameId").toString());
```

A program és a webszolgáltatás kommunikációja a 18 ábrán látható.

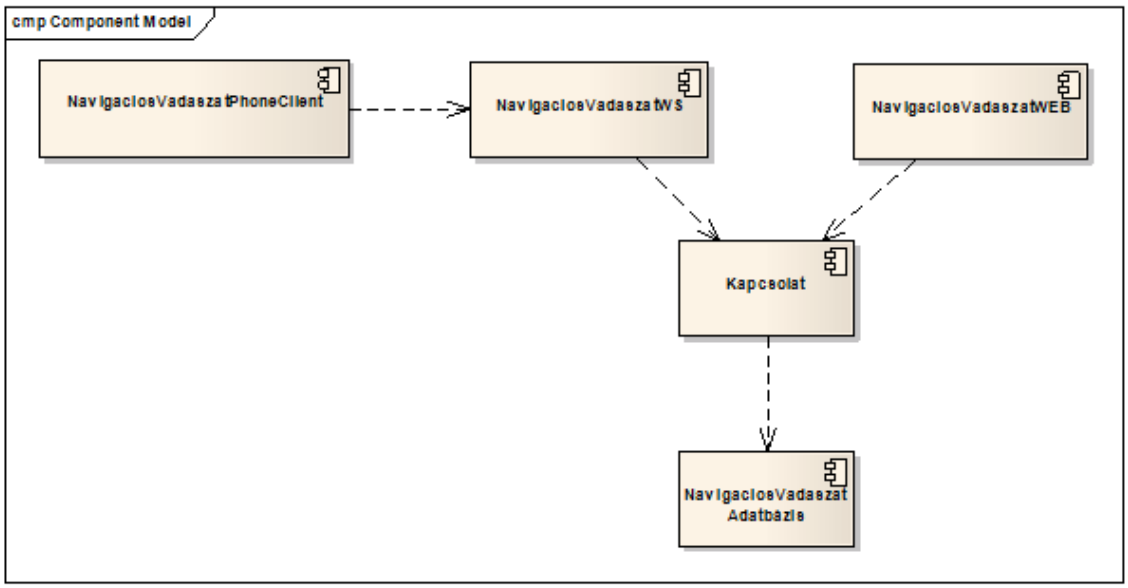


18. ábra: Szekvencia diagram

4.5. Rendszerszerkezet

A teljes rendszer az alábbi részekből áll (19. ábra):

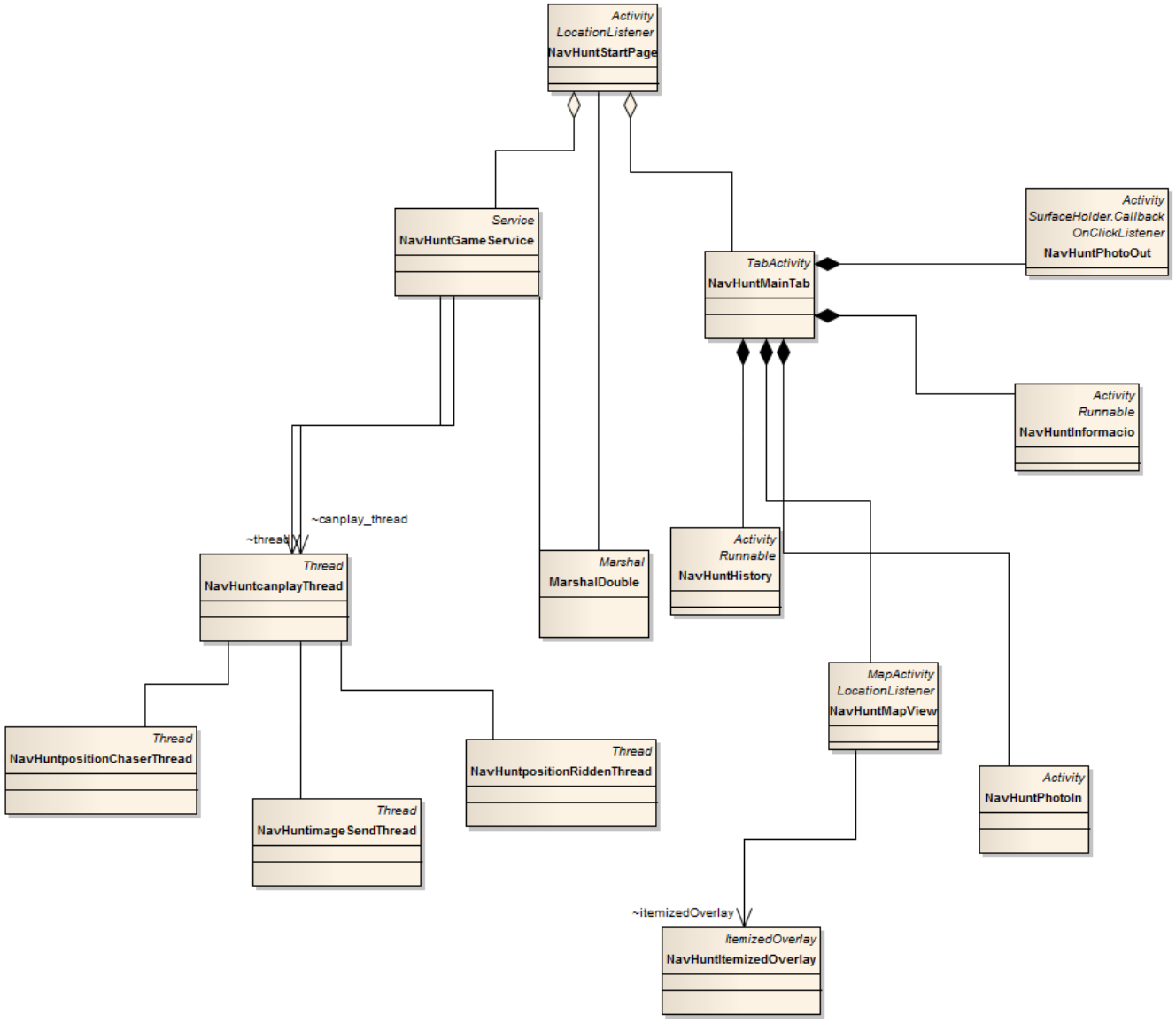
- Adatbázisból, mely a játékhoz és a weboldal használatához szükséges adatokat tárolja, illetve tartalmazza a fontosabb tárolt eljárásokat és tranzakciókat.
- Kapcsolat osztály, melyen keresztül lekérdezhető és karbantartható az adatbázis.
- Webszerver, mely a kliensek számára a játék megszervezését és lebonyolítását végzi a kapcsolat osztályon keresztül kommunikálva az adatbázissal.
- Weboldal, amin keresztül a felhasználók regisztrálni tudnak a rendszer használatához, megváltoztathatják adataikat, megnézhetik statisztikáikat és az éppen futó játékokba is betekintést nyerhetnek.
- Kliens, ami a játék felületét biztosítja. A felhasználók kliens programba való belépéssel csatlakozhatnak be a játékba. A kliens a webszolgáltatással kommunikál, így kapva információt a többi kliensről.



19. ábra: A teljes rendszer komponens modellje

4.6. A kliens program szerkezete

A program osztályait két fő csoportba lehet sorolni. Első csoport a játék felületét alkotja, ezek a következőek: *NavHuntStartPage, NavHuntMaintTab, NavHuntInformacio, NavHuntMapView, NavHuntHistory, NavHuntPhotoIn, NavHuntPhotoOut*. A másik csoport a háttérfolyamatokért felelős, úgymint a különböző szálak és a háttérben futó szolgáltatás.



20. ábra: Osztály diagram

4.6.1 Osztályok leírása

4.6.1.1. NavHuntStartPage

A *NavHuntStartPage* osztály (21. ábra) egy *Activity* leszármazott, ami a játék bejelentkező felületét alkotja. A felületen található két *EditText*, ahova a belépéshez szükséges nevet és jelszót kell megadni. Továbbá az osztály megvalósítja a *LocationListener* interfészt.

A *LocationListener* és *LocationManager* segítségével lekérhetjük az aktuális földrajzi helyzetet, de erről bővebb információ a 4.6.1.4. fejezetben olvasható.

Az *OnClickListeneren* belül kerültek megvalósításra az osztály lényegesebb részei. Ez először is meghívja a *signIn* függvényt, ami a bejelentkezésért felelős. Paraméterként megkapja a szövegmezőkbe írt felhasználónevet és jelszót, illetve az adott pozíció szélességi és hosszúsági koordinátáit. Ezeket küldi tovább a webszolgáltatásnak, ami válaszul két attribútummal tér vissza, amennyiben a megadott paraméterek helyesek voltak:

- *Indulhate* – ez egy logikai érték, ami igaz, ha van elegendő játékra várakozó bejelentkezett felhasználó a kapott földrajzi helyzet környezetében.
- *Pontosido* – Aktuális szerver idő.

Ha nem futott a program kivételbe, tehát sikeres volt az azonosítás, akkor következő lépés a szolgáltatás elindítása

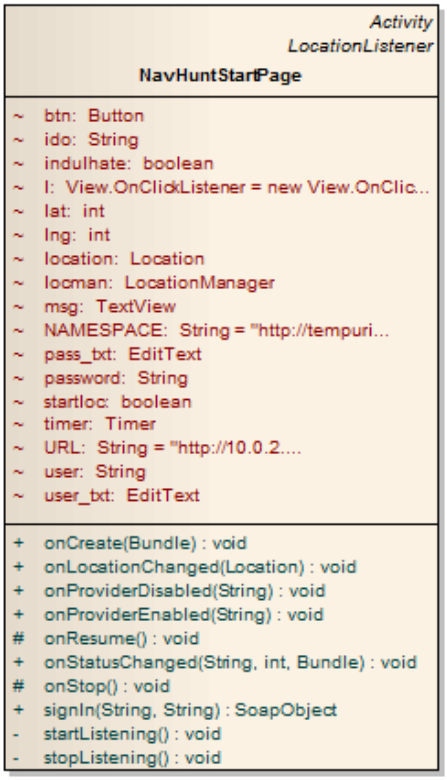
```
    Intent myIntent2 = new Intent(v.getContext(),
NavHuntGameService.class);

    startService(myIntent2);
```

Majd ezután elindul a *NavHuntMainTab Activity*, ami játék a felületét alkotó füleket kezeli:

```
    Intent myIntent = new Intent(v.getContext(),
NavHuntMainTab.class);
```

```
startActivityForResult(myIntent, 0);
```



21. ábra: NavHuntStartPage osztály

4.6.1.2. NavHuntGameService

A *NavHuntGameService* (22. ábra) a *Service* osztályból származik, ami rögtön a bejelentkezés után elindul. Feladata a háttérben futó folyamatok és szálak kezelése. Az osztály jelentősebb metódusai:

- *canPlay*

A *canPlay* függvény meghívja a webszolgáltatás *canPlay* metódusát az alábbi paraméterekkel: felhasználónév, jelszó, aktuális szélességi koordináta, aktuális hosszúsági koordináta. A válaszban visszakap egy logikai értéket, ami megmondja, hogy elindulhat-e a játék, illetve egy játékos és játék azonosítót. Az utóbbiak értéke -1 egészen addig, amíg az *indulhate* hamis. Amint elegendő ember van már a játékhoz, a webszolgáltatás kioszt minden játékosnak egy egyedi azonosítót, és elküldi annak a játékmenetnek az azonosítóját is, amibe be lett osztva az adott felhasználó.

- *PlayerType*

A *PlayerType* függvény a webszolgáltatás azonos nevű függvényét hívja meg. Paraméterként itt csak a nevet jelszót, és a játékos azonosítót kell megadni, és egy egész számot küld vissza. A visszatérési érték 0 vagy 1, ami a játékos szerepét jelöli (0: üldözött, 1: üldöző).

- *newPositionChaser*

A *newPositionChaser* függvény hasonlóan az előbbiekhez, meghívja a webszolgáltatás ugyanilyen nevű függvényét. Amennyiben a játékosnak kisorsolt szerep üldöző, úgy ez a függvény fogja küldeni percenként a koordinátáját, és az alábbi értékekkel tér vissza:

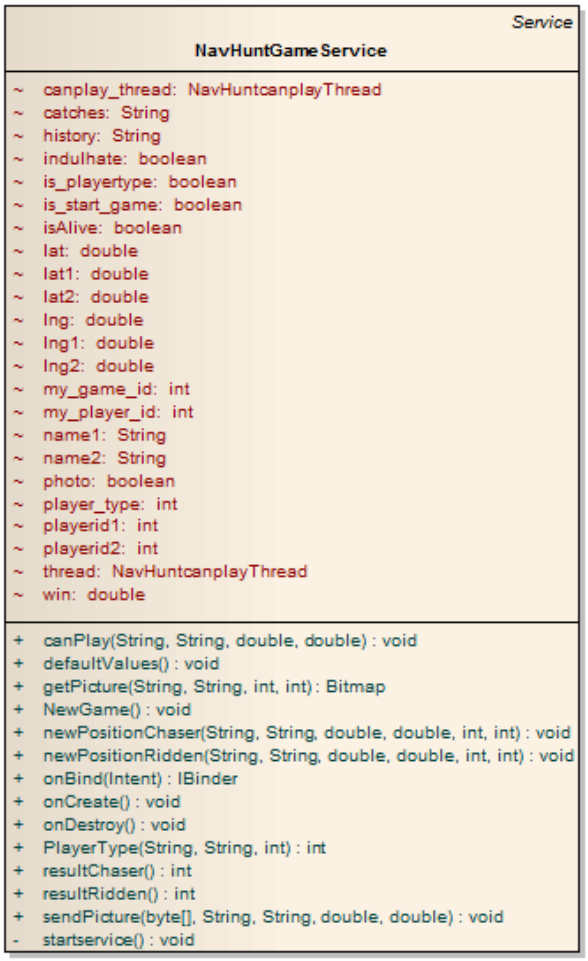
  - *Lat1:* Az egyik legközelebbi üldözött szélességi koordinátája.
  - *Lng1:* Az egyik legközelebbi üldözött hosszúsági koordinátája.
  - *Lat2:* Másik legközelebbi üldözött szélességi koordinátája.

- *Lng2:* Másik legközelebbi üldözött hosszúsági koordinátája.
- *Name1:* Az egyik legközelebbi üldözött neve.
- *Name2:* Az másik legközelebbi üldözött neve.
- *Photo:* Van-e elérhető kép az üldözöttektől.
- *Win:* Még tart a játék(0), nyert a csapat(1), vesztett a csapat(-1).
- *Catches:* A játékos által elfogott üldözött(ek).
- *Playerid1:* Ha van kép, akkor az egyik üldözött azonosítója, különben -1.
- *Playerid2:* Ha van kép, akkor a másik üldözött azonosítója, különben -1.

- *newPositionRidden*

A *newPositionRidden* az előző függvény párja, abban az esetben hívódik meg a programban ez a metódus, ha a kapott szerepe a játékosnak üldözött. Visszatérési értékek:

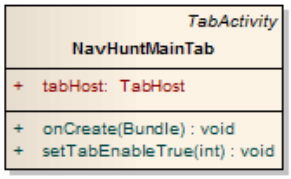
- *Lat1:* Egy üldöző szélességi koordinátája.
- *Lng1:* Egy üldöző hosszúsági koordinátája.
- *Lat2:* Egy másik üldöző szélességi koordinátája.
- *Lng2:* Egy másik üldöző hosszúsági koordinátája.
- *Name1:* Az első üldöző neve.
- *Name2:* A másik üldöző neve.
- *Win:* Még tart a játék(0), nyert a csapat(1), vesztett a csapat(-1).
- *IsAlive:* Él-e még a játékos, vagy nem.



22. ábra: NavHuntGameService osztály

4.6.1.3. NavHuntMainTab

A *NavHuntMainTab* (23. ábra) osztály egy *TabActivity* leszármazott, amelyben egy *TabHost* segítségével hozzuk létre a fülekre osztott játék felületet. Minden létrehozott fülhöz hozzárendelünk egy *Intent*-et (ami elindítja a kívánt tevékenységet), egy szöveget (ami a fül alatti felirat lesz), és egy képet (ami a hozzá tartozó ikon).



23. ábra: NavHuntMainTab osztály

```
intent = new Intent().setClass(this, Informacio.class);
spec = tabHost.newTabSpec("info").setIndicator("Adataim",
     res.getDrawable(R.drawable.infos))
          .setContent(intent);
tabHost.addTab(spec);
```

4.6.1.4. NavHuntMapView

A *NavHuntMapView* (24. ábra) a *MapActivy*-ből származtatott osztály, amelyben implementálva van a *LocationListerner* interfész. Ez alkotja a játék térképfelületét. A *MapActivity* nem része a standard *Android* könyvtárnak, ezért az *AndroidManifest* XML-ben külön definiálnunk kell:

```
<uses-library android:name="com.google.android.maps" />
```

Maga a *MapActivity* az *Activity* leszármazottja, amely tartalmazza a *Maps* könyvtárat, és a térkép kezeléséhez szükséges funkciókat, mint például a közelítés és távolítás, illetve a térkép felületére illeszthető pozíciójelzéseket. Mivel a *MapView* a *Google Maps* térképét használja, így ennek használatához internetkapcsolat szükséges, amit engedélyezni kell szintén az *AndroidManifest* fájlban.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Az osztály fontos részét képezi még a *locman:LocationManager* és a *LocationListener* attribútumok. A *LocationManager* hozzáférést biztosít az rendszer helymeghatározó szolgáltatásaihoz. Segítségével lekérhetjük a készülék aktuális helyzetét, vagy akár eseményeket is rendelhetünk egy-egy adott pozícióhoz. A *LocationListener* osztályon keresztül ezeket az információkat kezelhetjük. Egyik fontos függvénye az *onLocationChanged (Location location)* ami a szerint hívódik meg, hogy előzetesen milyen értékeket állítottunk be a *locman:LocationManager* objektumban. Ez lehet egy időintervallum (azaz egy minimum idő, aminek el kell telnie két lekérdezés között), vagy egy minimum távolság, ami méterben megadható.

Az osztály fő feladata a játékosok pozíciójának a megjelenítése a térképen. Ez úgy

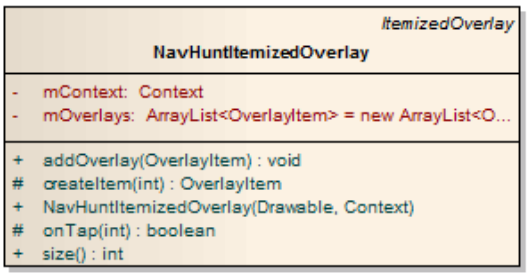
történik, hogy a térkép első betöltődésekor egy markerrel jelöli a legutóbbi ismert pozíciót, ami a bejelentkezéskor került elküldésre. Ezután minden alkalommal, amikor az *onLocationChanged* függvény meghívásra kerül, frissíti a térképen a felhasználó földrajzi helyzetét, illetve, ha már van adat a többi játékosról, akkor az ő helyüket is jelzi a térképen.

Mivel a térképen nem minden esetben állapítható meg pontosan, hogy melyik játékos milyen messze van, így a felhasználó pontos információt kaphat erről, ha rákattint egy másik játékost jelölő markerre. Ekkor egy felugró felületen láthatóvá válik az adott játékos neve, és a légvonalbeli távolsága méterben.

24. ábra: NavHuntMapView osztály

Érdekesség, hogy a térképfelületen megjelenítendő pontok nem a szokásos **`double`** típusú értékeket használják, mint minden más *Google Maps* felület, hanem úgynevezett *GeoPointokat*. A *GeoPoint* szintén egy hosszúsági és egy szélességi koordinátából áll, de ezek egész számok, az eredeti **`double`** értékek egymillióval felszorozva.

4.6.1.5. NavHuntItemizedOverlay



25. ábra: NavHuntItemizedOverlay osztály

A *NavHuntItemizedOverlay* (25. ábra) egy *ItemizedOverlay*-ből származtatott osztály, amely segítségével markereket rendelhetünk pontokhoz, amit a térképfelületen megjeleníthetünk, ezenfelül megadható, hogy a markerrel való kattintás esetén mi jelenjen meg felugró felületen.

4.6.1.6. NavHuntHistory

A *NavHuntHistory* (26. ábra) osztály az események megjelenítéséért felelős. Itt jelennek az aktuális játékmenettel kapcsolatos információk. A játékos láthatja, hogy mikor kezdődött el a játék, melyik játékmenetbe lett beosztva, a kapott szerepét, és a szereptől függően további információkat is.

26. ábra: NavHuntHistory osztály

4.6.1.7. NavHuntInformacio

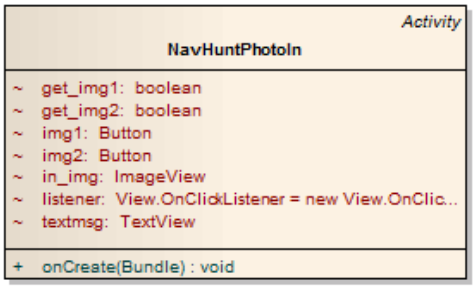
A *NavHuntInformacio* (27. ábra) osztály a személyes adatokat jeleníti meg, mint a felhasználónév, a játékos azonosítója, és betöltött szerepe. Ezen kívül mutatja a legközelebbi játékos távolságát és nevét is. Ennek és az előző osztálynak is az adatai fél percenként frissülnek.

27. ábra: NavHuntInformacio osztály

4.6.1.8. NavHuntPhotoIn

Ez az osztály (28. ábra) alkotja az üldözőfelületet, ahol a játékos lekérheti a legközelebbi üldözöttek aktuális helyzetéről készült fényképeket. Két gombbal lehet váltani a képek között. A gomb megnyomása után három eset áll fenn:

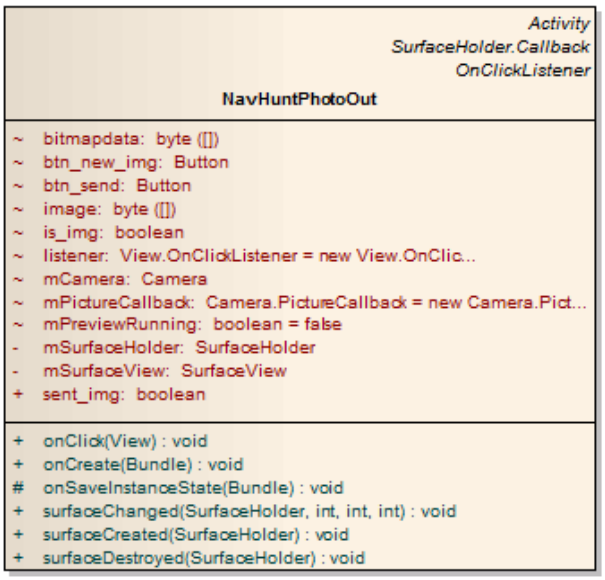
- Még nincs kép, ilyenkor a szövegdobozban megjelenik erről egy üzenet.
- Friss kép érkezett. A gombhoz tartozó *OnClickListener* meghívja a webszolgáltatás *GetPicture* függvényét, és lekéri a legújabb képet
- Már érkezett kép, de régebben – A legutóbbi képet tölti be, nem küld külön kérést ezért a webszolgáltatáshoz



28. ábra: NavHuntPhotoIn osztály

4.6.1.9. NavHuntPhotoOut

Ez az osztály az üldöző nézet (29. ábra), melynek feladata, hogy az üldöző játék közben bizonyos időközönként képet készíthessen, amit aztán elküld. A készített fotó egy **`byte`** tömb, ez adódik át egyenesen a webszolgáltatásnak, ami elküldi az üldözőnek, ha érkezik kérés. A képet külön nem tárolja a memóriakártyán.



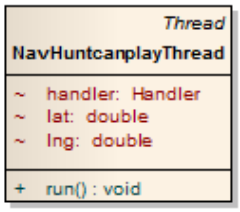
29. ábra: NavHuntPhotoOut osztály

4.6.2. Szálak

A program több szálat használ a webszolgáltatással való kommunikáció lebonyolításához. A szálak ciklikusan futnak, amely kezeléséhez Handler objektumokat használ. A *Handler* segítségével megadható, hogy mennyi idő elteltével fusson le megint a szál *run* metódusa.

4.6.2.1.NavHuntCanPlayThread

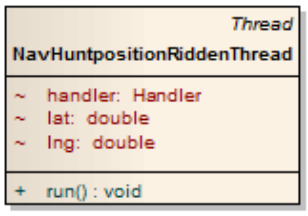
A *NavHuntGameService* (30. ábra) osztály *startservice* metódusában indul el ez a szál, melynek feladata, hogy a *NavHuntGameService canPlay* függvényét percenként meghívja, így folyamatosan küldve a webszolgáltatásnak a pozíciót. Miután a webszolgáltatás által küldött *Indulhate* változó értéke igazra vált, úgy a szál meghívja a *playerType* függvényt, és ennek eredményétől függően a *NavHuntPositionRiddenThreadet*, vagy a *NavHuntPositionChaserThreadet* indítja el, majd ezt követően leáll.



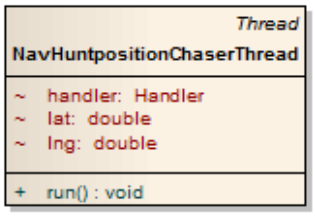
30. ábra: NavHuntCanPlayThread osztály

4.6.2.2. NavHuntPositionRiddenThread, NavHuntPositionChaserThread

Meghívja a *NavHuntGameService* azonos nevű függvényét percenként (31-32. ábra), és frissíti az Előzmények felületen található információkat.



31. ábra:
NavHuntPositionRiddenThread osztály



32. ábra:
NavHuntPositionChaserThread osztály

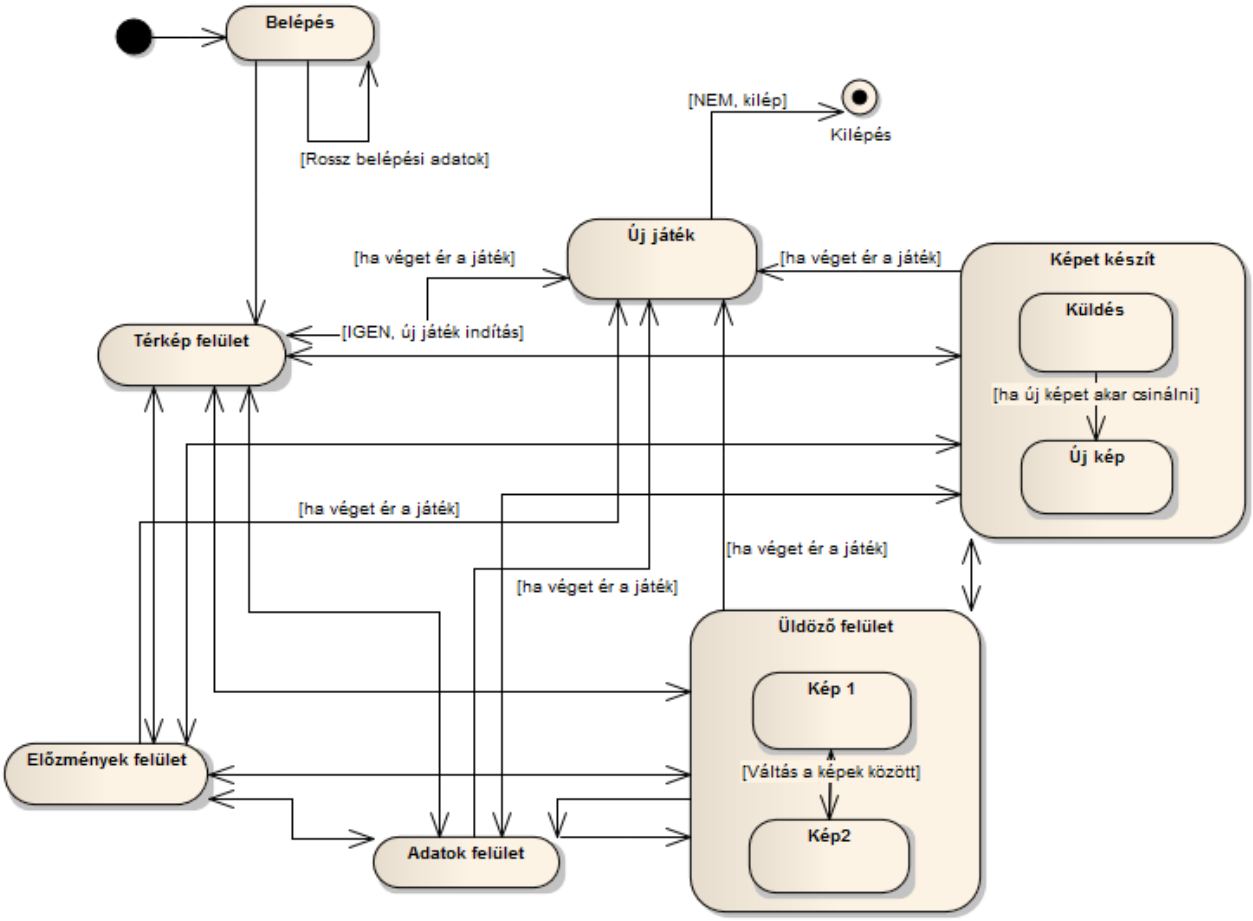
4.6.2.3. NavHuntImageSendThread

A *NavHuntcanPlayThread* indítja el ezt a szálat, amint elkezdődött a játék. Feladata, hogy tíz percenként figyelmeztető üzenetet jelenítsen meg az *Előzmények* felületen.(33. ábra)

33. ábra:
NavHuntImageSendThread osztály

4.6.3. A játék állapotai

A játék állapotait a 34. ábra mutatja be. Bejelentkezés után (amennyiben helyes adatokat adtunk meg) a térkép felület jelenik meg, ahol a játékos nyomon követheti saját és ellenfelei földrajzi helyzetét is. A továbbiakban tetszőlegesen lehet váltani a fülek (a különböző felületek) között aszerint, hogy éppen mit szeretne tenni a felhasználó. Ezt mindaddig megteheti a játékos, amíg véget nem ér a játék, amikor is egy felugró ablak jelenik meg. Ebben az ablakban lehet választani, hogy, kilép, vagy csatlakozik egy új játékhoz.



34. ábra: Állapotdiagram

4.7. Tesztelés

A tesztelés javarészt emulátorok segítségével történt. Fontos volt, hogy a tesztelés során minden eshetőséget kipróbáljunk, hogy aztán a hibákat le lehessen kezelni. Az erőforrások végessége miatt egyszerre 3-4 játékosra tudtuk vizsgálni, hogy mi történik a program futása alatt, de a rendszer ugyanúgy működik több játékos esetén is.

4.7.1. Fehér doboz tesztelés

- **Hibás bejelentkezési adatok**

Hibás bejelentkezési adatok esetén a bejelentkezés sikertelen. A webszolgáltatás egy kivétellel tér vissza, amit a program elkap, és a bejelentkező felületen megjeleníti annak tartalmát.

- **Az adatbázis nem elérhető**

Ha az adatbázis nem elérhető a bejelentkezésnél, akkor a program még egy ideig próbálkozik, majd egy időkorlát után „Connection timed out” hibaüzenetet kap, amit a bejelentkező képernyőn megjelenít.

- **Webszolgáltatás elérése lokális gépen**

A webszolgáltatás lokális elérést általában egyszerűen meg lehet tenni, elég megadni a webszolgáltatás címét a következő módon: **`http://localhost:<port>/Service1.asmx`**

Míg ez a módszer a legtöbb esetben működik, addig az Android emulátoron nem, ha ugyanis emulátorba teszteljük a programunkat, ami ilyen módon kívánja elérni a webszolgáltatást, akkor hibát kapunk. Ez azért történik, mert az emulátor egy virtuális eszköz, ami számára a „localhost” önmagát jelenti.

- **Helyes belépési adatok**

Helyes adatok esetén a bejelentkezés sikeres, amennyiben a webszolgáltatás is hibátlanul működik.

- **Nem megfelelő formátumú adatok a webszolgáltatástól**

Ha a program futása közben a webszolgáltatás valamilyen oknál fogva nem olyan adatokat ad vissza, mint amit a kliens vár, akkor *SoapFault*-ot kapunk. Ez egy speciális SOAP-os kivétel. Ebben az esetben a program fut tovább, annyi különbséggel, hogy nem sikerült (új) értékeket adni bizonyos változóknak.

4.7.2. Fekete doboz tesztelés

- **Bejövő hívás játék közben**

Bejövő hívás esetén a híváskezelő felület lesz az aktív, de a háttérben a program szálai futnak tovább, így mikor a játékos befejezte a beszélgetést, visszatérhet a játékba.

- **Kikapcsolt GPS vevő**

Kikapcsolt GPS vevő esetében nem lehetséges a belépés a játékba, hiszen legalább egy ismert koordináta kell, hogy az bejelentkező gomb aktív legyen. Ha ezután kapcsoljuk ki a GPS vevőt, akkor a legutóbbi ismert pozíció kerül elküldésre a webszolgáltatásnak. Ugyanez a helyzet akkor, ha olyan helyre megyünk, ahol nincs GPS jel.

- **Internetkapcsolat elvesztése**

Ha elveszítjük az internetkapcsolatot, tehát a program több mint két percig nem tud küldeni pozíciót, akkor a webszolgáltatás inaktívvá nyilvánítja a játékost, és kidobja az aktuális játékmenetből. Ha két percen túl létrejön ismét az internetkapcsolat akkor sem léphet vissza abba a játékmenetbe, amelyből előtte volt.

- **Hibás bejelentkezési adatok**

Rossz felhasználó/jelszó páros esetén a belépés sikertelen.

- **Üldöző felület használata a játékostípus kiosztása előtt**

Ha a játékos azelőtt akarja lekérni a fényképeket, hogy ki lettek volna osztva a

játékosoknak a típusok, akkor gombok megnyomásával csak egy szöveg fog megjelenni, ami figyelmezteti erre.

- **Üldözött felület használata a játékostípus kiosztása előtt**

Ha a játékos fényképet akar készíteni, és elküldeni azelőtt, hogy megkapta volna a szerepét, akkor megjelenik egy szöveg, ami figyelmezteti arra, hogy erre nincs lehetősége egyelőre.

- **Nem jogosult használata az üldöző felületnek**

Ha egy üldözött szeretné ezt a felületet használni, akkor a program nem engedi, és a gombok lenyomása által csak egy üzenetet kap, miszerint ezt az üldözők használhatják csak.

- **Nem jogosult használata az üldözött felületnek**

Egy üldöző nem készíthet képet az üldözött felületen, mert amint a Küldés gombra kattint, egy szöveg figyelmezteti erről.

- **Kilépés a programból játék futása közben (Program leállítása)**

Ebben az esetben két lehetőség van.

  - Ha a felhasználó úgy lépett ki a programból, hogy teljesen leállította (például Task Managerben) akkor a webszolgáltatás két perc után ki fogja dobni őt a játékosok közül, így nem zavarja a többiek további játékát.

  - Ha csak a játék felületet zárta be, például azzal, hogy a HOME gombra lépett, akkor a folyamatok a háttérben továbbra is futnak, és továbbra is aktív része játékmenetnek.

5. Összegzés

A szakdolgozat célja egy szórakoztató, izgalmas közösségi játék megírása volt, ami kimozdítja az embereket a számítógép elől. Azáltal, hogy *Android* platformra íródott, sok ember számára elérhető, hiszen az okostelefonok nagyon elterjedtek, melyek között az egyik legnépszerűbb ez az operációs rendszer.

A játékot játszhatja egy adott területen összegyűlt baráti társaság, vagy bárki, akinek éppen kedve támad, ha a környéken akadnak még elegen egy játék elindításához. A játék, hosszából adódóan-, lehet csak egy kis délutáni kikapcsolódás a szabad levegőn az ismerősökkel, vagy akár egész napos verseny nagyobb társaság esetén. Ebből adódóan remek szórakozás lehet bárki számára, aki szívesen tölti az idejét kinn a szabadban, vagy csak mozgásra vágyik, hiszen a játék egy jó lehetőséget kínál arra, hogy kimozduljunk, és megismerjük környezetünket.

A megírt program a kitűzött céloknak megfelel teljes egészében, de megvalósítás szempontjából kicsit eltér a kezdeti tervektől. Ilyen például a webszolgáltatással való kommunikáció, amit eredetileg cookie-k segítségével történt volna, de a *kSOAP* könyvtárcsomag erre nem adott lehetőséget. A problémát úgy kellett áthidalni, hogy minden kérésnél a webszerver azonosítja a klienst, majd a sikeres azonosítás után hajtja csak végre a kérést.

A program megvalósítása során sok új ismeretet szereztem, hiszen előtte nem foglalkoztam *Androidos* alkalmazásfejlesztéssel. Ebből adódóan a megvalósítás hosszabb folyamat volt, mert el kellett sajátítani a szükséges háttértudást is. A program sokrétűsége miatt lehetőségem volt mélyebben is megismerni az *Androidot* (GPS-es helymeghatározás, kamera használat, háttérben futó szolgáltatás készítése stb..) Elégedett voltam a fejlesztőkörnyezettel és az *Android* nyújtotta lehetőségekkel. Hiányosságként – a már korábban említett – webszolgáltatással való kommunikációs eszközöket tudnám megemlíteni.

Továbbfejlesztési lehetőségek:

- Privát játékok: Amikor egy adott társaság tagjai szeretnének egymás ellen játszani, „kívülállók” nélkül.
- Rugalmasan beállítható játékidő (Privát játék esetén).
- Nagyobb mennyiségű játékos esetén több egymás ellen játszó csapat.
- Globális chat felület biztosítása a játékban az egyes csapatok számára.
- Privát csevegés egy-egy másik játékossal.
- Csapatok regisztrálása: Állandó csapatok játszhatnának egymás ellen, így külön csapatokra vonatkozó statisztikát is lehetne vezetni, nem csak egyénit.
- Többnyelvűség: A felhasználónak bejelentkezéskor lehetősége lenne nyelvet választani, hogy a játék folyamán az aktuális információkat a kiválasztott nyelven kapja meg.

Irodalomjegyzék

[1] Wikipedia: **http://en.wikipedia.org/wiki/Android_%28operating_system%29**

Dátum: 2011.03.05.

[2] Android Hungary: **http://androidhungary.com/2010/09/android-fejlesztoi-lecke-%E2%80%93-az-android-os-rovid-tortenete-es-felepitese/**

Dátum: 2011.04.20.

[3] Wikipedia: **http://en.wikipedia.org/wiki/Smartphone**

Dátum: 2011.03.07.

[4] Android Developers: **http://developer.android.com/sdk/index.html**

Dátum: 2011.04.20.

[5] Wikipedia: **http://en.wikipedia.org/wiki/Java_%28programming_language%29**

Dátum: 2011.04.23.

[6] Wikipedia: **http://en.wikipedia.org/wiki/Google_Maps**

Dátum: 2011.04.24.

[7] Robert Englander: Java and SOAP, O'Reilly, 2002, [276],

ISBN: 0-596-00175-4

[8] James Steele Nelson To: The Android Developer's Cookbook, Addison-Wesley, 2010, [339],ISBN-10: 0-321-74123-4