Eötvös Loránd Tudományegyetem
Informatikai Kar
Programozáselmélet és
Szoftvertechnológiai tanszék

Androidos GPS-es játékalkamazás

Dr. Istenes Zoltán
egyetemi docens

Csatári Albert
programtervező informatikus

Budapest, 2014. 04. 26.

Tartalomjegyzék

1. Bevezetés

A mai korban az okostelefon az egyik legmeghatározóbb eszközzé nőtte ki magát. Lassan mindenkinek a tulajdonában lesz egy, és ez a tendencia Magyarországra is igaz. Én ezen felbuzdulva választottam a mobil platformot, ezen belül is az Android operációs rendszert a nagy felhasználótábora miatt.

A feladat az, hogy az okostelefon felhasználója kihasználja az eszköze által nyújtott szolgáltatásokat, az én esetemben a helymeghatározást, az internetelérést és a mobilitást.

A My Little Fellow nevű játék hőse egy 300000 évvel ezelőtt élő ősember. Az ő világa néhány dologban megegyezik a miénkkel, ezek közül a legfontosabb a látnivalók területi elhelyezkedése. Ezeknek a pontoknak a fizikai meglátogatása kulcsfontosságú a játék szempontjából és egyben a felhasználónak is helyismeretet tanít.

A My Little Fellow játék tehát két részre bontható: Az első részben a játékos fizikailag elmegy a térkép minél több részére és mellette felfedezi az ott található látványosságokat, a másik részben pedig kihasználja a játékban a megismert területeket, ugyanis itt nyersanyagokat gyűjthet, ezekből eszközöket, épületeket építhet. De itt nincs vége, mert a nyersanyagok végesek és az építkezések sem olcsók, ezért még többet kell utazni a fizikai világban és felfedezni. Ennek a két résznek a váltakozásán alapszik a játék.

A játékos célja az, hogy egy területet teljesen beépítsen, várossá váljon és ő legyen a földesura. Az én célom pedig az, hogy a játékos a már meglévő modern eszközét kihasználva megismerje környezetét.

2. Felhasználói dokumentáció

2.1. A játék célja

A My Little Fellow célja az, hogy minél több területet és látványosságot látogassanak meg a felhasználók, idővel felfedezve az egész országot. Ezzel megismerhetik azokat az érdekességeket, amik mellett lehet, hogy eddig csak egyszerűen elsétáltak.

Egy felhasználó feladata a következő: az eszközzel felfedezni, szimpatikus területet kiválasztani, nyersanyagokat gyűjteni, építkezni. Végül az első épületet – a városházát – felfejleszteni a maximális szintre, és ezzel megalapítani azt a várost, amiben a felhasználó lesz a földesúr.

2.2. A játék menete

2.2.1. Alapok

A játékba belépés után a térképen találjuk magunkat. Jobb felül három gomb jelenik meg. Ezek a Google Maps ki- és bekapcsolása, a Felfedező mód és a Karakterre ugrás. Belépés után a térkép automatikusan ki van kapcsolva - ezzel csökkentve az adatforgalmat – kapcsoljuk be, ha lényegtelen az adatforgalom mennyisége. Első belépéskor a Felfedező mód be van kapcsolva. Ilyenkor az eszköz GPS szenzora működésbe lép, és megpróbálja meghatározni az első pontot, ahova bejelentkezünk. Amint ez sikerül, megjelenik a karakter a térképen ott, ahol a felhasználó is éppen tartózkodik és egy színes téglalap körülötte. Ez a téglalap határozza meg a területet, a színe pedig annak a típusát. Ameddig a Felfedező mód bekapcsolva marad, addig keresi a telefon a tartózkodási helyünket, és ha új, felfedezetlen területre érkeztünk, azt felfedezi.

A térképen megjelennek további kék jelölőpontok. Ezek a pontok jelölik a valóságban megtalálható látványosságokat. Ezek lehetnek akár múzeumok, kastélyok vagy szobrok, útszéli keresztek. A jelölőpontra való kattintás után több információt tudhatunk meg róla. Ha ezeket a pontokat meglátogatjuk – ami azt jelenti, hogy a hatósugarában, vagy 15 méter távolságra tartózkodunk felfedező módban – akkor kapunk egy intelligencia pontot. Ezek a pontok szükségesek lesznek később az épületek, eszközök továbbfejlesztéséhez.

Ha a felfedező módot kikapcsoljuk az ikonjával, akkor a karakter azon a területen marad, ahol utoljára volt. Innen el tud utazni másik, már felfedezett területre anélkül, hogy mi a tényleges helyszínre mennék.

2.2.2. A terület

A színes téglalappal való területről több információt tudunk meg, ha rákattintunk. Először a térképen jelenik meg egy kis ablak néhány információval, majd erre az ablakra kattintva jön be a terület oldala. Innen van lehetőségünk a játékot folytatni további interakciókkal. A területeken, a típusától függően, lehet nyersanyagokat szerezni. Ezek szükségesek lesznek később épületek és eszközök kifejlesztésére, építésére.
Egyik területről a másikra utazás is itt lehetséges. Ennek az ikonja akkor jelenik meg, ha nem arra a területre léptünk be, amelyiken éppen tartózkodik a karakterünk.

2.2.3. A letelepedés

Ha kiválasztottunk egy nekünk szimpatikus területet, akkor menjünk oda a karakterünkkel, lépjünk be a területre és nyomjuk meg a letelepedés ikonját. Idővel a művelet befejeződik, és a területet megjelöljük az otthonunknak. Ez után indulhat az építkezés ezen a területen.

2.2.4. Építkezés, nyersanyaggyűjtés

A letelepedés után indul el a játékra jellemző kör: nyersanyag és intelligencia pont gyűjtése, épület kifejlesztése, épület építése. Néhány épületet meg tudunk rögtön építeni, de a legtöbbhez fejlesztés szükséges. A fejlesztés a Tudás Tornyában lesz elérhető, ezt is szükséges megépíteni.

2.2.5. Földesúrrá válás

Ahhoz, hogy elérjük a legnagyobb rangot egy területen, meg kell építeni a városházát a legmagasabb szintűre. Ekkor a terület automatikusan várossá alakul, és elérhetővé válik az adóztatás. Ezt az adót mindenkinek ki kell majd fizetnie, aki a területre lép, és a befolyt nyersanyagokat a földesúr kapja meg.

2.3. Rendszerkövetelmények

Minimum követelmények az Androidos készülékhez:

- 2.2-es Android verzió vagy annál magasabb
- GPS szenzor
- Internetelérés
- 15 MB szabad hely

Ajánlott követelmények:

- min. 800 x 480 felbontás
- GPS szenzor GLONASS támogatással
- mobilinternet
- 50 MB szabad hely
- min. ARM v7 processzor

2.4. Telepítés

A játék a Play Store-ból telepíthető, a következő linkről:
https://play.google.com/store/apps/details?id=hu.jex.mylittlefellow
A játék megtalálható a My Little Fellow név alapján kereséssel is.
Ezen kívül a játék telepíthető a mellékelt „apk” kiterjesztésű fájl alapján is a következő módszerrel:

1. Másoljuk fel a fájlt a telefonra
2. Keressük ki a fájlt a telefonon és indítsuk el
3. Ha kéri, akkor engedélyezzük az ismeretlen alkalmazások telepítését a Beállítások menüben
4. Kattintsunk a telepítés gombra

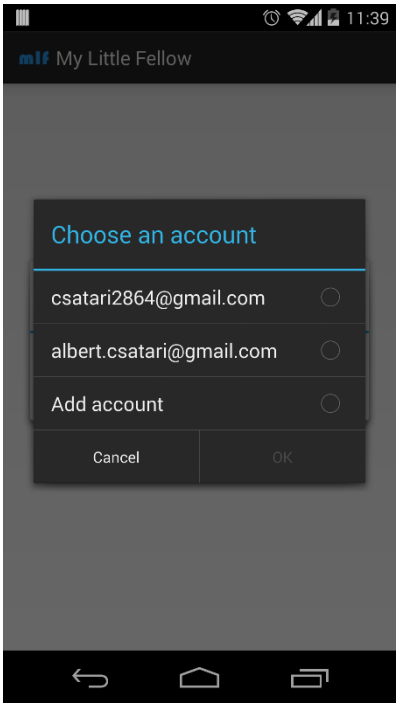
2.5. Regisztrálás és bejelentkezés

A játék az azonosításhoz a Google fiók e-mail címét veszi igénybe. Első bejelentkezéskor kell kiválasztani, hogy melyik e-mail címet szeretnénk használni (ha csak egy van, akkor automatikusan azt választja a rendszer). Ez után kér egy felhasználónevet és kétszer a jelszót. Ha még nem létezik a fiók, akkor az elküldés után regisztrálja a megadott adatokkal a felhasználót, azonban ha már volt regisztrálva felhasználó az e-mail címmel, akkor ellenőrzi a bevitt adatokat, és ha azok helyesek, bejelentkezteti a felhasználót.
Bejelentkezés után szinkronizálás is történik, az összes a kliensnek szükséges adat letöltődik. Ezért, ha a felhasználó lecseréli az eszközét, vagy valamiért letöröli a programot és újratelepíti, az adatai nem vesznek el.
Az adatok kitöltése után a program nem kéri többször a felhasználónév/jelszó párost a bejelentkezéshez, csak akkor, ha valamilyen oknál fogva a megadott adatok eltűntek.

2.6. Egy lehetséges játékmenet

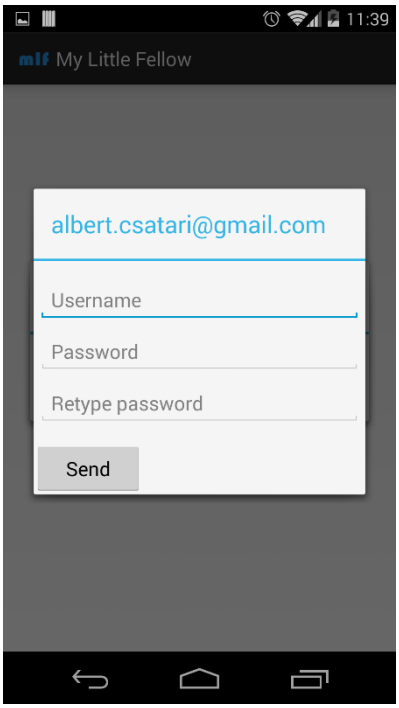
2.6.1. Regisztrálás és bejelentkezés

A telepítés után a játékba belépéskor a regisztráláshoz szükséges e-mail címet kell kiválasztani. Csak olyan e-mail címet lehet kiválasztani, amit az eszközhöz fiókként hozzáadtunk. Ha csak egy fiókot használunk az eszközön, akkor ezt a választási lehetőséget nem ajánlja fel a játék, a választás automatikusan történik.



2.1. ábra – E-mail cím választás

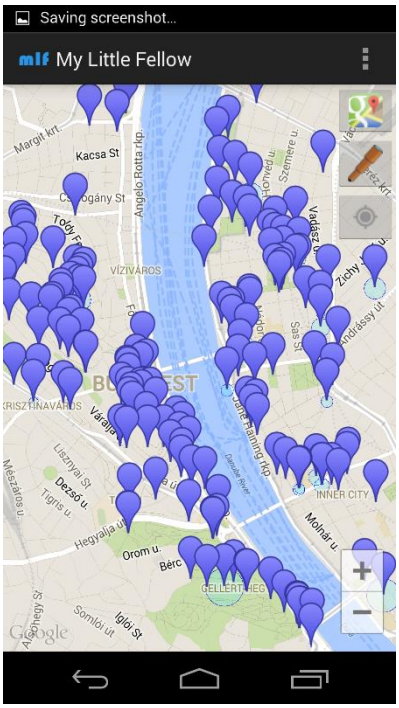
Az e-mail cím kiválasztása után a kötelező regisztrációs adatokat kell megadni. Ahogy a Regisztrálás és bejelentkezés alfejezetben is meg van írva, ha az e-mail címhez már tartozik egy regisztráció, akkor az adatok megadása után ennél a pontnál csak bejelentkezés és szinkronizálás történik. A regisztrációs ablakról a 2.2. ábrán látható képernyőkép.



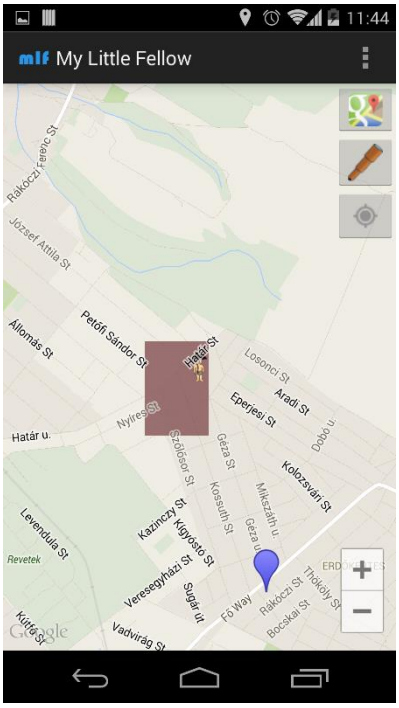
2.2. ábra – Regisztrációs ablak

2.6.2. Térkép

Az első bejelentkezés után a térkép ablakhoz kerülünk. A felfedező üzemmód automatikusan bekapcsolt, lévén, hogy még nincs semmilyen helyszíni adatunk a karakterről. Miután a GPS megtalálta a pontos pozíciónkat, a térkép a karakterünkhöz ugrik, és a területet rögtön fel is fedezi.



2.3. ábra – első bejelentkezés utáni látvány, Budapest középponttal és az összes látványossággal



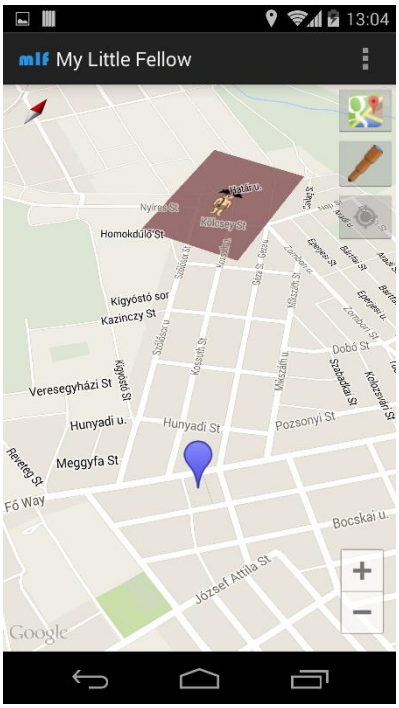
2.4. ábra – a helymeghatározás után a sikeres terület felfedezés

A felület jobb felső részében három gomb található. A legfelső gombbal a térképet tudjuk ki- és bekapcsolni, kikapcsolt állapotban az adatforgalom a töredékére csökkenthető. A második gomb, ami egy távcsővel van szimbolizálva, a felfedező módot kapcsolja ki és be. A harmadik gomb megnyomása után a térkép a karaktert helyezi középpontba.

A térképen elérhetőek a más programokban megszokott gesztusok:

- A térkép mozgatása érintéssel
- Nagyítás és kicsinyítés két ujj szét- vagy összehúzásával, illetve egy ujj duplakattintásával, majd mozgatásával
- Térkép döntése két ujj párhuzamosan le- vagy felhúzásával

A térképen mindig csak azok az információk jelennek meg, amik a felhasználó által nézett térképrészletben szerepelnek. Egy lehetséges elforgatott és eldöntött térképet a 2.5. ábrán láthatunk.

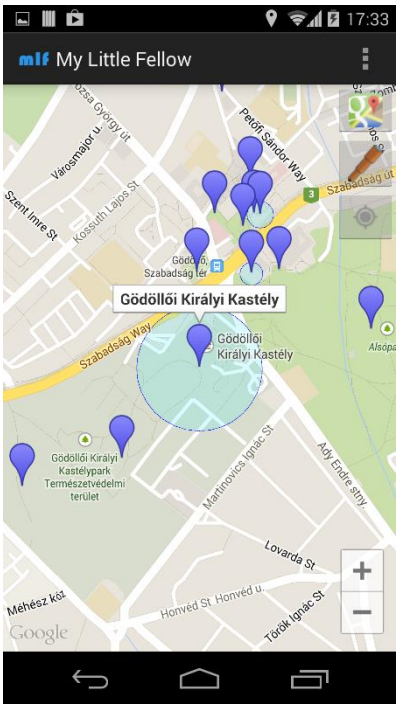


2.5. ábra – Elforgatott és eldöntött térkép

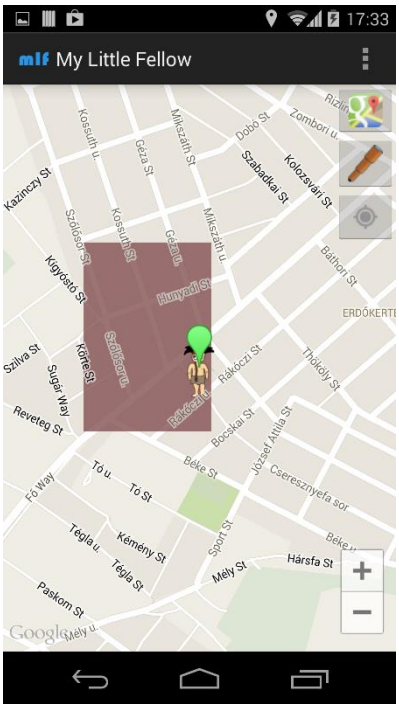
2.6.3. Felfedező mód

A játékot két részre lehet osztani, amikor aktívan játszunk, és amikor passzívan. Az aktív játék a térkép felfedezését jelenti: új területek bejárását és látványosságok meglátogatását. A passzív játék a maradék részt, amikor a már felfedezett területeken nyersanyagokat gyűjtünk, letelepedünk, építünk. Ha a felfedező mód be van kapcsolva, akkor az aktív játék működik. Az első játéknál érdemes bejárni pár területet, majd a látványosságoknak is utána nézni.

Egy látványosságot a játékban ott találunk meg, ahol a valós világban is megtaláljuk. Két féle típus létezik ezekből: a nagy kiterjedésű, ez egy jelölőponttal és egy hozzá tartozó körrel van jelölve; illetve a pontszerű, ami csak egy jelölőponttal szerepel a térképen. A még fel nem fedezett látványosságok kék színnel, a már felfedezettek zöld színnel vannak jelölve. Keressünk fel pár ilyen pontot. Érdekességként szolgál, ha a térképen a jelölőpontra kattintunk, mert ilyenkor információt kapunk a ponton található különlegesség nevéről (illetve, ha nincs név megadva, akkor a típusáról). Erről egy példát a 2.6. ábrán lehet látni. Továbbá a névre kattintva, a játék a nevezetesség weboldalára ugrik. Ha nincs ilyen csatolva, akkor egy Google keresést indít el.



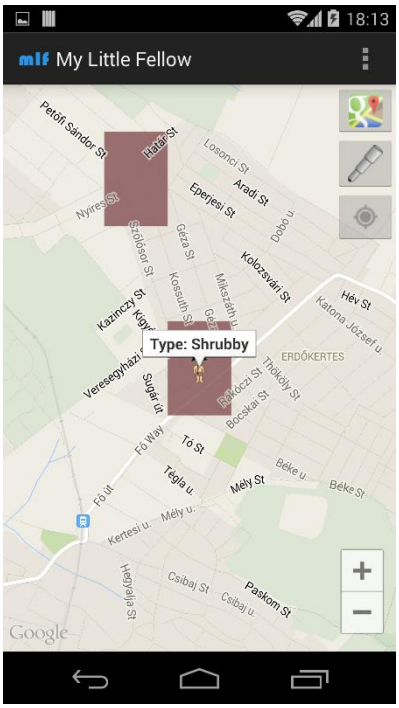
2.6. ábra – A látványosságra kattintva megjelenik annak a neve. A névre kattintva pedig a weboldala jelenik meg



2.7. ábra – A látványosság felfedezés után zöld színűre változik

2.6.4. Terület

Ha elég területet felfedeztünk, kapcsoljuk ki a felfedező módot, majd kattintsunk arra a területre, ahol éppen tartózkodik a karakter. Ekkor megjelenik az információ, hogy milyen típusú területről van szó.



2.8. ábra – Területre kattintás után megjelent információ, itt egy bozótos típust láthatunk

A játékban a következő típusok szerepelnek:

- Erdő
- Puszta
- Bozótos
- Sivatag
- Kőbánya
- Tó
- Folyó
- Település

Minden egyes típuson más nyersanyagokat lehet kibányászni, kitermelni. A következő nyersanyagok léteznek a játékban:

- Fa
- Gally
- Hús
- Bogyó
- Kő
- Hal

Ha a megnyílt információra kattintunk, akkor a terület oldalára kerülünk.

2.9. ábra – Bozótos, elérhető a megvizsgálás, bogyószedés, gallygyűjtés és a letelepedés

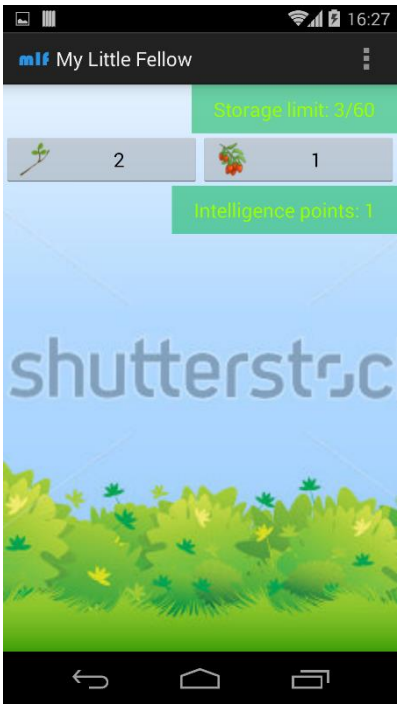
A 2.9 ábrán is láthatjuk, hogy egy terület két részre tagolódik. A bal oldalon a területen található nyersanyagokkal kapcsolatos interakciók jelennek meg, a jobb oldalon pedig az épületekkel kapcsolatos interakciók. Először vizsgáljuk meg a területet, hogy pontos információt kapjunk az ott található nyersanyagok számáról. Ehhez a távcső ikonra kell kattintani. Itt ismerkedünk meg először a visszaszámlálóval, amely majdnem minden oldalon jobb felül található.

2.10. ábra – Megvizsgálás, jelenleg 4 mp van hátra

A számláló lejárta után megtörténik a megvizsgálás és megjelennek jobb oldalon a területen található nyersanyagok.

2.11. ábra – Megtörtént a megvizsgálás, 27 bogyó és 10 gally található itt

Kezdjünk el bogyókat szedni, nyomjunk rá a bogyó ikont tartalmazó gombra bal oldalon. Ha befejeztük szedjünk össze gallyakat is. Ez után ellenőrzésképpen menjünk be a raktárba. A raktár jobb oldalon található egy doboz ikonnal jelölve.

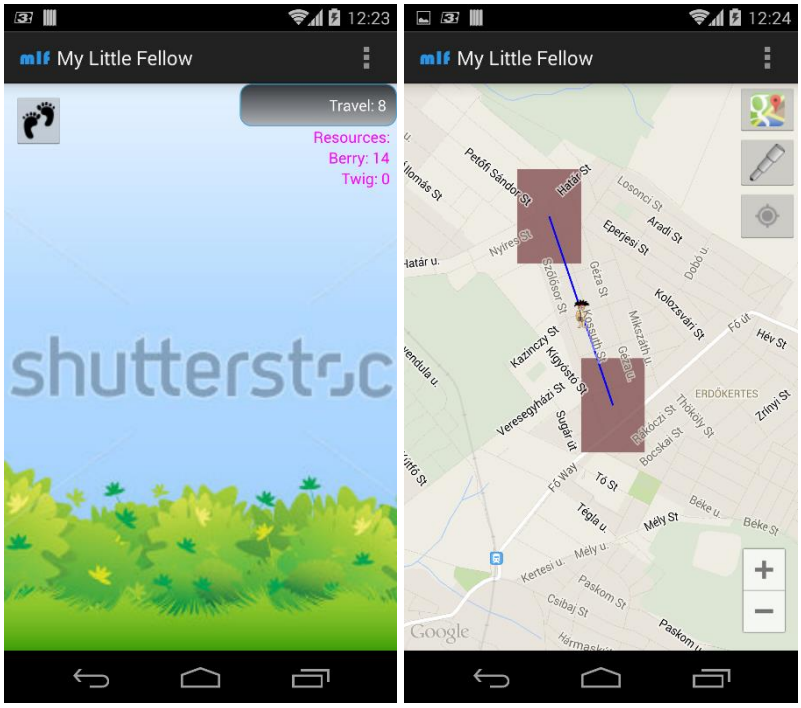


2.12. ábra – Raktár, 1 gallyat és 1 bogyót tartalmaz, a raktár mérete 60 és 1 látványosságot találtunk meg

Ha problémában lennénk a raktár méretével, akkor építsünk új raktárt, vagy a nyersanyagra kattintva eldobhatunk abból.

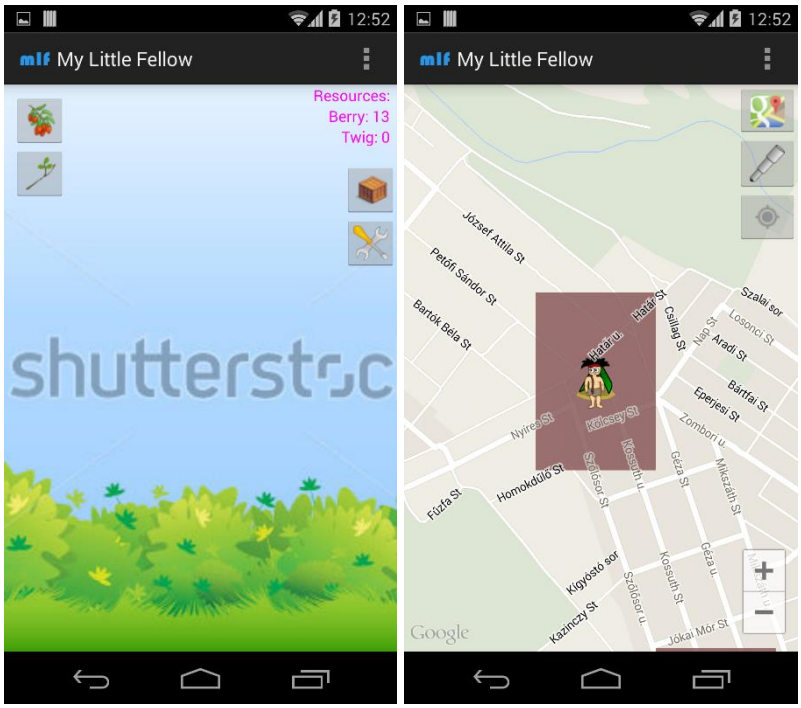
2.6.5. Utazás és letelepedés

A következő feladat a leendő lakhelyünk kiválasztása lesz. A területek között a karakter a felfedező mód kikapcsolt állapotában is mozoghat. Ehhez a térképre kell navigálnunk (ha egy területen vagyunk, a vissza gombbal tudunk a térképhez ugrani), majd ki kell választani egy olyan területet, ahol nem tartózkodik a karakterünk. Erre a területre lépjünk be, bal oldalon megjelenik egy lábnyom ikon, ez jelenti az utazást. Utazzunk számunkra szimpatikus helyre.



2.13. ábra – Az utazás megjelenése a terület és a térkép oldalán

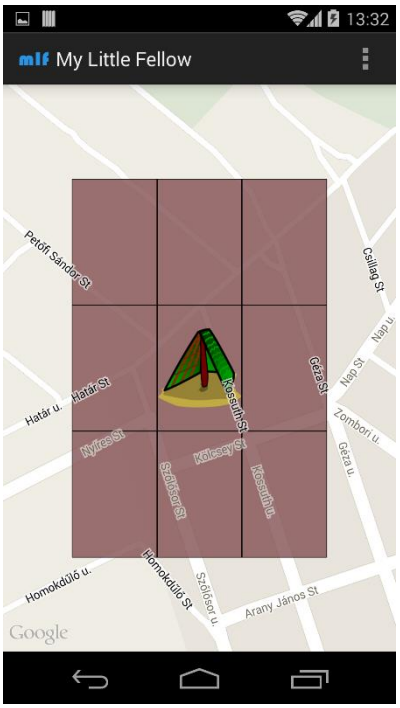
Ha megérkeztünk arra a területre, ahol le szeretnénk telepedni, lépjünk be a területre, majd válasszuk a sátor ikont. Idővel a letelepedés befejeződik, ekkor megjelenik a területen az építkezés lehetősége. A térképen a sátor jelzi, hogy melyik területen telepedtünk le.



2.14. ábra – Letelepedés után a területen megjelenik az építkezés lehetősége, a térképen a sátor jelenti a lakhelyünket

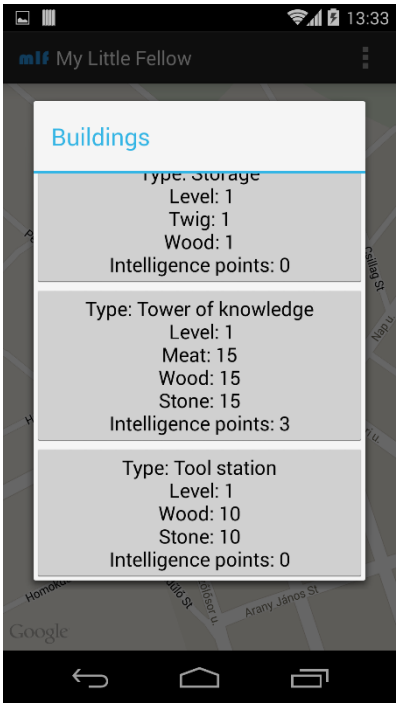
2.6.6. Építkezés

Az első cél az, hogy megépítsük a Tudás Tornyát. Lépjünk be a területünkre, ahol letelepedtünk, majd nyomjuk meg az építkezés gombját. Ekkor megjelenik újra a térkép az otthonunkkal középpontban, a területünk pedig 9 részre lett felosztva. Erre a 9 részre lehet épületeket építeni.



2.15. ábra – A terület 9 részre felosztása

Kattintsunk egy olyan helyre, ahol még nincs épület, majd keressük ki a Tudás Tornyát (Tower of knowledge). Láthatjuk, hogy milyen nyersanyagokra és hány intelligencia pontra van a megépítéshez szükség. Ezen az ablakon mindig csak azok az épületek jelennek meg, amik a kattintott területrészletre építhetőek. Menjünk és szerezzük be a szükséges dolgokat a Felfedező mód és a Terület alfejezetekben már megismert eszközökkel.



2.16. ábra – Az építhető épületek a szükséges nyersanyagokkal

Jelen esetben 15 hús, 15 fa és 15 kő szükséges a Tudás tornya megépítéséhez, ezeket a következő típusú területeken fogjuk megtalálni:

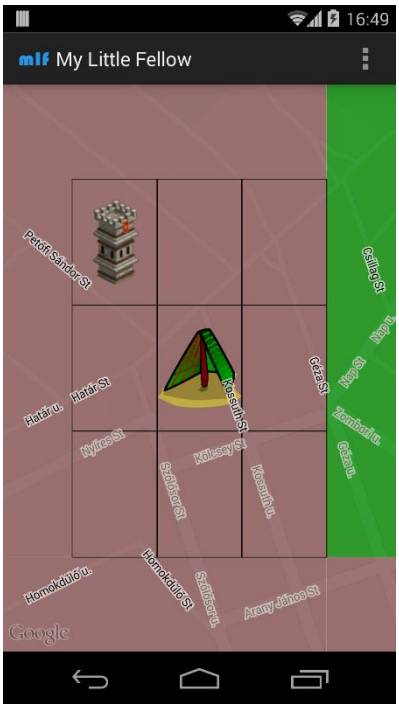
- Puszta: állatok vadászása
- Erdő: fák kivágása
- Kőbánya: kő bányászása

Ha megvan minden, akkor indítsuk el az építkezést és várjuk meg, amíg befejeződik.

A Tudás Tornyában lehet kifejleszteni az új épületeket, amik alapesetben nem elérhetőek. Ilyenek például az eszközkészítő vagy a Tudás Tornya magasabb szintjei.

Ha kész van egy épület, akkor azt lehet fejleszteni is, ami vagy növeli a kapacitását az épületnek (például raktár), vagy új dolgokat engedélyez (Tudás Tornya). Egy épület maximális szintje az 5-ös szint.

A következő cél legyen az Eszközkészítő állomás. Ezt ugyanúgy, mint az előbb, az építkezés pontban tehetjük meg.



2.17. ábra – Befejeződött a Tudás tornyának építése

2.6.7. Fejlesztés

Ha az eszközkészítő állomás is fel van építve, akkor menjünk a területre. Észrevehetjük, hogy jobb oldalon megjelent két új ikon, egy torony és egy fúró. Itt lehet kifejleszteni új épületeket és eszközöket. Példaképp lépjünk be a toronyba, és fejlesszük ki a hármas szintű raktárt (ha nincs elég nyersanyagunk, akkor gyűjtsünk).

2.18. ábra – Épületek fejlesztési lehetőségei

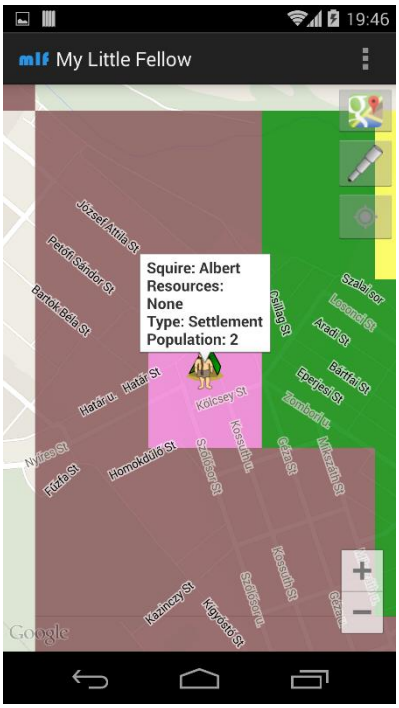
Ezekkel a fejlesztésekkel lehet elérni a magasabb szintű épületeket. Ezek elengedhetetlenek a végső célhoz, az 5-ös szintű városházához.
Az eszközök kifejlesztése nem előfeltétele a városháza felfejlesztésének, de felgyorsítja azt. Minden nyersanyag kitermeléséhez szükséges megadott számú idő. Ha kifejlesztünk egy eszközt, akkor ez az idő csökkenni fog.

2.19. ábra – Idővel eljutunk addig a pontig, hogy már fa fejszét tudunk kifejleszteni

Jelenleg 3 szintű fejszét és 3 szintű csákányt lehet kifejleszteni. Ha elfogytak a lehetséges fejlesztések, akkor magát az épületet is magasabb szintűre kell építeni. A 2.19. ábrán a 2. szintű fejszét lehet kifejleszteni.
A következő cél, hogy próbáljunk eljutni az 5-ös szintű városházáig.

2.6.8. Földesúrrá válás és az adó

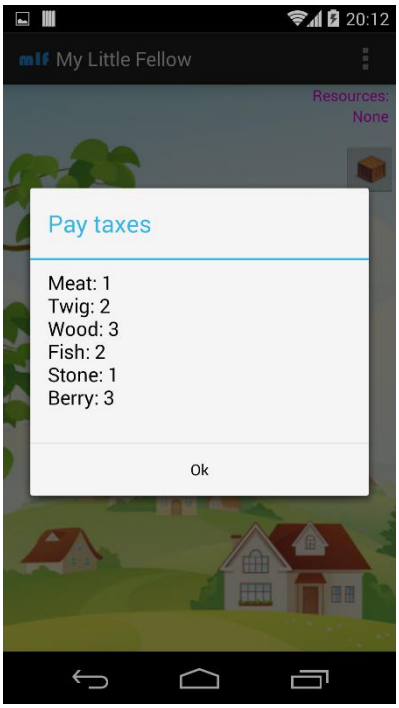
Amikor elérjük a városházával az 5-ös, vagyis a maximális szintet, a terület, ahol építettünk, várossá változik. Ez azt jelenti, hogy az összes eddigi nyersanyag eltűnik róla és átalakul a típusa. Ez az átalakulás csak akkor történik meg, ha még előtte nem volt város típusú. Ha a területen nekünk sikerült először elérni ezt a szintet, akkor mi válunk a terület földesurává. Ez azt jelenti, hogy a nevünk megjelenik, amikor valaki a területnek az információit tekinti meg, és adót állíthatunk be. A maximális beállítható adó nyersanyagonként: 5. Ezt az adót mindenkinek be kell fizetnie, aki a területre lép. Állítsunk be adót a 2.21.-es ábrához hasonlóan.



2.20. ábra – A terület földesurává nőttük ki magunkat, a nevünk megjelenik az információs ablakon

Ha olyan területre lépünk, ahol már van földesúr és van adó is beállítva, akkor egy üzenet után kifizettetik az adott összeget. Ha nincs nálunk elég nyersanyag, akkor természetesen annyit vonnak le, amennyit lehet. Egy levonás után egy hétig szabadon beléphetünk a területre.

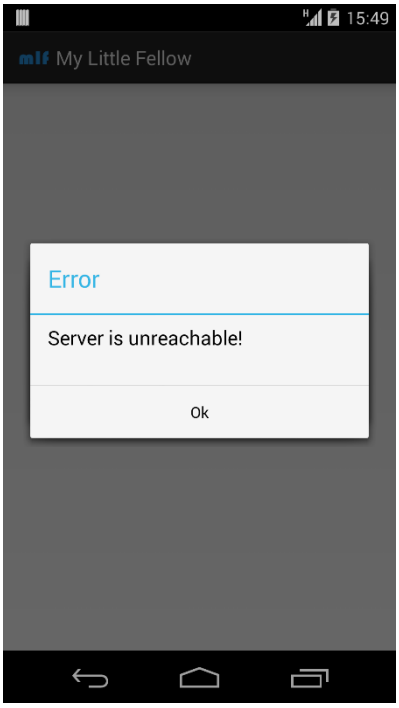
2.21. ábra – Beállított adó



2.22. ábra – Egy másik játékos szemszögéből a területen beállított adót kell befizetni

2.7. A játék hibakezelése

A játékban kétféle hiba létezik, a játék általi és a szerver általi. Ezt a kettőt futtatás során általában nem lehet megkülönböztetni. A leggyakoribb hiba az internetkapcsolat hiányából ered.



2.23. ábra – Az internetkapcsolat megszűnt

Ha az internet valamilyen oknál fogva megszakad, akkor a következő olyan interakciónál, ami a szervert is igénybe veszi, a „Server is unreachable!” hibát kapjuk.

Az időt igénybe vevő interakciók során előfordul, hogy az eszköz nem számol pontosan vissza és máskor telik le az idő helyben, mint a szerveren. Ha ilyen előfordul, azaz hamarabb telik le az idő, mint kéne, akkor a szerver „nincs változás” üzenettel válaszol. Ez után az eszköz még 5 másodpercenként ötször megpróbálja megismételni az előző ellenőrzést. Ha ezek után sem sikerül befejezni az interakciót, akkor a szerveren történt valamilyen hiba, és a játék visszavonja a változásokat. Ez a probléma a felhasználónak úgy jelenik meg, hogy a számláló még újra visszaszámol 5 másodperctől.

Ha egyéb, nem várt hiba történik a programban, akkor automatikusan elküldi a szervernek a problémát és újraindítja magát.

3. Fejlesztői dokumentáció

3.1. A rendszer felépítése

3.1.1. Ötletek

Az alapötlet az volt, hogy egy olyan játékot valósítok meg, ami egyesíti a gyűjtögetős játékokat a helymeghatározással. Mivel Androidos eszközzel rendelkezem és már használtam korábban is fejlesztői célokra, így adva volt, hogy milyen platformot fogok használni. A Google Maps beágyazott térkép nagyon egyszerűen és jól használható, az én igényeimnek megfelelően paraméterezhető. Az adott eszközökkel és az ötlettel együtt már majdnem minden megvolt. A térképet felosztottam négyzetekre, és minden négyzethez típust rendeltem. Két világot alkottam: a játékbelit és a valódit. Kellett valami, ami összeköti a kettőt, így jutott eszembe a látványosságok felvitele a játékba. A Google térkép nem ad túl sok lehetőséget lokalizációs információk lekérdezéséhez, így az OpenStreetMap-hez fordultam, hogy ezt a problémát megoldjam. Van limit ennek a szervernek a használatára, így arra a döntésre jutottam, hogy legyen egy új réteg, egy adatbázis, amiben tárolom az összes adatot.

A végső ötlet az lett, hogy az összes adat egy központi szerveren, adatbázisban legyen tárolva, ami kommunikál az OpenStreetMap-pel, az Androidos kliens pedig a központi szerverrel.

3.1.2. Megvalósítás

A My Little Fellow az Android operációs rendszert veszi alapul. A játék adatai egy központi MySQL adatbázisban kerülnek mentésre, ennek eléréséhez egy webszerver biztosít kapcsolatot. A webszerver PHP weboldalakból áll és megadott input esetén ad vissza outputot. Az adatok a szerver felé a HTTP POST metódusában vannak tárolva, a kliens felé pedig az output (ha van output) JSON szabvánnyal van kódolva. A kliens az Androidos eszközön egy programként fut és megadott interakciók esetén kommunikál a webszerverrel. A kliensen tárolva vannak a szervertől lekért adatok egy helyi adatbázisban, ezáltal mindig csak a szükséges információ kerül átvitelre. Mint már korábban említettem, a lokalizációs információkat az OpenStreetMap-ről kérem le a szerverrel. A nagyobb adathalmazokat a saját adatbázisban tárolom, mert ezeknek a lekérése általában lassú folyamat. Így a tavakat és a látványosságokat előre, egy megadott script segítségével le kell kérni és fel kell tölteni az adatbázisba.

3.2. A szerver

3.2.1. Az adatbázis felépítése

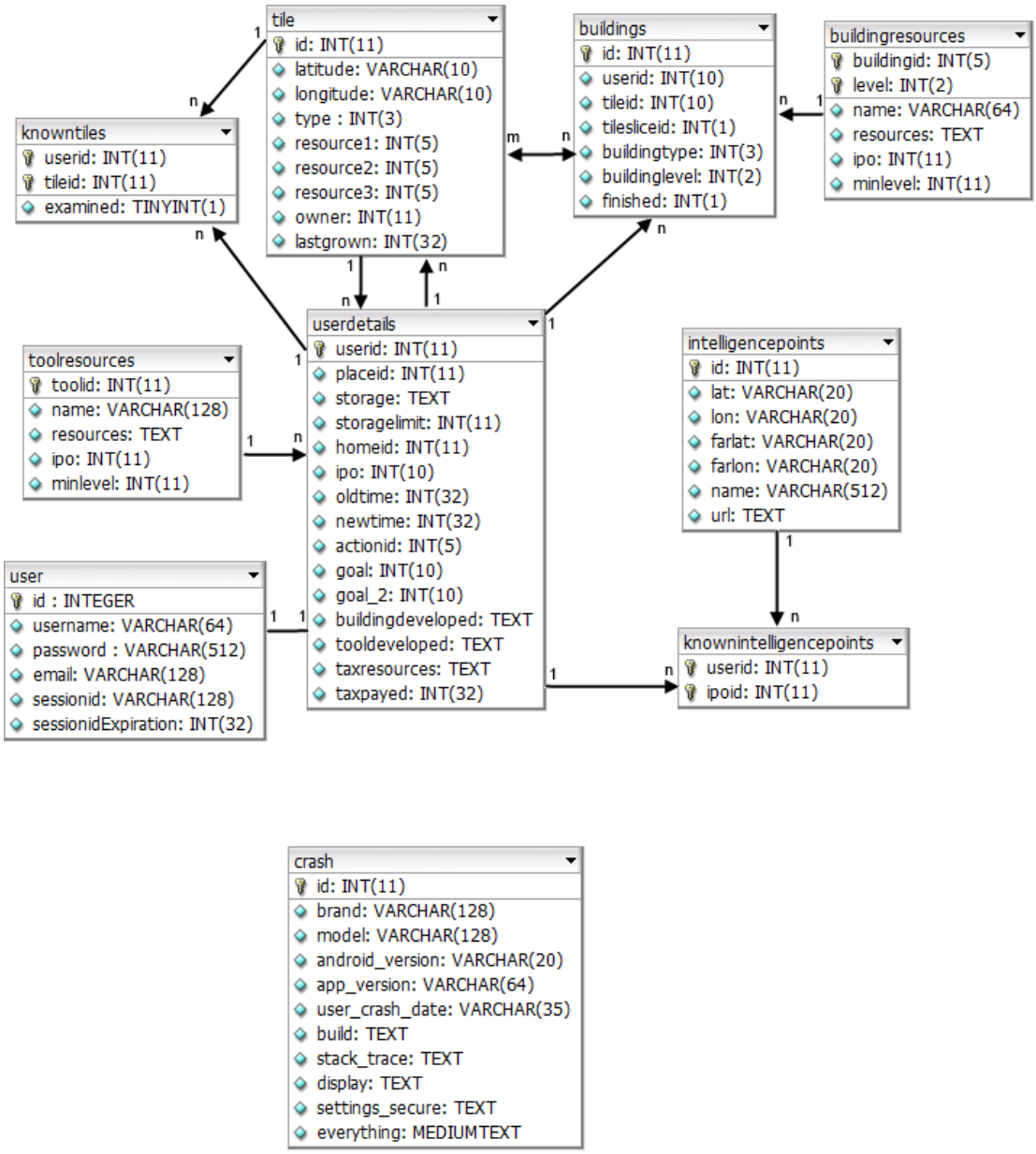
A szerveren MySQL adatbázist használok az adatok tárolására. Az adatbázis modellje a 3.1.-es ábrán látható.

A táblák leírása a következő:

buildingresources	Az épületekhez szükséges nyersanyagokat tárolja
buildingid	Az épület típusa
level	Az épület szintje
name	Az épület neve
resources	A szükséges nyersanyagok JSON-ben
ipo	A szükséges intelligenciapontok
minlevel	Minimum Tudás Tornya szint

Kapcsolatok:

- 1:n kapcsolat a building táblával (buildingid, level – buildingtype, buildinglevel)



3.1. ábra – Az adatbázis modellje

building	A karakterek által megépített épületeket tárolja
id	Az elsődleges kulcs
userid	Melyik karakteré a kulcs
tileid	Melyik területen található az épület
tilesliceid	A terület 9 részre felosztott részéből melyik területen található az épület
buildingtype	Milyen típusú az épület
buildinglevel	Hányas szintű az épület
finished	Elkészült-e már az épület

Kapcsolatok:

- m:n kapcsolat a tile táblával (tileid - id)

crash	A kliens, ha hibát talál, elküldi a szervernek. Ez a hiba itt tárolódik
id	Az elsődleges kulcs
brand	Az eszköz márkája
model	Az eszköz típusa
android_version	Az eszközön futó Android verziója
app_version	Az eszközön futó kliens verziója
user_crash_date	A hiba futásának dátuma
build	Az eszköz részletesebb statisztikái
stack_trace	A hiba leírása
display	Az eszköz képernyőjének adatai
settings_secure	Az eszköz beállításai
everything	A hibával kapcsolatos összes adat

intelligencepoints	Az OpenStreetMap-től lekért látványosságok adatai
id	Az elsődleges kulcs
lat	A szélességi fok
lon	A hosszúsági fok
farlat	Ha nagy kiterjedésű látványosságról van szó, annak a legtávolabbi szélességi foka
farlon	Az előző hosszúsági foka
name	A látványosság neve
url	A látványossághoz tartozó weboldal

Kapcsolatok:

- 1:n kapcsolat a knownintelligencepoints táblával (id – ipoid)

knownintelligencepoints	A karakter által felfedezett látványosságokat tartalmazza
userid	A karaktert leíró azonosító
ipoid	A látványosságot leíró azonosító

knowntiles	A karakter által felfedezett területeket tartalmazza
userid	A karaktert leíró azonosító
tileid	A területet leíró azonosító
examined	Megvizsgálta-e a területet

tile	A szerver által generált területeket tartalmazza
id	Az elsődleges kulcs
latitude	A terület középpontjának szélességi foka
longitude	A terület középpontjának hosszúsági foka
type	A terület típusa
resource1	A területen található egyik nyersanyag száma
resource2	A második nyersanyag száma
resource3	A harmadik nyersanyag száma
owner	A terület földesura
lastgrown	A terület nyersanyagai mikor nőttek utoljára

Kapcsolatok:

- 1:n kapcsolatban áll a knowntiles táblával (id – tileid)
- 1:n kapcsolatban áll a userdetails táblával (id – placeid)
- 1:n kapcsolatban áll a userdetails táblával (id – homeid)

toolresources	Az eszközök kifejlesztéséhez szükséges nyersanyagokat tartalmazza
toolid	Az elsődleges kulcs
name	Az eszköz neve
resources	A szükséges nyersanyagok
ipo	A szükséges intelligencia pont
minlevel	Az eszközkészítő épület szükséges minimum szintje

Kapcsolatok:

- 1:n kapcsolatban áll a userdetails táblával (toolid – tooldeveloped)

user	A felhasználó adatai
id	Az elsődleges kulcs
username	A felhasználó neve
password	A felhasználó jelszava hash-elve
email	A felhasználó e-mail címe
sessionid	A felhasználó játékmenetének az azonosítója
sessionidExpiration	A játékmenet lejárásának az ideje

Kapcsolatok:

- 1:1 kapcsolatban áll a userdetails táblával (id – userid)

userdetails	A karakter adatai
userid	A felhasználót leíró azonosító
placeid	A karakter tartózkodási területének azonosítója
storage	A karakter raktára
storagelimit	A karakter raktárának mérete
homeid	A karakter lakhelyének azonosítója
ipo	A karakter intelligenciapontja
oldtime	A karakter időigényes interakciójának a kezdeti ideje
newtime	A karakter időigényes interakciójának a befejezési ideje
actionid	A karakter időigényes interakciójának a leírója
goal	Az időigényes interakció egyik segédváltozója
goal_2	Az időigényes interakció másik segédváltozója
buildingdeveloped	A karakter által kifejlesztett épületek listája
tooldeveloped	A karakter által kifejlesztett eszközök listája
taxresources	Ha a karakter földesúr, akkor az általa beállított adó
taxpayed	A karakter által utoljára befizetett adó ideje

Kapcsolatok:

- 1:n kapcsolatban áll a building táblával (userid – userid)
- 1:n kapcsolatban áll a knownintelligencepoints táblával (userid – userid)
- 1:n kapcsolatban áll a knowntiles táblával (userid – userid)
- 1:n kapcsolatban áll a tile táblával (userid – owner)

3.2.2. Adattárolási módszerek

A következő típusokat használtam a tárolások során:

- INT
- TINYINT
- VARCHAR
- TEXT
- MEDIUMTEXT

A koordináták tárolásához VARCHAR típust használtam, mert a tapasztalat azt mutatta, hogy a DOUBLE típus néha pontatlanul tárolta az értékeket. Ezek a pontatlan értékek pedig a térképen akár méteres elcsúszásokat is okoztak.

Több helyen kulcs-érték párokat JSON-ben tároltam az új tábla létrehozása helyett. A több táblánál azt tapasztaltam, hogy feleslegesen bonyolulttá vált az adatlekérés a szerver részéről. Ezzel a módszerrel a szervernek csak le kell kérnie az adatbázistól az adatot és adhatja is tovább a kliensnek, nem szükséges külön legenerálnia az adatszerkezet JSON leíróját.

3.2.3. Megoldási terv

Az alapkoncepció az volt, hogy olyan konstrukciót alakítsak ki, amit könnyű átlátni és könnyű kiegészíteni. A terv az volt, hogy a szervertől való lekérdezéseket csoportosítom aszerint, hogy milyen adatbázis táblát használnak, de ez nem volt egy szigorú feltétel. A lekérdezéseket ugyanakkor szét kell osztani két részletre, két külön fájlra, ahol az egyik fájl fogadja az adatokat és a paramétereik szerint továbbküldi a másik fájlnak, ami pedig kezeli a tárolást, adatkezelést, majd a végén visszaadja a kliensnek küldendő adatot.

A következő feladat az autentikáció megoldása. Olyan megoldásra van szükség, ami minden egyes lekérdezésnél azonosítja is a felhasználót. Erre a SessionId-t fogom használni, ami pontosan definiál egy felhasználót, és az időt, amikor érvényes a lekérdezése.

3.2.4. A szerver szerkezete

A szervert PHP nyelven írtam meg. Maga a szerver nem más, mint több weboldal, amik elvégzik a megfelelő adatkezelést az adott hívás hatására. A hívásokat különszedtem az adatkezeléstől, ezt az alábbiakban részletezem.

A szervernek két nagyobb rétege van, a fogadó oldal és az adatfeldolgozó oldal. A fogadó oldal kapja meg a klienstől az adatokat, ami definiálja, hogy milyen hívást kell végrehajtani, majd ez továbbítja az adatfeldolgozónak a megfelelő információt. A fogadó végzi minden esetben az autentikációt.

Azt az elvet követtem, hogy a fogadó oldal minden esetben a nev.php nevet kapja, a hozzá tartozó adatfeldolgozó pedig a nevClass.php-t. Az adatfeldolgozóban mindig egy osztály szerepel, hogy ha később hivatkozom rá, akkor el legyenek különítve az adott függvények.

A fogadó oldalt hívja meg a kliens és a HTTP POST metódusban adja át a paramétereket. Kötelező adat mindig a Sessionid és az Operation. A Sessionid az autentikációhoz szükséges, az Operation pedig egy pozitív egész szám, ami meghatározza, hogy milyen hívást szeretne végrehajtani a kliens. Ezeken a paramétereken kívül még bármilyen más paramétert lehet használni, de ezek meglétének az ellenőrzése mindig szükséges. Ha

minden paraméter megfelelő, utána lehet meghívni az adatfeldolgozó megfelelő függvényeit.

A következő fogadó-adatfeldolgozó párosokat hoztam létre:

- Authenticate – regisztrálja, bejelentkezteti vagy ellenőrzi a felhasználót
- Building – az épületekkel kapcsolatos hívásokat tartalmazza
- Ipo – a látványosságokkal kapcsolatos hívásokat tartalmazza
- KnownTiles – a karakterrel és a területtel kapcsolatos hívásokat tartalmazza
- Log – a kliens által küldött hibákat menti el
- Tile – a csak a területtel kapcsolatos hívásokat tartalmazza, pl.: terület lekérdezése
- TimedAction – az időigényes interakciókkal kapcsolatos metódusok
- Tool – minden, ami az eszközökkel kapcsolatos
- UserDetails – a karakter adatainak beállításai

Ezen felül vannak még egyéb oldalak, amik nem sorolhatóak be a kétrétegű rendszerbe, ezek pedig a következők:

- db – Az adatbázis kapcsolathoz szükséges adatokat tartalmazza.
- exceptions – A szerver, amikor hibát dob, akkor küld egy üzenetet is. Ezek az üzenetek itt találhatóak
- globals – Az adatbázisban található összes konstans (táblák, oszlopok nevei) itt találhatóak, hogyha valami változás történik, akkor csak itt kelljen átírni. Minden ilyen változó globálisan van elmentve.
- openstreetmap – A Generátoroldalak alfejezetben részletesebben.
- sessionAuthentication – az interakcióknál végzi el az autentikációt

3.2.5. Szerverhívások

A kliens által hívott interakciókat a fogadó oldalnév és a művelet azonosító (operation) páros határozza meg. Minden hívásnál, ha valamilyen hiba történik, vissza lehet adni a hibaid, hibaszoveg párost, amit a kliensnek tudnia kell kezelni. Így a következőkben, ha ez a kimeneti paraméter, azt nem fogom külön feltüntetni. Ugyanígy a bemeneti paramétereknél nem fogom külön feltüntetni a sessionid, operation paramétereket, mert ugyanúgy használatosak az összes hívásnál.

A hívások a következők:

3.2.5.1. Authenticate

- 0-s művelet

Regisztrálja a felhasználót a megadott paraméterek alapján. Ha már szerepel a felhasználó e-mail címe az adatbázisban, akkor csak bejelentkezteti.

Bemeneti paraméterek:

`username`: a felhasználó neve

`password`: hashelt jelszó

`email`: a felhasználó e-mail címe

`version`: a kliens verziókódja

Kimeneti paraméter:

`sessionid`

3.2.5.2. Building

- 0-s művelet

A megadott paraméterekre elindít egy építkezést. Ez időigényes interakció.

Bemeneti paraméterek:

`tileid`: Melyik területre építsen

`tileslice`: A terület melyik szeletére a 9 részből

`buildingtype`: Milyen típusú épületet építsen

Kimeneti paraméterek:

`time`: Az építkezés befejezésének ideje

`oldtime`: Az építkezés kezdetének ideje

- 1-es művelet

Lekérdezi a megadott területen építhető épületek listáját.

Bemeneti paraméterek:

`tileid`: Melyik terület kérdezi le az épületeket

`tileslice`: Melyik területrészletre kérdezi le az épületeket

Kimeneti paraméterek:

Egy tömb az épületek listájáról a következő paraméterekkel:

`buildingtype`: Az épület típusa

`buildinglevel`: Az épület szintje

`resources`: Az építéshez szükséges nyersanyagok tömbje

`name`: Az épület neve

`ipo`: Az építéshez szükséges intelligenciapontok száma

- 2-es művelet

Lekérdezi a felhasználó által épített összes épületet.

Kimeneti paraméterek:

Az épületek egy tömbje a következő paraméterekkel:

`id`: Az épület azonosítója

`userid`: A felhasználó azonosítója

`tileid`: Az épület területének azonosítója

`buildingtype`: Az épület típusa

`buildinglevel`: Az épület szintje

`finished`: Be van-e fejezve az épület

- 3-as művelet

Lekérdezi az összes kifejleszthető épülethez szükséges nyersanyagokat.

Kimeneti paraméterek:

Az épületek egy tömbje a következő paraméterekkel:

`buildingid`: Az épület típusa

`level`: Az épület szintje

`name`: Az épület neve

`resources`: A nyersanyagok tömbje

`ipo`: A szükséges intelligenciapontok

`minlevel`: A minimum szükséges Tudás Tornya épület szint

- 4-es művelet

Elkezdi kifejleszteni a paraméterekben definiált épületet.

Bemeneti paraméterek:

`buildingtype`: Az épület típusa

`buildinglevel`: Az épület szintje

Kimeneti paraméterek

`time`: A fejlesztés idejének vége

`oldtime`: A fejlesztés idejének kezdete

3.2.5.3. Ipo

- 0-s művelet

Legenerálja a paraméterek által definiált látványosságokat az OpenStreetMap szerver segítségével. Részletek a Generátoroldalak alfejezetben.

Bemeneti paraméterek:

`latfrom`: A legdélnyugatibb szélességi fok

`lonfrom`: A legdélnyugatibb hosszúsági fok

`latto`: A legészakkeletibb szélességi fok

`lonto`: A legészakkeletibb hosszúsági fok

`radius`: A lépték mértéke fokokban

- 1-es művelet

Lekérdezi a felhasználó által látott látványosságokat a paraméterben megadott keretben.

Bemeneti paraméterek:

`latfrom`: A legdélnyugatibb szélességi fok

`lonfrom`: A legdélnyugatibb hosszúsági fok

`latto`: A legészakkeletibb szélességi fok

`lonto`: A legészakkeletibb hosszúsági fok

Kimeneti paraméterek:

Az látványosságok tömbje a következő paraméterekkel:

`id`: A látványosság azonosítója

`lat`: A látványosság szélességi foka

`lon`: A látványosság hosszúsági foka

`farlat`: A legtávolabbi pont szélességi foka (ha van, egyébként NULL)

`farlon`: Az előző hosszúsági foka

`name`: A látványosság neve

`url`: A látványosság weboldala

`known`: A felhasználó felfedezte-e már a látványosságot

- 2-es művelet

A karakter felfedezi a látványosságot.

Bemeneti paraméter:

`ipoid`: A látványosság azonosítója

3.2.5.4. KnownTiles

- 0-s művelet

Megvizsgál egy területet a karakter.

Bemeneti paraméterek:

`tileid`: A terület azonosítója

Kimeneti paraméterek:

`time`: A vizsgálás végének időpontja

`oldtime`: A vizsgálás kezdetének időpontja

- 1-es művelet

Lekérdezi egy területről, hogy a karakter megvizsgálta-e már

Bemeneti paraméter:

`tileid`: A terület azonosítója

Kimeneti paraméter:

`examined`: Megvizsgálta-e már a karakter

3.2.5.5. Log

Ez az oldal kivételt képez az előzőekhez képest. Az Androidos kliens az Acra nevű könyvtárat használja a hibák szerverre küldéséhez. Amit a fogadó oldal kap POST metódusban, azt továbbítja az adatfeldolgozó oldalnak, ami rögzíti az adatbázisban.

3.2.5.6. Tile

- 0-s művelet

Frissíti a megadott területet a paraméterekben megadott nyersanyagokkal.

Bemeneti paraméterek:

`id`: A terület azonosítója

`resource1`: Az első helyen található nyersanyag

`resource2`: A második helyen található nyersanyag

`resource3`: A harmadik helyen található nyersanyag

- 1-es művelet

Lekérdezi a paraméterekben definiált terület információit.

Bemeneti paraméter:

`id`: A terület azonosítója

Kimeneti paraméterek:

`id`: A terület azonosítója

`latitude`: A terület középpontjának a szélességi foka

`longitude`: A terület középpontjának hosszúsági foka

`type`: A terület típusa

`resource1`: Az első helyen található nyersanyag

`resource2`: A második helyen található nyersanyag

`resource3`: A harmadik helyen található nyersanyag

`population`: A területen élő karakterek száma

`tax`: A fizetendő adó tömbje

`owner`: A terület földesurának azonosítója

`examined`: A területet megvizsgálta-e már a karakter

- 2-es művelet

A megadott paraméterek által definiált területen lekérdezi az összes tavat az OpenStreetMap szerverről. Részletek a Generátoroldalak alfejezetben.

Bemeneti paraméterek:

`latitude`: A legdélnyugatibb szélességi fok

`longitude`: A legdélnyugatibb hosszúsági fok

`latitudeTo`: A legészakkeletibb szélességi fok

`longitudeTo`: A legészakkeletibb hosszúsági

- 3-as művelet

Lekérdezi egy területről, hogy a felhasználó-e a földesúr

Bemeneti paraméter:

`id`: A terület azonosítója

Kimeneti paraméter:

`owner`: 1, ha a felhasználó a földesúr a területen

3.2.5.7. TimedAction

Ennek az oldalnak a működéséről részletesebben az Időigényes interakciók alfejezetben lesz szó.

- 0-s művelet

Lekérdezi egy időigényes interakció adatait

Kimeneti paraméterek:

`actionid`: Az interakció típusának azonosítója

Építkezés esetén:

`id`: Az épület azonosítója

`userid`: A karakter azonosítója

`tileid`: A terület azonosítója

`tilesliceid`: A területrész azonosítója

`buildingtype`: Az épület típusa

`buildinglevel`: Az épület szintje

`finished`: Be van-e fejezve

Egyébként:

`goal`: Az első segédváltozó – típustól függően változó

`goal_2`: A második segédváltozó – típustól függően változó

- 1-es művelet

Az interakció végén kell meghívni, befejezi az interakciót, ha ténylegesen vége van a hívásnak.

Kimeneti paraméter:

`result`: 1, ha sikeresen befejeződött az interakció, 0 egyébként

- 2-es művelet

Leállítja a futó interakciót, utazás esetén használt egyedül.

3.2.5.8. Tool

- 0-s művelet

Lekérdezi az összes karakter által kifejleszthető eszközt

Kimeneti paraméter:

Az eszközöket leíró tömb a következő paraméterekkel:

`toolid`: Az eszköz azonosítója

`name`: Az eszköz neve

`resources`: Az eszközhöz szükséges nyersanyagok

`ipo`: Az eszközhöz szükséges intelligenciapontok

`minlevel`: A minimum szükséges Eszközkészítő épület szintje

- 1-es művelet

Elkezdi kifejleszteni a megadott eszközt.

Bemeneti paraméter:

`tooltype`: Az eszköz típusa

Kimeneti paraméter:

`time`: A fejlesztés végének időpontja

`oldtime`: A fejlesztés kezdetének időpontja

3.2.5.9. Userdetails

- 0-s művelet

Felfedez egy új területet.

Bemeneti paraméterek:

`latitude`: A terület középpontjának szélességi foka

`longitude`: A terület középpontjának hosszúsági foka

Kimeneti paraméterek:

A terület azonosítói:

`id`: A terület azonosítója

`latitude`: A terület középpontjának szélességi foka

`longitude`: A terület középpontjának hosszúsági foka

`type`: A terület típusa

`resource1`: A területen az első nyersanyagok száma

`resource2`: A területen a második nyersanyagok száma

`resource3`: A területen a harmadik nyersanyagok száma

`owner`: A terület földesurának azonosítója

- 1-es művelet

Lekérdezi a felhasználó által felfedezett összes területet.

Kimeneti paraméterek:

Az összes terület tömbben, az összes paraméterrel együtt (az előző művelet kimeneti paramétere)

- 2-es művelet

Lekérdezi, hogy melyik területen áll a karakter.

Kimeneti paraméter:

`placeid`: A terület azonosítója

- 3-as művelet

Lekérdezi a karakter lakhelyét.

Kimeneti paraméter:

`homeid`: A terület azonosítója

- 4-es művelet

Lekérdezi a karakter raktárának méretét.

Kimeneti paraméter:

`storagelimit`: A raktárnak a mérete

- 5-ös művelet

Lekérdezi a karakter raktárának tartalmát.

Kimeneti paraméter:

`storage`: A raktár tartalma (nyersanyagokat tartalmazó tömb)

- 6-os művelet

Beállítja az első helyen lévő nyersanyagot

Bemeneti paraméterek:

`type`: A nyersanyag típusa

`amount`: A nyersanyag darabszáma

- 7-es művelet

Beállítja, hogy melyik területen legyen a karakter (elutazás).

Bemeneti paraméter:

`placeid`: A terület azonosítója

Kimeneti paraméterek:

`time`: Az utazás végének időpontja

`oldtime`: Az utazás kezdetének időpontja

- 8-as művelet

Elindítja a területre letelepedést.

Bemeneti paraméter:

`homeid`: A terület azonosítója

Kimeneti paraméterek:

`time`: A letelepedés végének időpontja

`oldtime`: A letelepedés kezdetének időpontja

- 9-es művelet

Lekérdezi, hogy tele van-e a karakter raktára.

Kimeneti paraméter:

`storagelimit`: 1, ha tele van, egyébként 0

- 10-es művelet

Az első helyen lévő nyersanyag kitermelésének az elindítása.

Bemeneti paraméterek:

`type`: A terület típusa

`placeid`: A terület azonosítója

Kimeneti paraméterek:

`time`: A termelés végének időpontja

`oldtime`: A termelés kezdetének időpontja

- 11-es művelet

A második helyen lévő nyersanyag kitermelésének az elindítása.

Bemeneti paraméterek:

`type`: A terület típusa

`placeid`: A terület azonosítója

Kimeneti paraméterek:

`time`: A termelés végének időpontja

`oldtime`: A termelés kezdetének időpontja

- 12-es művelet

A harmadik helyen lévő nyersanyag kitermelésének az elindítása.

Bemeneti paraméterek:

`type`: A terület típusa

`placeid`: A terület azonosítója

Kimeneti paraméterek:

`time`: A termelés végének időpontja

`oldtime`: A termelés kezdetének időpontja

- 13-as művelet

Lekérdezi a karakter által beállított adókat.

Kimeneti paraméter:

`taxresources`: Az adónak beállított nyersanyagok tömbje

- 14-es művelet

Beállítja az adóban a paraméterben megadott nyersanyagot.

Bemeneti paraméterek:

`type`: A nyersanyag típusa

`amount`: A nyersanyag darabszáma

- 15-ös művelet

Kifizeti a karakter az adót, ha az adott területre lépett.

Bemeneti paraméter:

`tileid`: A terület azonosítója

- 16-os művelet

Lekéri a karakter intelligenciapontját

Kimeneti paraméter:

`ipo`: Az intelligenciapont

3.2.6. A szerver és az adatbázis kapcsolata

Az adatbázishoz való csatlakozáshoz a PHP Data Object osztályát használtam fel. Ez az osztály biztosít könnyen kezelhető és biztonságos (az SQL Injection problémát oldja meg) adatbázishívásokat. Az adatbázist a kliens általi szerverhívások közül az elsőnél nyitom meg, majd utána a cache-ben lementem, hogy később ne kelljen erőforrást pazarolni a megnyitáshoz. Ehhez a PDO beépített „persistent connection" módszerét használom.

Az adatbázis tábláknak a neveit és az oszlopok neveit globális változókban tárolom, és ezeket a kliensnek visszaküldendő adatok azonosítására is felhasználom.

Például:

A Userdetails 0-s művelete a Tile tábla oszlopainak a nevével azonosítja az adatokat, a felesleges átnevezések elkerülése érdekében.

Egy példa lekérdezés az adatbázis hívások szemléltetéséhez:

```
$sql = "SELECT ".$GLOBALS["table_userdetails_placeid"]." FROM
".$GLOBALS["table_userdetails_title"]."
WHERE ".$GLOBALS["table_userdetails_userid"]." = ?";
$stmt = $this->db->prepare($sql);
$stmt->execute(array($this->userid));
$resultArray = $stmt->fetch(PDO::FETCH_ASSOC);
return $resultArray;
```

A fenti lekérdezés a karakter jelenlegi tartózkodási helyét adja vissza a $resultArray tömbben, ami a ''placeid'' => n kulcs-értéket tartalmazza (n a terület azonosítója).

Az adatbázis többi hívását (beszúrás, módosítás, törlés) hasonlóképpen lehet elvégezni.

3.2.7. Generátoroldalak

3.2.7.1. Az OpenStreetMap-ről

A játékban a területek között van tó és folyó típusú, az érdekesség az, hogy ezek a területek a helyszín tényleges földrajzi adatai szerint kerülnek az adott típusba. Ugyanígy a látványosságok is valódi látványosságok, nem véletlenszerűen generált adatok. Ezeknek az információknak a lekérésében segít az OpenStreetMap szerver. Ez egy ingyenesen használható eszköz, ami földrajzi információkat szolgáltat. Az ingyenessége miatt nincsenek olyan erős szervereik, mint például a Google-nek, ezért a túl sok lekérés akár

letiltáshoz is vezethet. Ezért döntöttem úgy, hogy a dinamikus lekérdezések helyett inkább generátoroldalakat hozok létre, amik csak lefutáskor kérdeznek le adatot.

A lekérdezésekhez az Overpass-API híváskészletet használtam, ami egy XML lekérdezés, úgynevezett OSM-script alapján visszaadja a számomra fontos információkat, ugyancsak XML-ben. Ezt az XML-t főként 3 adat alkotja

- node
- way
- relation

A node a térképen egy pontot jelöl, a hozzá tartozó információk: id (azonosító), latitude (szélességi fok), longitude (hosszúsági fok).

A way egy node-okból álló sorozat, amik egy sokszöget alkotnak. A hozzá tartozó információk: id (azonosító), node-ok id-vel ellátva

A relation egy way-ekből álló sorozat, amik több sokszögből alkotnak egy alakzatot. Általában nagy kiterjedésű objektumoknál használják, vagy akkor, ha a sokszögből ki szeretnének venni egy darabot. (Például egy olyan tó, aminek a közepén van egy sziget)

A hozzá tartozó információk: id (azonosító), way-ek id-vel ellátva, plusz a típusa: outer (külső részhez tartozik) vagy inner (belső részhez tartozik).

3.2.7.2. Adatok generálása

Generátoroldalakat a tavaknál és a látványosságoknál használok. A folyóknál egy köztes megoldást választottam: nem kell generálni őket, dinamikusan jönnek létre, de csak egyszer. Egy generáláshoz két koordinátapontra és egy távolságra van szükség. A két koordinátapont egy téglalapot zár be (az első pont a téglalap bal alsó pontja, azaz a délnyugati sarka, a másik a jobb felső pontja, azaz az északkeleti sarka). A generálás menete az, hogy ebben a megadott téglalapban sorra veszünk kisebb négyzeteket a távolságban megadott átmérővel, és erre kérdezzük le az OpenStreetMap-től az információkat. Az ebben talált tavakat vagy látványosságokat pedig elmentjük a saját adatbázisunkban. Ez a generálás a mérettől függően sok időt is igénybe vehet.

3.2.7.3. Tavak

A tavakat a legbonyolultabb generálni, ugyanis csak a két bonyolultabb típus (way, relation) előfordul. Miután a tavakat sikerült lekérnünk, még fel kell dolgozni a területekhez. Ezt a következőképpen végzem el:

1. Egyesítem a way-eket és a relation-öket úgy, hogy a relation-ökből is way-eket csinálok (kihagyom a szigeteket).
2. Végigmegyek egyesével az összes tavon
3. Behatárolok egy tavat egy nagy téglalapba a legtávolibb koordinátái alapján
4. Végigmegyek sorban a nagy téglalapon kis téglalapokat vizsgálva (ez a kis téglalap megegyezik egy területtel)
5. Megvizsgálom, hogy a tó benne van-e a kis téglalapnak a középpontjától számított 0.004 fok (a terület 0.005 fok széles) átlóhosszúságú rombuszban. Ha benne van, akkor tóként elmentem az adatbázisban a területet.

Ennek a vizsgálatnak az eredménye, hogy a nagy tavak biztosan megjelennek a területeken, a kis tavak pedig csak akkor, ha belelógnak a terület közepébe.

3.2.7.4. Folyók

A folyóknál kicsit egyszerűbb a helyzet, itt csak way típusú alakzatokat kapok vissza, nem sokszögeket, csak vonalakat. Itt nem alkalmazok generátoroldalt, így ez kivételt képez a többihez képest.

A folyó területre illesztése a következő módszerrel történik:

1. Ha valaki felfedez egy területet, amit még senki sem fedezett fel, akkor a szerver lekérdezi az OpenStreetMap-től, hogy van-e ezen a területen folyó.
2. Ha van folyó, akár a sarokban is, a terület folyó típust kap.

Az OpenStreetMap sajnos nem tartalmaz információt a folyók szélességéről, így előfordul, hogy hibásan jelenik meg a folyó a játékban.

3.2.7.5. Látványosságok

A látványosságok a térképen két féle módon jelennek meg: node-ként, vagy way-ként.

A kérdés először az volt, hogy milyen típusú látványosságokat is szeretnék látni a játékban. A következőket választottam ki:

- Épület típusú látványosságok:
    - templom
    - katedrális
- Történelmi típusú látványosságok:
    - régészeti lelőhely
    - légi jármű
    - kastély
    - ágyú

    - városkapu
    - emlékmű
    - rom
    - rúna kő
    - síremlék
    - útszéli kereszt
    - útszéli emlék
- Turizmus típusú látványosságok:
    - műalkotás
    - múzeum
    - szép kilátás

A node és a way típusú látványosságokat a megjelenítésben különszedtem. A node pont típusú, így itt nem történt semmi változás. A way viszont nagyobb kiterjedésű, így itt a következő ötletet alkalmaztam:

1. Veszem az összes, a way-ben előforduló koordinátát
2. Kiszámolom a számtani közepét, így megkapom közelítve a látványosság közepét
3. Végül megkeresem a középponttól a legtávolabbi pontot.

A középpontból és a legtávolabbi pontból pedig a kliensen egy kört állítok fel, és ha a játékos ebben a körben van, akkor megtalálta a látványosságot.

3.2.8. Időigényes interakciók

Bonyolult kérdést vetett fel az időigényes interakció. A szerver, mivel weboldalakból áll, nem képes visszahívni a klienst. A kliensnek ugyanakkor tudnia kell, hogy mikor indult a számolás (a szerver számolása) és időben vissza kell tudnia hívnia a szervert. Így azt találtam ki, hogy az időigényes hívásnál a szerver visszaad két számot: a számlálás kezdeti időpontját és a végpontját. Ezeket az időpontokat a kliens majd a sajátjához igazítja (ha más időzónában van, mint a szerver), és visszahívja a szervert, ha letelt a visszaszámlálás. Ezt a visszahívást csak akkor fogadja el a szerver, ha tényleg lejárt a visszaszámlálás. Ameddig ez a hívás nem történt meg, addig le lehet kérdezni az ezzel kapcsolatos információkat, ha a kliens valamilyen oknál fogva (program újrarakása) elfelejtené azt.

Az interakciókat egy actionid szerint csoportosítottam. A következő interakciók lehetségesek:

1 - Utazás
2 - Terület megvizsgálása
3 - Letelepedés
4 - Első nyersanyag kitermelése
5 - Második nyersanyag kitermelése
6 - Harmadik nyersanyag kitermelése
7 - Építkezés
8 - Épület kifejlesztése
9 - Eszköz kifejlesztése

Az interakciókhoz általában tartozik egyéb információ, ami minden egyes hívásnál különbözik. Pl.: utazásnál az új terület azonosítója, épület kifejlesztésénél az épület típusa és szintje. Ezeket az egyéb információkat a goal és a goal_2 segédváltozókban tárolom. Ha több információ szükséges, azt más módszerrel kell megoldani. Ehhez példa az épület építés, ott egy új oszlopot vettem fel az adatbázisban finished névvel.

Az időigényes interakciót a timedActionClass.php függvényei végzik. A függvények hívásának menete a következő:

Első hívás: set... (pl.: setBuild), elindítja az interakciót

Adatok lekérése: getData()

Interakció vége: refreshData()

Opcionálisan interakció leállítása: stopAction()

3.3. A kliens

A kliens Android operációs rendszerre íródott és a következő Google szolgáltatásokat alkalmazza:

- Google Maps API v2
- Google Play Services

Az Androidos programok nagyrészt Java nyelven íródnak, így én is ezt a nyelvet választottam.

3.3.1. A kliens négy rétege

A klienst négy nagyobb csoportra bontottam. Minden csoportot egy külön csomagban (package-ben) helyeztem el. Ez a négy csoport a következő:

- Communicator
- Storage
- GUI
- Model

A négy rész egymástól teljesen különböző funkciókat lát el. Az egymásra hivatkozás megengedett, kivéve a Model és a GUI részek között. A modell nem tartalmaz semmilyen grafikus osztályra való hivatkozást, csak eseményt küldhet neki.

3.3.2. Kommunikáció a szerverrel

Minden kommunikáció a Communicator csomagban található. A szerverhívásokat az általuk hívott szerveroldal szerint csoportosítottam külön osztályokba. Minden ilyen osztálynak az őse a CommunicatorBase absztrakt osztály. Ennek az osztálynak a fontosabb függvényei a következők:

- `getResponse`

Privát láthatóságú függvény, ami a kapott paraméterekkel meghívja a megfelelő műveletet a szerveren, majd visszatér annak az eredményével.

Bemenő paraméter:

`nameValueList` – egy kulcs-érték párokat tartalmazó tömblista, ami a szerverhívásban a paramétereket jeleni

Kimenő paraméter:

`result` – egy karakterlánc, amit a szerver ad vissza

- `addPair`

Védett láthatóságú függvény, ami hozzáad egy paramétert az éppen beállított híváshoz.
Bemenő paraméterek:

`name` – a paraméter neve

`value` – a paraméter értéke

- `getSingleResponse`

Védett láthatóságú függvény, meghívja a szervert a paraméterekkel, majd visszatér a hívás JSONObject típusú eredményével. A JSONObject kulcs-érték párokat tartalmaz.
Kimenő paraméter:

`result` – JSONObject típusú válasz

- getMultiResponse

Védett láthatóságú függvény, meghívja a szervert a paraméterekkel, majd visszatér a hívás JSONArray típusú eredményével. A JSONArray JSONObject-eket tartalmaz.
Kimenő paraméter:

`result` – JSONArray típusú válasz

Mint már korábban említettem, minden kommunikációhoz használt osztály ebből az osztályból származik. Saját magam által használt konvenció, hogy minden kommunikáló osztály a CommunicatorNev alakú nevet használja.
A következő osztályokat használom:

- CommunicatorBuilding
- CommunicatorIpo
- CommunicatorKnownTiles
- CommunicatorTimedAction
- CommunicatorTile
- CommunicatorTool
- CommunicatorUser
- CommunicatorUserDetails

3.3.2.1. Hibakezelés

Kívülről a getSingleResponse és a getMultiResponse függvények használatosak, ahol előfordulhat kommunikációs hiba. Három féle kivétel fordul elő általában a hívásoknál: NetworkErrorException, JSONException, CommunicatorException.

A NetworkErrorException hálózati probléma esetén szokott előfordulni.

A JSONException parse-olási probléma.

A CommunicatorException a szerver által dobott általam definiált hibákat kezeli, például „No more resources” azaz nincs több nyersanyag.

Ismeretlen hiba esetén Exception váltódik ki.

A négy esetre lehet használni az ősosztály handleNevException függvényt.

Egy példa egy egyszerű szerverhívásra:

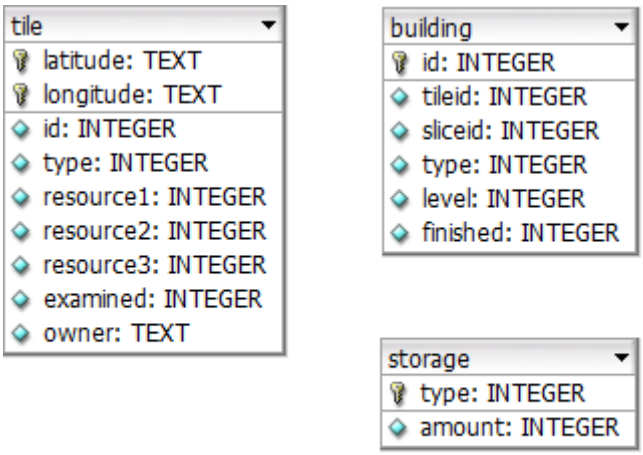
```
public JSONObject demoFunction() {
//A PARAM_OPERATION egy konstans String
addPair(PARAM_OPERATION, ”0”);
//Egyéb paraméter hozzáadása
addPair(”parameter2”, ”test”);
try {
//maga a szerverhívás, eredménnyel
JSONObject jObject = getSingleResponse();
return jObject;
}
//hibakezelés, ősosztályba beépített függvénnyel
catch(NetworkErrorException e)
{ handleNetworkErrorException ( e); }
catch(JSONException e) { handleJSONException ( e); }
catch(CommunicatorException e)
{ handleCommunicatorException ( e); }
catch(Exception e) { handleException( e); }
return null;
}
```

3.3.3. A kliens adatbázisa és egyéb adattárolása

Ahhoz, hogy a kliens megfelelő gyorsasággal működjön, szükség van egy helyi adatbázisra, ahol olyan adatokat tárolok, amik csak az adott kliensre vonatkoznak, és mindig kiszámítható események bekövetkeztekor változnak. Ezzel redukálható a kliens által felhasznált sávszélesség adatmennyisége is. Ezt az adatbázist csak a kliens éri el, más program elől rejtve van. Az Androidon elérhető adatbázis az SQLite, így én is ezt használtam.

3.3.3.1. Szerkezet

Az adatbázis mindig regisztráláskor/bejelentkezéskor jön létre. 3 táblát definiáltam, de később mindenképp szükség lenne többre. A következő ábra szemlélteti a táblák szerkezetét.



3.2. ábra – A kliens adatbázisa

A táblák szerkezetének leírása a következő:

tile	A felhasználó által felfedezett területeket tartalmazza
latitude	A terület közepének szélességi foka
longitude	A terület közepének hosszúsági foka
id	A terület azonosítója
type	A terület típusa
resource1	A terület első nyersanyaga
resource2	A terület második nyersanyaga
resource3	A terület harmadik nyersanyaga
examined	Meg lett-e vizsgálva a terület
owner	A terület földesura

Az egyedi kulcsokat a koordináta két értékére raktam, mert ezt a kliens generálja, így biztosítva van, hogy nem szerepel kétszer ugyanaz a terület. A terület azonosítót a szerver generálja, így attól az elvárt, hogy nem ütközik másik azonosítóval. Ütközés esetén az új beszúrandó sort figyelmen kívül hagyja.

building	A felhasználó által épített épületeket tartalmazza
id	Az épület azonosítója
tileid	A terület azonosítója
sliceid	A terület részletének azonosítója
type	Az épület típusa
level	Az épület szintje
finished	Be van-e fejezve az épület

Az egyedi kulcs az épület azonosítóján van, új sor ütközése esetén figyelmen kívül hagy.

storage	A felhasználó raktárát tartalmazza
type	A nyersanyag típusa
amount	A nyersanyag darabszáma

Az egyedi kulcs a nyersanyag típusa, új sor ütközése esetén figyelmen kívül hagy.

3.3.3.2. Használat

Minden táblához külön osztály tartozik, ami a DatabaseAdapter absztrakt osztály leszármazottja. Ezek az osztályok a NevDatabaseAdapter nevet viselik. A származtatás során szükség van egy SaveableObject típusú osztályra is, ezeket a modellben definiálom. A SaveableObject annyit vár el a típustól, hogy legyen neki egy egyedi kulcsa, id-je. A következő metódusokat szükséges megvalósítani a származtatás után:

- `String[] getAllColumns()`

Feladata: Visszaadja az adatbázisbeli táblának az oszlopainak a nevét egy tömbben.

- `String getTableName()`

Feladata: Visszaadja az adatbázis táblának a nevét.

- `String getIdColumnName()`

Feladata: Visszaadja az egyedi kulccsal rendelkező táblának a nevét.

- `ContentValues getObjectContentValues(T object)`

ContentValues típusú objektumba rakja bele oszlopnevek szerint a T típusú objektum adatait, majd visszatér vele.

- `T cursorToObject(Cursor cursor)`

Egy Cursor típusú objektumból csinál T típusú objektumot, majd visszatér a T-vel.

Ezek megvalósítása után egy teljes értékű osztályt kapunk, ami minden alapvető műveletet képes végrehajtani az adatbázison. Ezek a műveletek a következők:

- `boolean addRow(T row)`

Hozzáad egy T típusú objektumot a táblához.

- `ArrayList<T> getAll()`

Lekérdezi az összes objektumot egy listába.

- `T getById(int id)`

Lekérdezi az objektumot a megadott kulccsal.

- `ArrayList<T> getByWhereClause(String whereClause)`

Lekérdezi az objektumot a megadott where feltétellel.

- `update(T object)`

Felülírja a táblában a megadott objektumot, és ha eddig nem létezett, akkor hozzáadja a táblához.

- `update(T object, boolean addRowIfNotExist)`

Felülírja a táblában a megadott objektumot. Beállítható, hogy ha nem találja meg a sort, szúrjon-e be új sort.

- `boolean isExists(T object)`

Igazzal tér vissza, ha az objektum már szerepel a táblában, egyébként hamissal.

- `delete(T object)`

Kitörli a megadott objektumot a táblából.

- `deleteAll()`

Kitörli az összes objektumot a táblából.

- `boolean isEmpty()`

Igazzal tér vissza, ha üres a tábla.

- `open()`

Megnyitja a táblát.

- `close()`

Bezárja a táblát.

3.3.3.3. Egyéb adattárolás

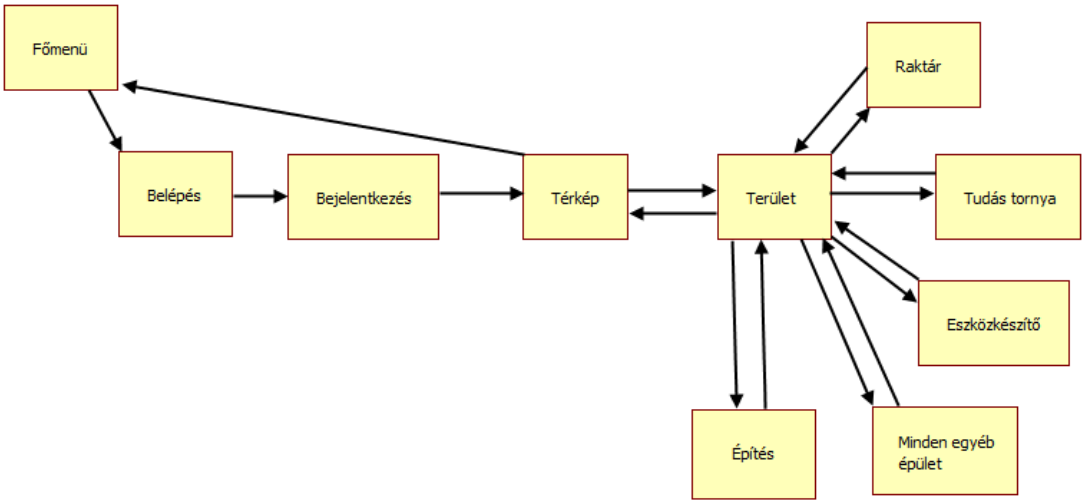
Szükség volt egyedi adatok tárolására is, amik nem igényelték, hogy egy adatbázisban legyenek tárolva. Erre az Android SharedPreferences osztályát használtam, ami egy elkülönített helyen kulcs-érték párokban tárol adatokat.

A Storage nevű osztályban tudok lementeni és betölteni ilyen adatokat. Jelenleg négy féle típusú adatot lehet lementeni: String Integer, Double és Long. A következő adatokat tárolom itt:

- email
- username: felhasználónév
- password: hashelt jelszó
- sessionid
- userTileId: A karakter tartózkodási helyének azonosítója
- homeId: A karakter lakhelyének azonosítója
- timedActionStartTime: Időigényes interakció kezdeti időpontja
- timedActionEndTime: Időigényes interakció végének időpontja
- timedActionType: Időigényes interakció típusa
- timedActionGoal: Időigényes interakció egyik segédváltozója
- timedActionGoal2: Időigényes interakció másik segédváltozója
- storageLimit: Raktár mérete
- intelligencePoints: Intelligenciapontok száma
- lastLatitude: karakter ismert utolsó koordinátája
- lastLongitude: karakter ismert utolsó koordinátája

3.3.4. Az oldalak szerkezete

A tervezés során felrajzoltam egy szerkezetet, hogy hogyan képzelem el az oldalak közötti navigálást. Ez a rajz kicsit szebben a 3.3. ábrán látható. A nyilak azt jelentik, hogy bizonyos oldalról melyik oldalra tudunk átjutni, a visszafelé mutató nyíl pedig a Vissza gombot szimbolizálja.



3.3. ábra – Oldalak közötti navigálás

Ezt az ötletet sikerült tartani a megvalósítás során is, bár a visszajelzések azt sugallják, hogy ezen később változtatni kell.

Az Android az oldalakat Activity-knek nevezi és ezekből mindig pontosan egy lehet előtérben. A 3.3. ábrán látható elemek is szintén egy Activity-t jelölnek, kivéve a főmenüt, mert az adva volt, a Belépés/Bejelentkezés össze lett vonva és a Minden egyéb épület nem lett megvalósítva. Minden Activity-hez tartozik egy úgynevezett layout is, ami az oldal kinézetét írja le egy XML fájlban. Összesen egy oldal a kliensen tehát 3 rétegből tevődik össze: a layout XML-ből, az Activity-ből és az Activity-hez tartozó modellből.

3.3.4.1. Ősosztály

Egy kivételével minden általam használt oldal egy absztrakt osztályból származik. Ez az ősosztály a BaseActivity. Ebben olyan dolgokat valósítottam meg, ami szükséges, hogy minden egyes oldalon jelen legyen. Ezek a következők:

- Portré tájolás beállítása
- Időigényes interakció kezelése
- Internetkapcsolat ellenőrzése
- Helyzet-érzékeny súgó megjelenítése
- Hibák kezelése

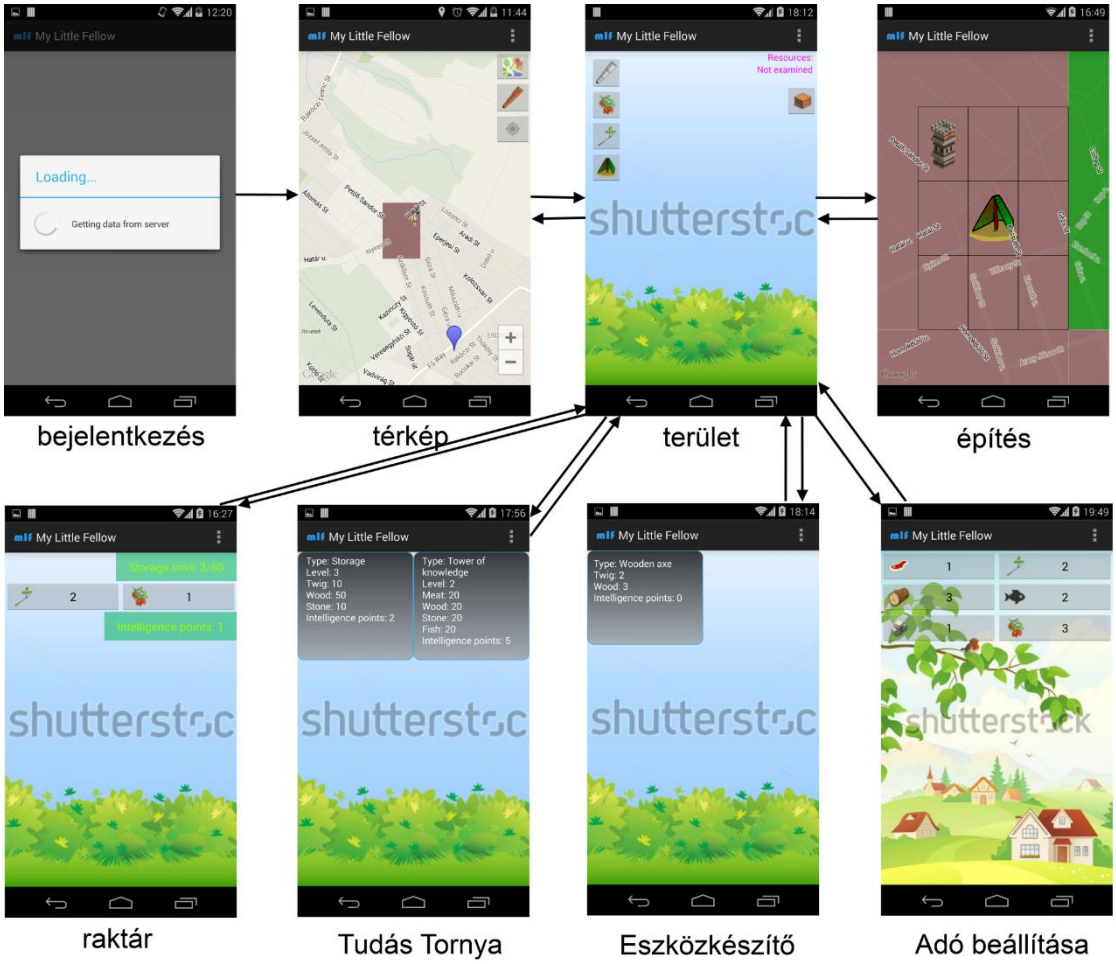
A helyzet-érzékeny súgó minden oldalon a Menüből elérhető és röviden leírja, hogy mire való az az oldal, amin éppen vagyunk.

3.3.4.2. Egy oldal feladata

Egy Activity feladata a felület megjelenítése és a felülettel való interakció továbbítása a modellnek. A felület megjelenítéséhez jórészt a layout XML-t használom, ami a statikus elemek beállítására szolgál. Előfordul, hogy dinamikusan megjelenő elemeket is ki kell rajzolni, ezeket az Activity-kben állítom be.

Minden oldalhoz tartozik egy modellosztály, ami a Model csomag része. Ez végzi el a megjelenítéshez nem köthető, vagy bonyolultabb metódusokat, és ez intézi a szerverhívásokat is. Ha kell, a modell tud eseményt küldeni az Activity-nek.

Az oldalak közti navigációt a 3.4-es ábra szemlélteti.



3.4. ábra – Oldalak közötti navigáció

A következő oldalakat definiáltam:

- MapActivity: A térkép megjelenítésére szolgál, ez a program központi oldala
- PlaceActivity: Egy területet jelenít meg
- StorageActivity: A raktárról szolgáltat információt
- TaxActivity: Itt lehet beállítani az adót

- TileSliceChooseActivity: Az építkezés zajlik ezen az oldalon
- ToolstationActivity: Az eszközfejlesztések oldala
- TowerActivity: Az épületeket lehet itt kifejleszteni
- LoginActivity: Az egyetlen, nem az ősosztályból származó osztály, a bejelentkezés alatt mutatja ezt az oldalt, automatikusan továbblép róla.

3.3.4.3. Eseménykezelés

Az oldalak közötti váltást a legtöbb esetben a felhasználó váltja ki. Egy kivétel van, a bejelentkezés. Itt a felhasználónak csak egy betöltő ablak jelenik meg, a sikeres bejelentkezés után automatikusan a következő oldalra, vagyis a Térképre navigálja a felhasználót.

A többi esetben a legtermészetesebb felhasználói eseményt használom ki, a gombnyomást. Így próbálom elérni, hogy egy olyan felhasználó is könnyen kezelje a programot, aki még nem ismeri.

Egyéb eset még a térképre való kattintás után megjelent információs ablakra való kattintás. Ez felhasználói szokásokhoz képest ismeretlen módszer, de ha egyszer megismerjük, észrevehetjük hasznosságát. Egyelőre nem sikerült okosabb módszert találnom erre.

3.3.5. Modell csomag

A modell csomagban található minden logikai résszel kapcsolatos kód. Ezek alkotják a program szerver részét, a következőkben leírom sorban, hogy melyik osztály milyen feladatot végez el.

3.3.5.1. Building

Egy épületet reprezentál. Tulajdonságok:

- `int id` – azonosító
- `int tileId` – terület azonosítója
- `int sliceId` – terület részletének azonosítója
- `int type` – az épület típusa
- `int level` – az épület szintje
- `boolean finished` – be van-e fejezve

3.3.5.2. BuildingReceipt

Az épület megépítéséhez szükséges receptet tárolja. Tulajdonságok:

- `int type` – az épület típusa
- `int level` – az épület szintje

- `String name` – az épület neve
- `int ipo` – a szükséges intelligenciapontok száma
- `ResourceStorage resources` – a szükséges nyersanyagok

3.3.5.3. Hash

Hasheléssel kapcsolatos függvényt tartalmaz.

Függvény:

```
String sha256(String base)
```

Bemeneti paraméter:

`base` – hashelendő karakterlánc

Kimeneti paraméter:

Hashelt karakterlánc

3.3.5.4. Ipo

Egy látványosságot reprezentál. Tulajdonságok:

- `double latitude`
- `double longitude`
- `String name`
- `String url`
- `double radius`
- `int id`
- `boolean known`

3.3.5.5. IpoGetter

A látványosságokat kéri le a szerverről, majd eseményekkel jelzi annak befejeztét.

Két részletben kéri le a látványosságokat: amit a karakter lát, és amit a felhasználó lát.

Fontosabb függvények:

```
setCharacterPlace(LatLng coordinate)
```

A karakter pozícióját állítja be, hogy majd erre a koordinátára kérjen le adatokat és el is indítja a letöltést.

Bemeneti paraméter:

`coordinate` – A karakter koordinátája

`setCameraPosition(LatLng nearLeft, LatLng nearRight, LatLng farLeft, LatLng farRight)`

Beállítja a felhasználó által látott térkép koordinátáit és el is indítja a letöltést

Bemeneti paraméter:

A kamera négy sarka

`downloadIposForCharacter()`

Letölti a karakter által látott látványosságokat.

`downloadIposForCamera()`

Letölti a felhasználó által látott látványosságokat.

3.3.5.6. Logger

Az általam használt osztály hibakereséshez.

Fontosabb függvények:

`writeToLog(String data)`

Kiírja a LogCat-be és egy fájlba a megadott karakterláncot.

Bemeneti paraméter:

`data` – Kiírandó szöveg

`writeException(Exception e)`

Kiírja a LogCat-be és egy fájlba a megadott kivételt.

Bemeneti paraméter:

`e` – Kivétel

3.3.5.7. Login

A LoginActivity modell osztálya.

Fontosabb függvények:

`startLogin()`

Elindítja a bejelentkezést.

`register(String username, String password, String rePassword)`

Elindítja a regisztrációt a megadott adatokkal.

Bemeneti paraméterek:

`String username` – felhasználónév

`String password` – Jelszó

`String rePassword` – Jelszó még egyszer

`login_(int status)` – Végigvezet a bejelentkezés lépésein.

Bemeneti paraméter:

`int status` – Egy állapotot jelent, 0,1,2,3 és 4 lehet.

0 – Elmentett adatok ellenőrzése

1 – Regisztráló ablak mutatása

2 – E-mail választása

3 – Sessionid lekérése

4 – Beléptetés és sessionid elmentése

3.3.5.8. Map

A MapActivity-hez tartozó modell. Ez kezeli az összes térképpel kapcsolatos logikai részeket.

A fontosabb függvények:

`drawIpos()`

Elindítja a látványosságok kirajzolását.

`startDiscoverMode()`

Bekapcsolja a GPS-szel együtt a felfedező módot.

`stopDiscoverMode()`

Kikapcsolja a GPS-szel együtt a felfedező módot.

`drawCharacter(int shownMarker, Marker clickedMarker)`

Megállapítja, hogy hova kell kirajzolni a karaktert, majd küld egy eseményt az Activity-nek.

Bemeneti paraméterek:

`shownMarker` – Hova kattintott utoljára a felhasználó – az információs ablak miatt fontos

`clickedMarker` – A jelölőpont, amire kattintott

`setClickedTile(LatLng centeredClickedCoordinate)`

Beállítja, hogy melyik területre kattintott, majd küld egy eseményt az Activity-nek, hogy nyissa meg a hozzá tartozó információs ablakot.

`centeredClickedCoordinate` – A kattintott koordináta területének a középpontja

`infoWindowClicked(Marker marker)`

Megállapítja, hogy melyik jelölőpont információs ablakára kattintott, majd küld egy eseményt az Activity-nek, hogy milyen oldalra kell továbbítani a felhasználót.

Bemeneti paraméter:

`marker` – A megnyitott jelölőpont

`loadAllVisibleTiles()`

Betölti az összes látható területet.

`setIpoGetter()`

Elindítja a látványosságok lekérését.

`LocationChanged(Location location)`

Akkor hívódik meg, amikor a GPS sikeresen lekérte a pozíciót.

`location` – A GPS által lekért helyszín adatai

`drawBuildings()`

Kirajzolja az összes épületet.

`downloadTile(Tile tile)`

Letölt újra egy területet a szerverről.

`tile` – Ezt a területet frissíti.

3.3.5.9. Place

Ez az osztály végzi el a PlaceActivity logikai részét, ami jelen esetben szerverhívásokból áll. Ha megvan a válasz, az Activity-t hívja meg a megfelelő üzenettel.

Fontosabb függvények:

`getTax()`

Lekérdezi a beállított adót.

`amIOwner()`

Lekérdezi, hogy földesúr-e az adott karakter.

`payTax()`

Befizeti az adót.

3.3.5.10. PlaceAction

A terület dinamikusan létrejött épületekkel kapcsolatos gombjainak az eseményeit kezeli ez az osztály. Pl.: építés, raktár

Fontosabb függvényei:

`int getPlaceActionCount(Context context, Tile tile, int usertileid)`

Lekérdezi, hogy hány gomb szerepel a megadott területen.

Bemeneti paraméterek:

`context` – Az Activity kontextusa

`tile` – A megnyitott terület

`usertileid` – A karakter tartózkodási helye

Kimeneti paraméter:

A gombok száma

`PlaceActionType getPlaceActionButtonType(Context context, ArrayList<Building> buildingList, int position, Tile tile, int usertileid)`

Lekérdezi az adott gomb típusát.

Bemeneti paraméterek:

`context` – Az Activity kontextusa

`buildingList` – Az épületek listája

`position` – A gomb sorszáma

`tile` – A terület

`usertileid` – A karakter melyik területen tartózkodik

Kimeneti paraméter:

`PlaceActionType`

Ez egy felsorolás típus a következő értékekkel:

`BUILD, STORAGE, TOWEROFKNOWLEDGE, TOOLSTATION, TAX`

3.3.5.11. Point

Két double értékű koordinátát tárol. Ezek a következők:

- `double x`
- `double y`

3.3.5.12. Resource

Egy nyersanyagot reprezentál.

Tulajdonságok:

- `int type`
- `int amount`

3.3.5.13. ResourceAction

A területhez kapcsolódó dinamikusan létrejövő interakciókat kezeli le. Pl.: utazás, favágás

Fontosabb függvények:

`int getActionButtonCount(int homeid, int usertileid, Tile tile)`

Megadja, hogy hány gombot kell megjeleníteni.

Bemeneti paraméterek:

`homeid` – A karakter otthonának azonosítója

`usertileid` – A karakter tartózkodási helyének azonosítója

`tile` – A terület, amin éppen van

Kimeneti paraméterek:

A gombok száma

`ActionButtonType getActionButtonType(int position, Tile tile, int homeid, int usertileid)`

Visszaadja a megadott sorszám alapján a gomb típusát.

Bemeneti paraméterek:

`position` – A gomb sorszáma

`tile` – A terület

`homeid` – A felhasználó otthonának azonosítója

`usertileid` – A felhasználó tartózkodási helyének azonosítója

Kimeneti paraméter:

`ActionButtonType`

Ez egy felsorolás típus a következő értékekkel:

EXAMINE, WOODCUT, COLLECTTWIG, HUNT, COLLECTBERRIES, MINE, FISH, SETTLEDOWN, TRAVEL

3.3.5.14. ResourceStorage

Több nyersanyagot darabszámmal reprezentál. Pl.: raktár, építéshez szükséglet

Tulajdonságok:

`ArrayList<Resource> storage` – a nyersanyagok tömbje

3.3.5.15. StorageModel

A StorageActivity-hez kapcsolódó logikát valósítja meg.

Fontosabb függvények:

`storagePreview()`

Amíg nem sikerült letölteni a raktárt, addig egy régebbi állapotot mutat.

`downloadStorage()`

Letölti a raktárt, majd eseményt küld az Activity-nek, ha végez.

`removeFromStorage(int resourceId, int amount)`

Eldob megadott számú nyersanyagot.

Bemeneti paraméter:

`resourceId` – A nyersanyag típusa

`amount` – A nyersanyag darabszáma

3.3.5.16. Tax

A TaxActivity-hez tartozó logikát valósítja meg.

Fontosabb függvények:

```
setTax(int id, int amount)
```

Beállítja a megadott paraméterek alapján az adót.

Bemeneti paraméterek:

`id` – A nyersanyag típusa

`amount` – A nyersanyag mennyisége

```
loadTax()
```

Betölti az adónak beállított nyersanyagokat.

3.3.5.17. Tile

Egy területet reprezentál az adott tulajdonságokkal:

`double tileCenterLatitude` – a terület középpontjának szélességi foka

`double tileCenterLongitude` – hosszúsági foka

`int type` – a terület típusa

`int resource1` – a terület első nyersanyaga

`int resource2` – a terület második nyersanyaga

`int resource3` – a terület harmadik nyersanyaga

`int id` – a terület azonosítója

`boolean examined` – meg van-e a terület vizsgálva

`String owner` – a földesúr neve

`ResourceStorage tax` – a beállított adó

`int population` – a lakosok száma

Fontosabb függvények:

```
Tile setTileCenterFromLocation(LatLng location)
```

Egy koordinátából megállapítja a terület központját, és be is állítja azt.

Bemeneti paraméter:

`location` – a koordinátát tartalmazó objektum

Kimeneti paraméter:

A beállított terület

`LatLng getCenteredCoordinate(LatLng coordinate)`

Egy koordináta alapján megmondja a területének a középpontját.

Bemeneti paraméter:

`coordinate` – a megadott koordináta

Kimeneti paraméter:

Központosított koordináta

`refreshTileWithDownload(Activity context, Tile tile)`

A szerverről frissíti az adott területet. Egy eseményt küld vissza a hívónak.

Bemeneti paraméterek:

`context` – a hívó Activity vászna

`tile` – a frissítendő terület

3.3.5.18. TileSliceChoose

A TileSliceChooseActivity-hez tartozó modell.

Fontosabb függvények:

`getAllBuildingsForDraw()`

Lekéri az összes épületet az adatbázisból és egyesével visszaküldi esemény formájában az Activity-nek, hogy rajzolja ki.

`downloadAllBuildables(LatLng coordinate)`

A térkép megadott koordinátája alapján megállapítja, hogy a terület mely részére kattintottunk, majd letölti az itt építhető épületek listáját. A listát egy eseménnyel küldi vissza.

Bemeneti paraméter:

`coordinate` – A kérdéses koordináta

`startBuilding(int buildingReceiptType)`

Elindítja a megadott épület építését.

Bemeneti paraméter:

`buildingReceiptType` – Az építhető épületek listájának az azonosítója

3.3.5.19. TimedAction

Az időigényes interakciót kezelő osztály.

A következő tulajdonságok reprezentálnak egy ilyen interakciót:

`int errorCount` – ha hiba történik, itt számolja a sikertelen küldések számát

`long firstStartTime` – a tényleges visszaszámlálás kezdete (az utazásnál van jelentősége)

`long startTime` – a visszaszámolás kezdete

`long endTime` – a visszaszámolás vége

`int type` – az interakció típusa

`int goal` – első segédváltozó

`int goal_2` – második segédváltozó

`int helperint` – harmadik, nem menthető segédváltozó

`int helperint2` – negyedik, nem menthető segédváltozó

`boolean RUNNING` – igaz, ha fut egy visszaszámlálás

`ActionCountDown actionCountDown` – a visszaszámláló osztály

Fontosabb függvények:

`long startTimedAction()`

Elindítja a már beállított típusú interakciót. Először a szervert hívja meg a típus szerint, majd ha minden megfelelő, akkor visszatér az interakcióhoz szükséges idővel.

Kimenő paraméterek:

Az interakcióhoz szükséges idő milliszekundumban.

`endTimedAction()`

Akkor hívódik meg, amikor letelt a számláló, ilyenkor elintéz minden típus szerinti beállítást.

`start()`

Elindítja az interakciót, ő hívja meg a startTimedAction()-t. Elkezdi a visszaszámolást is.

`resume()`

Ha még nincs befejezve egy visszaszámlálás, akkor azt folytatja onnan, ahol előzőleg abbahagyta.

`stop()`

Kényszeríti a visszaszámlálás megállítását és hibaüzenetet küld a megjelenítő felé.

`startWhenError()`

Hibára lépés esetén indítja újra magát ezzel a függvénnyel.

3.3.5.20. ToolReceipt

Egy eszköz kifejlesztéséhez szükséges dolgokat tartalmaz.

A következő tulajdonságok reprezentálják:

`int id` – a recept azonosítója

`String name` – az eszköz neve

`int ipo` – a szükséges intelligenciapontok száma

`ResourceStorage resources` – a szükséges nyersanyagokat tartalmazza

3.3.5.21. ToolStationModel

A ToolstationActivity-hez tartozó modell.

Fontosabb függvényei:

`loadToolStation()`

Letölti az összes kifejleszthető eszközt, majd eseményt küld róla az Activity-nek.

`startDeveloping(int id)`

A megadott azonosítóval ellátott eszközt elkezdi kifejleszteni.

Bemeneti paraméter:

`id` – Az eszköz azonosítója

3.3.5.22. TowerModel

A TowerActivity-hez tartozó modell.

Fontosabb függvényei:

`loadTowerDeveloping()`

Letölti az összes kifejleszthető épületet, majd eseményt küld róla az Activity-nek.

`startBuildingDevelop(int type, int level)`

Elkezdi kifejleszteni a megadott épületet.

Bemeneti paraméterek:

`type` – Az épület típusa

`level` – az épület szintje

3.3.5.23. VisibleTileAdapter

Azért felel, hogy mindig csak az az objektum rajzolódjon ki a térképre, ami ténylegesen látszódik.

A tulajdonságai a következők:

`ArrayList<Tile> allTiles` – tömb, ami tartalmazza az összes területet

`Map<Integer,GroundOverlay> visiblePolygon` – területazonosítóval definiált grafikai elem. Egy téglalapot jelent, ami a területet reprezentálja

`ArrayList<Point> regionCoordinates` – Azon koordináták tömbje, amik behatárolják a kamera által látott pontokat.

`ArrayList<Ipo> ipoSeenByCamera` – Azon látványosságok, amiket a kamera láthat.

`ArrayList<Ipo> ipoSeenByCharacter` – A karakter által látott látványosságok

`Map<Integer,Marker> visibleIpoMarker` – Az összes kirajzolt látványosság.

`Map<Integer,Circle> visibleIpoCircle` – Az összes kirajzol látványosság hatótávolságát jelző kör.

Fontosabb függvények:

`setVisibleRegion(LatLng nearLeft, LatLng nearRight, LatLng farLeft, LatLng farRight)`

Beállítja a kamera által látott pontokat.

Bemeneti paraméterek:

`nearLeft` – A bal alsó koordináta

`nearRight` – A jobb alsó koordináta

`farLeft` – A bal felső koordináta

`farRight` – A jobb felső koordináta

`getAllPolygons()`

A beállított kameranézet alapján megvizsgálja az összes kirajzolható alakzatot (azaz a látványosságot és területet), majd ami korábban látszott, azt úgy hagyja, ami újonnan bekerült a látótérbe, azt kirajzolja, a maradékot pedig törli.

`Map<Integer,Tile> getAllVisibleTiles()`

Visszaadja az összes olyan területet, ami látszódik a kamera által. Ezt használja fel a getAllPolygons() függvény is.

Kimeneti paraméter:

Egy olyan Map, amiben a kulcs a terület azonosítója, az érték pedig a terület.

`boolean isPointInPolygon(Point coordinate)`

Visszaadja egy koordinátáról, hogy a kamera látóterében van-e.

Bemeneti paraméter:

`coordinate` – a kérdéses koordináta

Kimeneti paraméter:

Igaz, ha a pont a látótérben van.

`drawAllVisibleMarkers()`

Kirajzolja a szükséges látványosságokat, azaz azokat, amiket a karakter és a kamera lát, ugyanolyan elven, mint a területek kirajzolása történik.

`tryToDiscover(LatLng characterCoordinate)`

Megkeresi melyik a legközelebbi látványosság a megadott koordinátához, és felfedezi azt.

Bemeneti paraméter:

`characterCoordinate` – A kérdéses koordináta, amit vizsgálunk.

3.3.6. Naplózás és hibakezelés

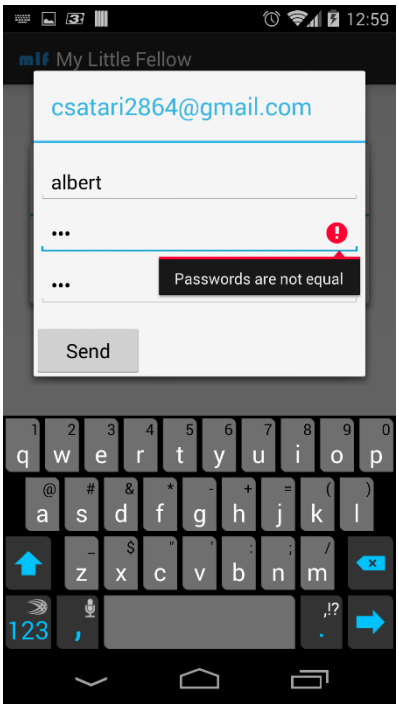
Az Android egyik szolgáltatása az úgynevezett LogCat, ami az alkalmazással kapcsolatos, fejlesztő által beállított szövegeket képes kiírni, hibakeresési célból. Ezt a szolgáltatást egészítettem ki a saját osztályommal (Logger osztály), ami a következő okokból könnyíti meg a hibakeresést:

- statikusan, bárhonnan meghívható, csak a kiírandó szöveget kéri paraméterként
- egyszerűen ki tud írni szöveget, vagy kivételt
- a szöveghez mindig csatolja a kiírt szöveg kódbeli helyét (Fájlnév, sorszám)
- a szöveghez csatolja a kiírás pontos dátumát
- minden kiírt szöveget egy szöveges fájlba írja, hogy később is visszakereshető legyen

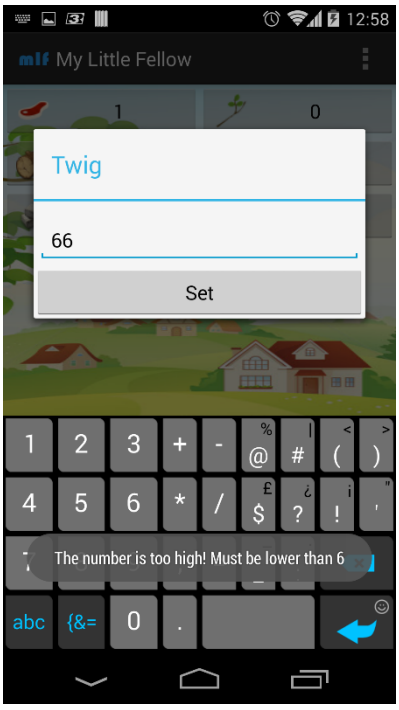
Ezen kívül nagyobb problémát okozott az alkalmazás hirtelen leállása. Ezen problémák kiderítésére beépítettem egy könyvtárat, ami képes automatikusan a szerveremnek elküldeni minden egyes ilyen hibaüzenetet a lehető legtöbb paraméterrel. Ennek a könyvtárnak a neve: Acra.

A sikeresen megtalált hibáknak mindig a kijavítására törekedtem, mint az elfedésére. Minden kritikus műveletet try-catch közé írtam, a catch-be pedig általában hibaüzenetet, a művelet kihagyását vagy megismétlését helyeztem. A súlyosabb hibák (pl.: szükséges, de az Android által felszabadított változó) esetén az alkalmazást újraindítom, de ezek legtöbbször akkor szoktak fellépni, amikor a felhasználó a programot régóta alvó állapotból indítja újra, így ilyenkor nem zavaró az újraindulás. Az újraindításhoz a BaseActivity-ben található goBackToRootActivity() függvényt használom. Minden ilyen hibát naplózok is.

Ezeken felül kiemelt figyelmet szántam a felhasználó által bevihető értékekre is. Ezeknél az ablakoknál mindig csak a megfelelő formátumot fogadom el. Ezekre találhatóak példák a 3.5. és a 3.6. ábrákon:



3.5. ábra – Nem egyezik a két jelszó



3.5. ábra – Adózásnál nem lehet 6-nál nagyobb értéket bevinni, emellett vegyük észre, hogy csak számot lehet beírni

3.4. Felmerült problémák és megoldásaik

Az egyik legbonyolultabb feladat a területek térképen való megjelenítése volt. Maga a kirajzolás nem volt nehéz feladat, de azt tapasztaltam, hogy bizonyos számú téglalap megléte után a telefon nagyon belassult. Ennek az oka a túl sok térképen megjelenő adat volt. Arra az elhatározásra jutottam, hogy mindig csak azokat az információkat jelenítem meg, amit tényleg szükséges, azaz azt, amit a felhasználó láthat. Szerencsére a Google Maps szolgáltat olyan függvényt, amivel azt lehet lekérni, hogy éppen mit lát a térkép kamerája, azaz maga a felhasználó.

Azt az elvet alkalmaztam, hogy amelyik téglalap középpontja pont a megadott négy koordináta között van (ez határozza meg a kamerát), azt jelenítem meg. Ez addig megfelelően működött, amíg rá nem jöttem, hogy a térképet el lehet forgatni és el lehet dönteni is, ami pedig elég fontos dolog. A probléma végül az lett, hogy hogyan állapítom meg egy téglalapról (a terület), hogy benne van-e egy trapézban (a kamera). Több sikertelen próbálkozás után (beépített függvények használata – koordinátákat pontatlanul számol, trapéz kiegészítése négyzetre – szintén nagyon pontatlan számítás) találtam egy módszert, ami megmondja egy pontról, hogy szerepel-e egy adott sokszögben. A módszer neve „Point Inclusion in Polygon Test”[1]. Ezt végül úgy alkalmaztam, hogy egy terület mind a négy sarkát és a középpontját is megvizsgáltam, hogy beleesik-e a kamera által látott területbe. Az algoritmus annyira gyorsan és hatékonyan működött, hogy ezt alkalmaztam a szerveren a tavak vizsgálatához is.

Egy másik, korábbi probléma az adatforgalom mennyisége volt. Ekkor minden adatot a szerverről töltöttem le, viszont a program sok adatnál lassú lett, sok memóriát igényelt, és rengeteg adatforgalmat is generált. Úgy döntöttem, hogy minél több adatot csak egyszer kérek le a szerverről, lokálisan mentek el, és ha változás történik, akkor írom felül. Ehhez használok a kliensen adatbázist. Sajnos még mindig nem sikerült minden ilyen lekérést az adatbázisba vezetni, de az adatforgalmat sikerült töredékére csökkenteni. Az adatforgalmat jelentősen most már csak a Google Maps használja túlzottan, erre a megoldás az offline térkép lesz.

3.5. Tesztelés

Nagyon sok lenne leírni minden tesztesetet, amit elvégeztem a program írása során. Ezért inkább a módszereket mutatnám be néhány példán keresztül.

Mint az előző fejezetben is említettem, a legnagyobb problémát a területek megjelenítése okozta. A végül kiválasztott „Point Inclusion in Polygon Test” algoritmust nagyon sokszor alkalmazom, ezért kulcsfontosságú a helyes működése és a gyorsasága is a mobileszközön is. A függvény a következő módon néz ki:

boolean isPointInPolygon(Point coordinate)

Azaz bemeneti paraméterként egy koordinátát kér, vissza pedig egy igaz-hamis választ ad. Emellett felhasználja a regionCoordinates tömböt is, ami egy sokszöget tartalmaz, a mi esetünkben azt a trapézt, amit a térkép kamerája lát. A regionCoordinates tömböt a setVisibleRegion(LatLng nearLeft, LatLng nearRight, LatLng farLeft, LatLng farRight) függvény állítja be.

Első körben azt vizsgálom, hogy mi történik, ha nincsenek beállítva a fontos változók:

- regionCoordinates == null: hamis válasz
- regionCoordinates mérete 0: hamis válasz
- coordinate == null: hamis válasz

Ezzel lefedtük azokat az eseteket, amikor érvénytelen adatokkal kellene dolgoznunk. A következő az értelmes adatokkal való tesztelés:

- (0;0),(7;0);(5;5),(2;5) koordinátájú sokszög, (3;2) koordinátájú pont: igaz válasz
- (0;0),(7;0);(5;5),(2;5) koordinátájú sokszög, (2;6) koordinátájú pont: hamis válasz
- (0;0),(-7;0);(-5;-5),(-2;-5) koordinátájú sokszög, (-3;-2) koordinátájú pont: igaz válasz
- (0;0),(-7;0);(-5;-5),(-2;-5) koordinátájú sokszög, (2;6) koordinátájú pont: hamis válasz
- (0,999;0,999),(7,999;0,999);(5,999;5,999),(2,999;5,999) koordinátájú sokszög, (3;2) koordinátájú pont: igaz válasz
- (0,999;0,999),(7,999;0,999);(5,999;5,999),(2,999;5,999) koordinátájú sokszög, (2;6) koordinátájú pont: hamis válasz

Most már ki lett próbálva pozitív, negatív és tört számokra is, a következő a trapéz elforgatása és elnyújtása:

- (1,9;-1,4),(6,7;2,4),(3,5;3,7),(1,5;2,3) koordinátájú sokszög (5,1;1,4) koordinátájú pont: igaz válasz

- (1,9;-1,4),(6,7;2,4),(3,5;3,7),(1,5;2,3) koordinátájú sokszög (-10;-10) koordinátájú pont: hamis válasz

Ezekkel az értékekkel a térképre vonatkozó összes lehetőséget lefedtük, már csak a gyorsaság vizsgálata maradt hátra. Itt láthatóak az eredmények:

Területek száma	**Vizsgált pontok száma**	**Szükséges idő**
26	189	0-2 ms
866	7690	19-50 ms
3464	25000	50-90 ms
11258	101010	190-286 ms

Látható, hogy a gyorsaság még rengeteg adat mellett is megfelelő.

Sajnos az Androidos programoknál majdnem lehetetlen minden esetet megvizsgálni. Nagyon sok fajta eszköz létezik és szinte mindegyiken más verziójú operációs rendszer fut. Ezért úgy láttam helyesnek, hogy játékosokat is bevonok a tesztelésbe, hátha találnak olyan hibát, ami felett elsiklott az én figyelmem. A cél az volt, hogy ezek a tesztelők passzívan – azaz a tudtuk nélkül vizsgálják a programot. Ezt az Acra nevezetű könyvtárral sikerült is megoldani, ugyanis a problémákról üzenetet küld a szervernek, amit én később meg tudok vizsgálni. Ez a fajta módszer és a naplózás az alapvető teszteléseken kívül rengeteget segített a hibák fellelésében.

3.6. Hogyan tovább?

A játék egyelőre nagyon kezdeti fázisban van. Szeretnék később mindent, amit itt felsorolok, megvalósítani, hogy tényleg élvezhető legyen. Az egyik számomra fontos dolog az, hogy közösségi játékot varázsoljak a jelenleg egyszemélyes játékból. Mindenképpen szeretném, hogy a gyűjtögetésekhez, építkezésekhez elengedhetetlen legyen más játékosokkal való összefogás.

Ugyanakkor vannak konkrétabb ötleteim is, amik a további verziókra maradtak:

- A szervert szeretném lecserélni tcp alapúra, hogy a lehető leggyorsabban működjön a kommunikáció és legyen lehetőség visszahívásra is – azaz arra, hogy a szerver hívja meg a klienst.
- Szükség van egy háttérben futó service-ra, ami kezeli az időigényes interakciókat.
- Legyen lehetőség az időigényes interakciók automatizációjára, azaz arra, hogy több ilyen interakciót sorba tudjak állítani, majd az a program önállóan el tudja végezni.
- Több épületet lehessen építeni
- A felület átszabása, például a raktár kivezetése a térképre, oldalról beúszó menüvel, illetve a terület oldalának teljes áttervezése

4. Összefoglalás

Mindig is érdekeltek az okostelefonok különböző szenzorai. Amióta rendelkezem Androidos telefonnal, azóta ötletelek állandóan, hogy miként tudnám én azokat ügyesen kihasználni. Most a szakdolgozattal kapcsolatban megadatott az a lehetőség, hogy az egyik ilyen szenzorhoz, a GPS-hez készítsek egy olyan játékot, ami nem csak az én, hanem mások szórakozására is válhat.

A feladat elkészítése során sok különböző eszközzel ismerkedtem meg, beleláttam, hogy működnek a térképek, a nagyobb adatbázisok. Tapasztalatot szereztem arról, hogy hogyan érdemes nekiállni egy szerver használatához, éles működésben való módosításhoz, és hogyan lehet egy saját ötletet olyan formába önteni, hogy mások számára is élvezetes legyen.

Őszintén remélem, hogy a játék legalább annyi örömet fog okozni a játékosoknak, mint amennyit nekem okoztak az elkészítés utáni sikerélmények és a pozitív visszajelzések.

Néhány vélemény a játékról:

„Nagyon tetszik a játéknak az összképe mely lehetőséget ad arra, hogy beleéljük magunkat a karakterbe azzal, hogy a helyszíneken együtt mozoghatunk. Ezen felül tetszik a kis cuki emberke és nagyon tetszenek a színes kis téglalapok.” Csatári Orsolya, 17

„Nagyon megfogott a játék különlegessége, nem ücsörögni kell felette, ha játszani szeretnénk, hanem kihív minket a szabadba, a körülöttünk levő nevezetességeket ismerhetjük meg játékosan Különleges ötletnek tartom,és még lehet továbbfejleszteni, ami előny szerintem. Egyszerűnek tartom, mégis inspirálónak, ezért tetszik. Szívesen ajánlanám a barátaimnak, mert szerintem nekik is tetszene.” Réti Noémi, 18

„Ötletes, hogy egy online játékban fizikai mozgás révén juthatunk előrébb, más játékokkal szemben nem csak taktikázásról és gombok nyomogatásáról szól, hanem valódi teljesítményt is igényel. A magával ragadó játékmenet a mindennapi mozgásterünkön felül újabb és újabb területek és látványosságok felfedezésére késztet, ezáltal a helyismeretünk is fejlődik akár saját településünkről akár egy teljesen idegen környékről van szó.” Székely László, 23

5. Akik segítettek

Irodalomjegyzék:

[1] http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html 2014. 02. 22.

Az elkészítés során a következő eszközöket használtam fel:

- Android SDK (http://developer.android.com/sdk)
- PhpStorm (http://www.jetbrains.com/phpstorm)
- NotePad++ (http://notepad-plus-plus.org)
- ACRA (https://github.com/ACRA/acra)
- OpenStreetMap (http://openstreetmap.org)
- Overpass API (http://overpass-api.de)

Ezúton szeretném megköszönni Dr. Istenes Zoltánnak, hogy a témavezetésem elvállalta és mindig a segítségemre volt a dolgozat írásakor.

Továbbá köszönöm Székely Lászlónak a rengeteg ötletet és a segítséget a tesztelésben.