MŰEGYETEM 1782

Elektromos roller töltőhálózat tervezése crowdsourcing alapokon

Készítette: Nagy Konstantin Olivér (GW5EL3)

Konzulens: Dr. Biczók Gergely

2015.10.26.

Tartalomjegyzék

1. Bevezető

Mi a baj napjaink közlekedésével? Erre sokan sok különféle választ tudnának adni, pedig eléggé egyértelmű, hogy mi a gond. A legtöbben egyedül ülnek autóikban, miközben az autók dudáit nyomkodják. Erre sokan javasolták már a múltban is azt, hogy használják a tömegközlekedést, de szerintem a tömegközlekedés, akármennyire jó is, nem képes átalakítani azt, hogy miként közlekedünk. Úgy vélem, hogy egy jobb alternatívát tudok mutatni, méghozzá az okos, elektromos rollerek képében.

Az autók és buszok nagy hátránya, hogy a legtöbb esetben füstfelhőt hagynak maguk mögött, amivel szennyezik a környezetet is. Erre természetesen a megújuló erőforrásokból előállított elektromos árammal működő járművek a megoldás, de ez hatékonyan nem összeegyeztethető azzal, hogy az emberek egyedül utaznak minden nap a munkába.

Egy elektromos autó akkumulátora napjainkban még lassan töltődik fel és az elektromos autó ugyanúgy beszorul a forgalmi dugóba, így nem old meg semmit tulajdonképp a környezetszennyezésen felül.

Az elektromos autó előnye természetesen az, hogy fedett és klimatizált, ezeket az előnyöket az okosroller természetesen nem tudja felmutatni, de ennél sokkal fontosabbat tud: friss levegőt biztosít használójának. Hiszen ha mindenki áttér az okosroller használatára, akkor annak jó hatása lesz városaink levegőjére is és végre tiszta levegőt szívhatunk majd.

Ha lemerül egy elektromos autó, akkor nincs mit tenni, bár lehet tolni is, de az nem igazán komfortos. Ezzel szemben egy elektromos roller egyáltalán nem teher, könnyedén hajtható lábbal is, amennyiben az akkumulátora lemerülne.

1. kép

Vegyük példának a CityBug elektromos rollerjeit (1. kép), melyek könnyű és kompakt kialakításuk ellenére is akár 20 km megtételére is alkalmasak egy feltöltéssel és kevesebb, mint 13 kilogrammot nyomnak, így tényleg könnyedén hajthatóak lábbal is. Most lehet, hogy felmerült Önben, hogy az okosroller tulajdonképp miért is különleges. Elektromos kerékpárok már viszonylag régóta kaphatóak és mégsem változtattak meg semmit.

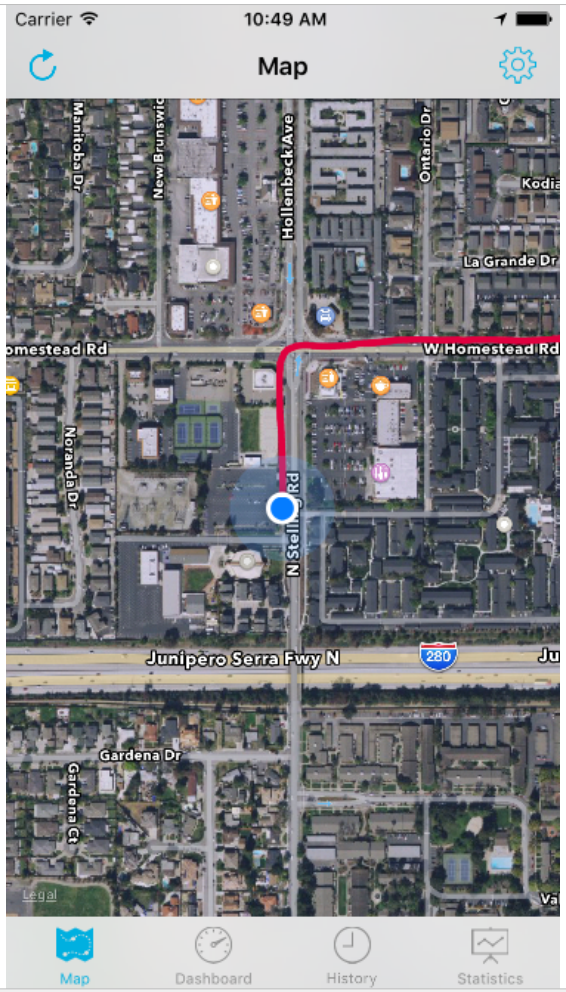
Nos, az elektromos rollerek és egészen pontosan az okosrollerek két előnyt is fel tudnak mutatni az elektromos kerékpárokkal szemben. Az idősek számára kényelmetlen felülni egy kerékpárra egy bizonyos kor után, míg a rollerre egyszerűen csak rá tudnak állni.

Ahogy azt fentebb írtam, az okosrollerben megvan a potenciál arra, hogy átalakítsa a tömegközlekedést, de ennek ellenére a villamosok és metrók nem fognak eltűnni a Földről. Az okosroller pedig ekkor jön majd jól, hiszen könnyedén összecsukható és hónalj alá csapva vihető akár a tömegközlekedési járművön is a többi utas nyugalmának megzavarása nélkül is.

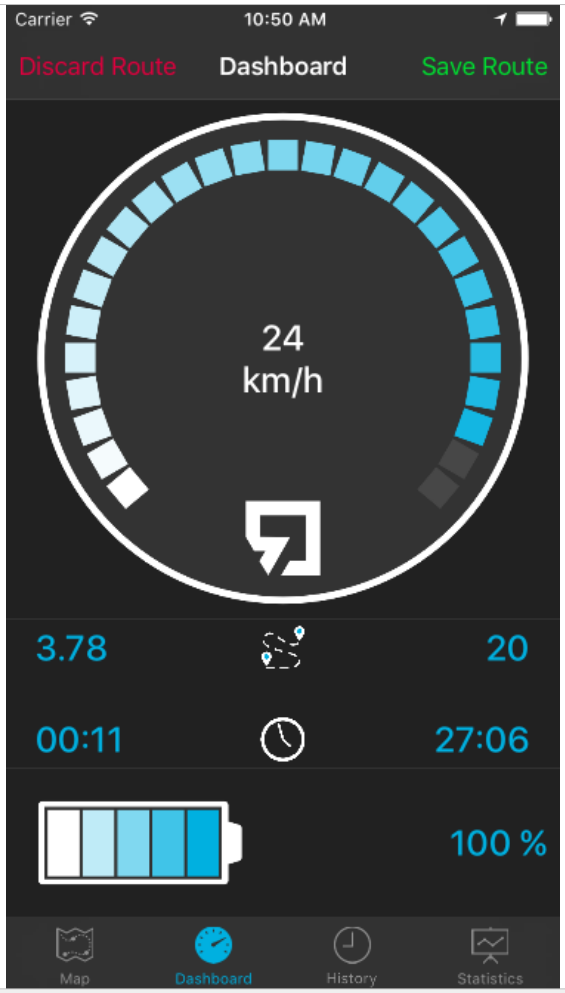
2. iOS alkalmazás bemutatása

A szebb jövő okosrollere igényel egy iOS alkalmazást is természetesen, mellyel bővíthető az élménye.

Az alkalmazás ehhez biztosít műszerfalat, korábbi útvonalak visszanézését, statisztikákat és anonim statisztikagyűjtést.

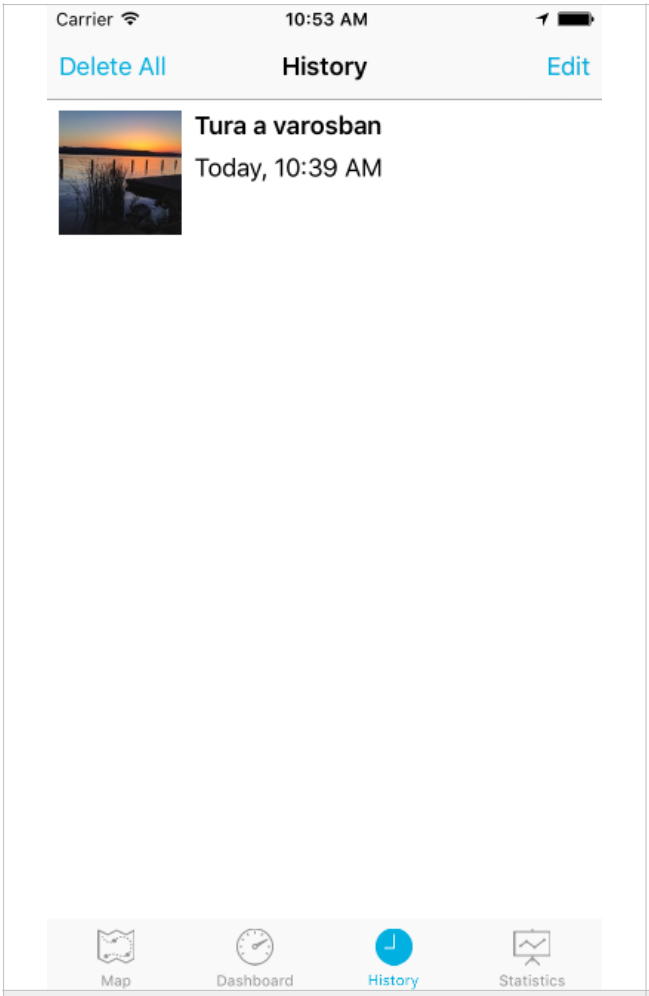


2. kép

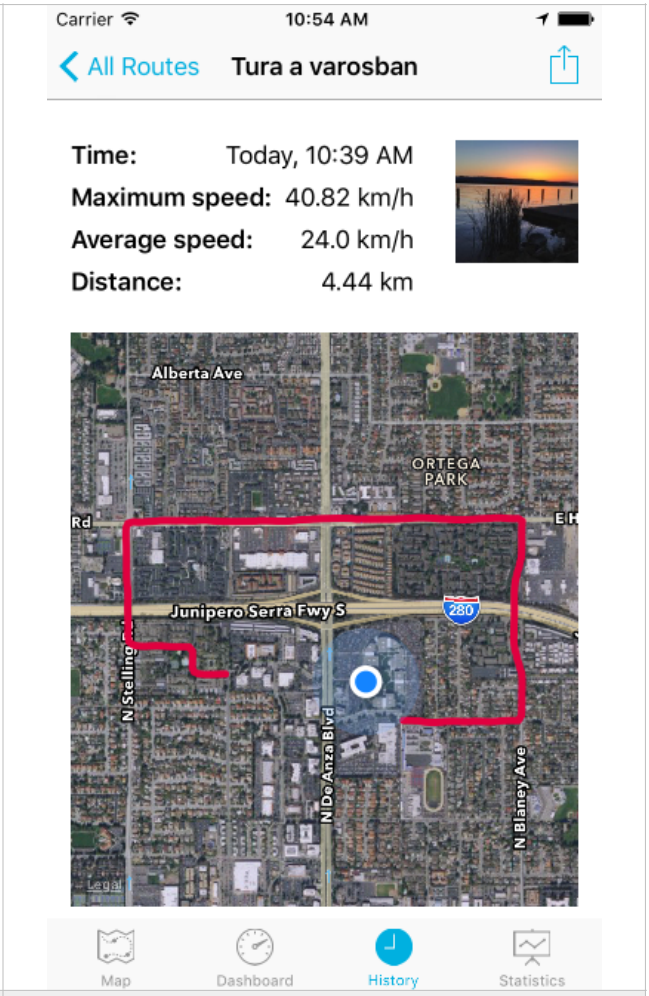


3. kép

Az alkalmazás első fülén egy térképen látható (2. kép), hogy a felhasználó milyen útvonalon közlekedik. A második fül a műszerfal része az alkalmazásnak, ahol látható az okosroller aktuális sebessége, a megtett út, eltelt idő és a hátralévő út, valamint a hátralévő idő (3. kép). Utóbbi adatokat az okosroller akkumulátorának állapotából számítja ki az alkalmazás, melyet a roller Bluetooth kapcsolaton keresztül továbbít az alkalmazás számára.



4. kép



5. kép

A műszerfalon van lehetőség útvonalfelvétel kezdésére is, így rögzíthető, hogy milyen utat tesz meg a felhasználó rollerével. Ez a funkció az általam létrehozott rendszer működéséhez kulcsfontosságú lesz a későbbiekben. Az alkalmazás harmadik fülén (4. kép) tekintheti át a felhasználó, hogy milyen útvonalakon járt már korábban. Itt érdemes megemlíteni, hogy az útvonalakhoz csatolhat a felhasználó fényképet is, valamint az alkalmazás kiszámítja a megtett távolságot, maximális sebességet, átlagsebességet és mentésre kerül a megtett útvonal is (5. kép).

Az alkalmazás negyedik fülén a statisztikák láthatóak, ahol a felhasználó megtekintheti a maximális sebességét, átlagsebességét és megtett távolságát hónapokra bontva.

Amikor ezt az alkalmazást készítettem, beépítettem egy diagnosztikai eszközt, mely a felhasználó beleegyezésével elküldi a rögzített adatokat egy külső szerverre, teljesen anonim módon. Ez a funkció az alkalmazás beállításai közt bármikor ki- vagy bekapcsolható, így tiszteletben tartva a felhasználó döntését.

3. A kitűzött feladat

Az okosrollerek persze önmagukban nem képesek a világ megváltoztatására. Az okosrollerek gyakorlatilag sima elektromos rollerek, melyeket kiegészít a gyártó egy Bluetooth modullal. Ez persze nem változtatja meg a tényt, hogy ezeket a rollereket tölteni kell, hogy működjenek.

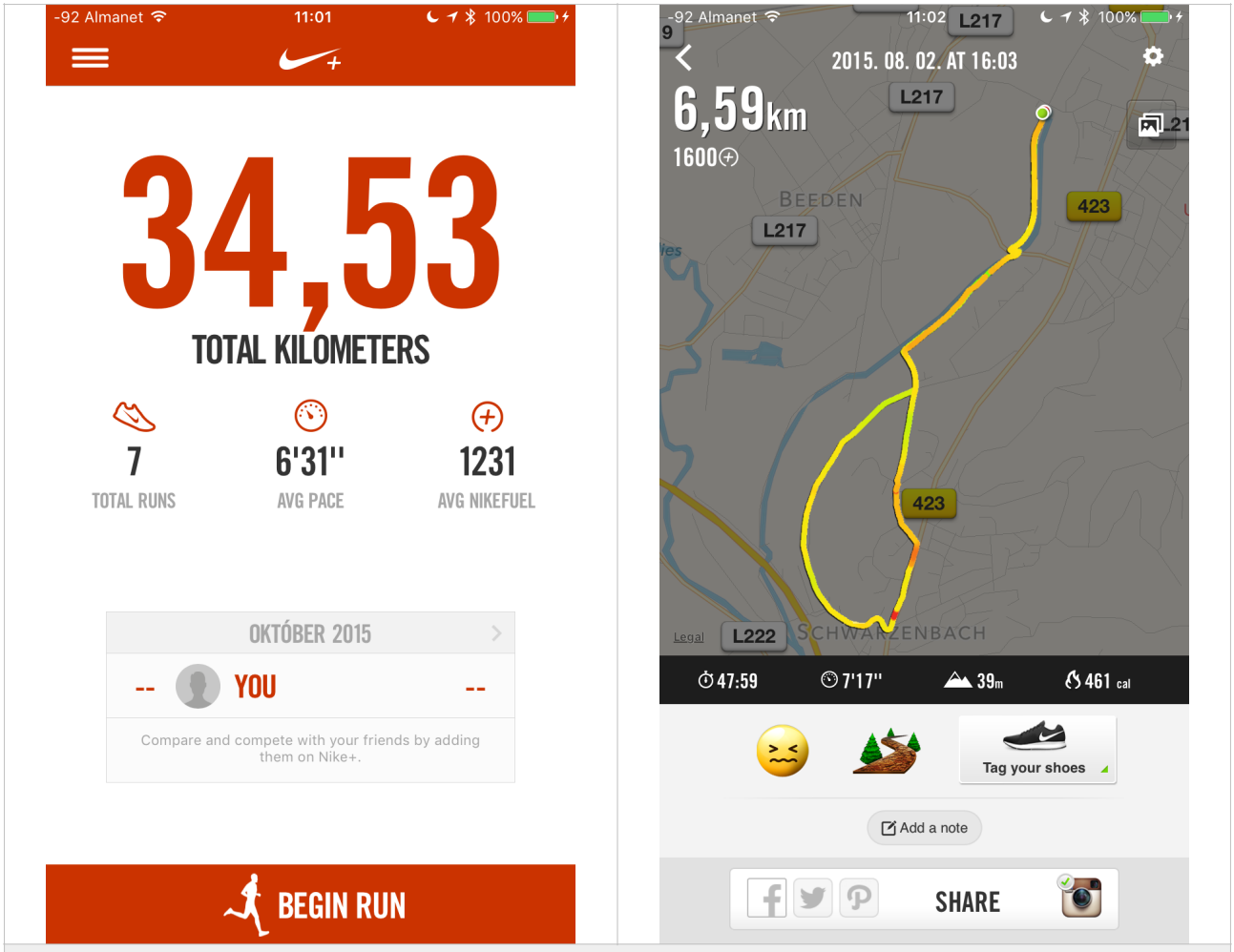
Innentől már egyértelmű volt, hogy milyen irányba kell elindulnom: a szerverre begyűjtött névtelen útvonalak alapján meg kell állapítanom, hogy hova érdemes töltőállomást helyezni, ahol a felhasználók feltölthetik rollerjüket, ha az alkalmazásban úgy látják, merül annak akkumulátora.

Csak olyan helyre érdemes töltőállomást helyezni, ahol sok felhasználó megfordul, vagyis a mentett és feltöltött útvonalak alapján kell kiszámolni, hogy mik az optimális helyek egy városon vagy községen belül egy töltőállomás elhelyezésére.

3.1 Konkurencia

Miután okosroller jelenleg még nem létezik a piacon, így okosroller műszerfal és útvonalrögzítő alkalmazás sincs, de vannak alkalmazások, melyek hasonló funkcionalitást valósítanak meg.

Vannak például olyan alkalmazások, melyek futás közben követik a felhasználó helyzetét, majd a rögzített útvonalból készítenek kimutatásokat. Ezek közül az egyik legnépszerűbb a Nike+ Running alkalmazás (6. kép).

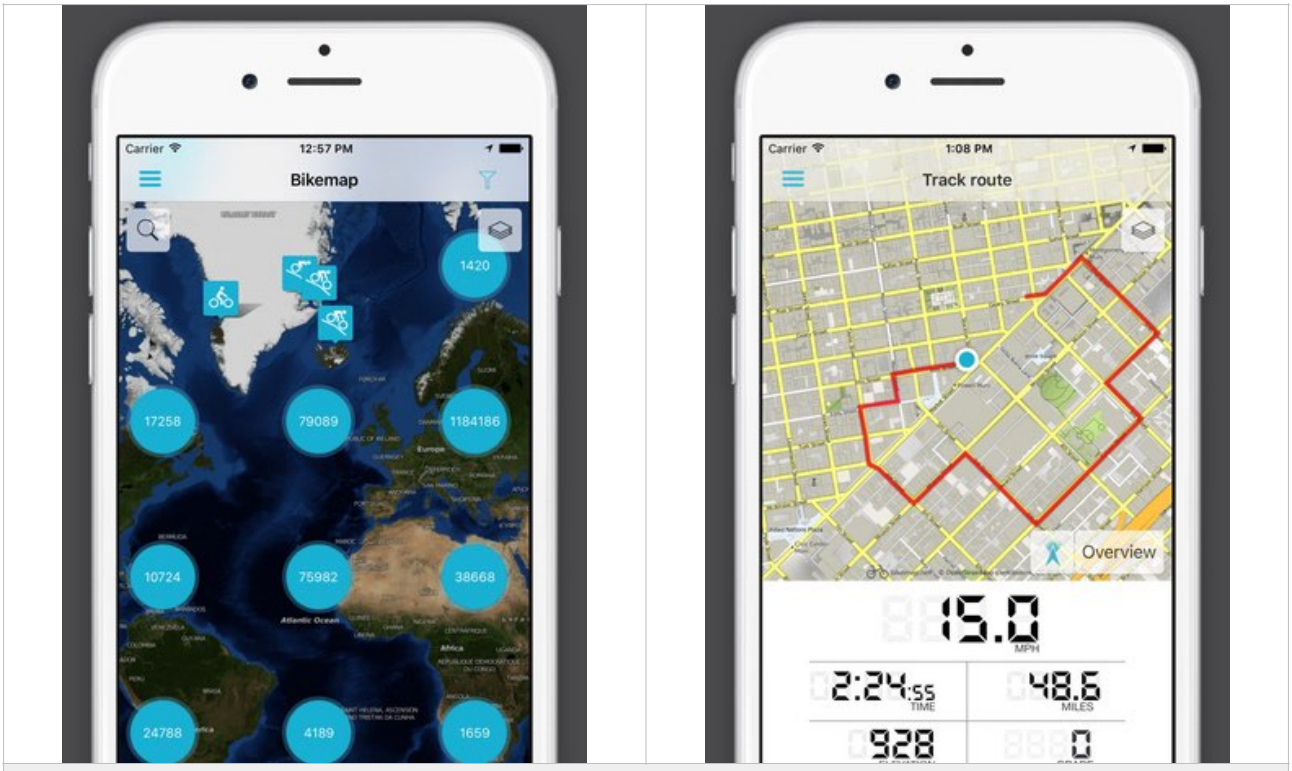


6. kép: A Nike+ Running alkalmazás

Az alkalmazásban létrehozhatunk új útvonalat, menthetjük a rögzített útvonalat, feltölthetjük a mentett útvonalat Nike+ fiókunkba és persze meg is oszthatjuk az útvonalainkat barátainkkal a népszerű közösségi oldalakon.

Lényegi eltérés, hogy a Nike+ nem kíván anonim lenni, a felhasználók és a hozzájuk tárolt adatok nincsenek elválasztva, így a Nike bármikor megnézheti, hogy melyik felhasználó merre futott az alkalmazás használata közben.

Kicsit közelebb áll koncepcióban a Bikemap alkalmazás és szolgáltatás, mely a Nike+ Running alkalmazáshoz hasonló felépítésűnek tűnhet elsőre, de itt a kerékpáros útvonalak rögzítése a cél.



7. kép: A Bikemag alkalmazás

A Nike+ Runninghoz hasonlóan a Bikemap is egy fiókhoz rendeli a mentett útvonalakat, de míg a Nike+ csak barátainkkal osztja meg a futásainkat, a Bikemap minden mentett útvonalat nyilvánosan elérhetővé tesz honlapján a felhasználó nevével együtt. Értelemszerűen ez egy teljesen más adatvédelmi hozzáállás.

Miután az App Store-ban több mint 1,2 millió alkalmazás található, így lehetetlen lenne végigvenni az összes futást, kerékpározást vagy egyéb sportot rögzítő alkalmazást, de az útvonalrögzítők kategóriájában a Nike+ Running és a Bikemap a legnépszerűbb.

3.2 Adathalmaz megszerzése

Ahhoz, hogy a feladatot el tudjam végezni, kellett egy adatbázis, melyen tesztelhető az általam írt algoritmus. Miután az okosroller még nem létezik és nincsenek is az alkalmazásnak felhasználói, így mesterségesen kellett előállítanom az adathalmazt, amin tesztelhettem az algoritmust.

Tudtam, hogy a legtöbb feltöltött útvonal a világhálón a GPS navigációk többsége által használt, népszerű GPX formátumban érhető el, ezért írtam egy GPX parsert, mely átalakítja a a kapott GPX fájlokat a rendszerem által használt adatbázis számára emészthető formátumúra.

Ekkor még nem találtam sehol se megfelelő forrást, ahonnan nagy mennyiségben tölthettem volna le GPX fájlokat. Ebből a szempontból pont kapóra jött a fent említett

Bikemap alkalmazás, mert honlapjukon több mint 100 egyéni útvonal van, amit a felhasználók feltöltöttek már Budapestről.

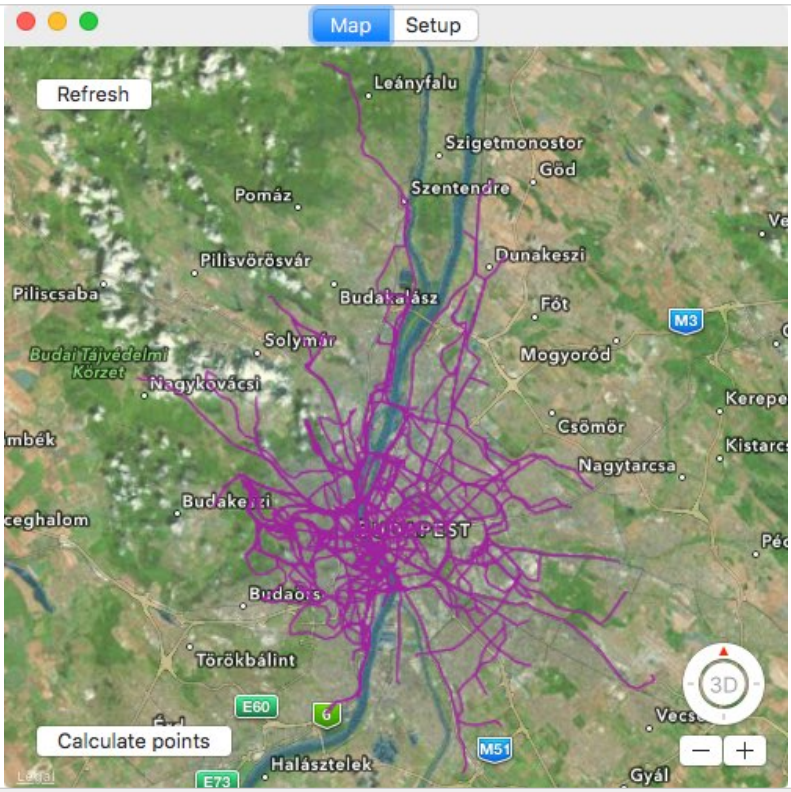
Úgy véltem, hogy a kerékpáros útvonalak kellően hasonlítanak arra, amiket az okosrollerrel tesznek majd a felhasználók, így egy reális képet kaphatunk arról, hogy hova kéne elhelyezni az okosroller töltőállomásokat Budapesten.

A Bikemap oldaláról ezeket a GPX fájlokat letöltve azonban szembesültem a ténnyel, hogy a Bikemap a GPX formátum nem sztenderd változatát használja, így írtam egy szkriptet, mely képes átalakítani a Bikemap által rögzített útvonalakat a GPX parserem számára kompatibilis formára.

Ezután már képes voltam feltölteni a rendszer adatbázisát. Jelen esszé írásakor a Bikemap adatbázisából 188 egyéni útvonalat emeltem át a budapestiek közül. Ezek mind 0-20 km hosszúságúak, mivel a legtöbb elektromos roller hatótávolsága is 20 km körüli.

3.3 Megoldási ötletek

A legtöbb problémára több megoldás létezik, számomra is egyértelmű volt, hogy a gyakori helyek megtalálása nem egy egyértelmű feladat.

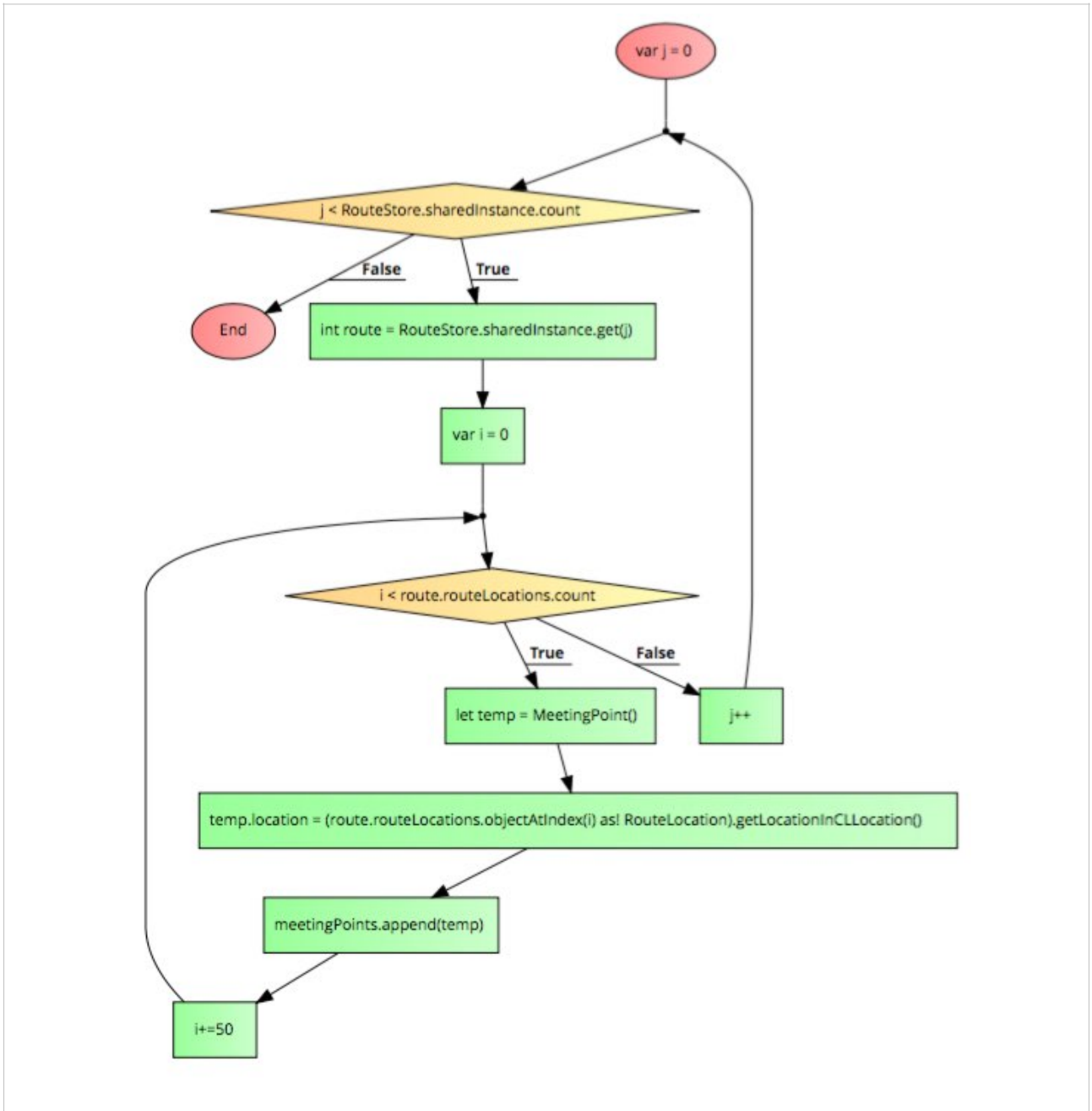


8. kép: Budapest és a bevitt útvonalak

Már csupán a térképre nézve látható (8. kép), hogy ránézésre nem eldönthető, hogy hol vannak gyakori helyek, mert az útvonalak folyton keresztezik egymást.

Hasonló probléma az ismert *Facility location problem*, mely több szempontot figyelembe véve keres optimális helyet adott számú objektumnak. Jó példa erre a gyárak elhelyezése, ahol figyelembe kell venni, hogy az alapanyagok szállításakor a mérgező anyagokat ne lakóövezet mellett kelljen szállítani és a gyár messze legyen a konkurens gyáraktól is.

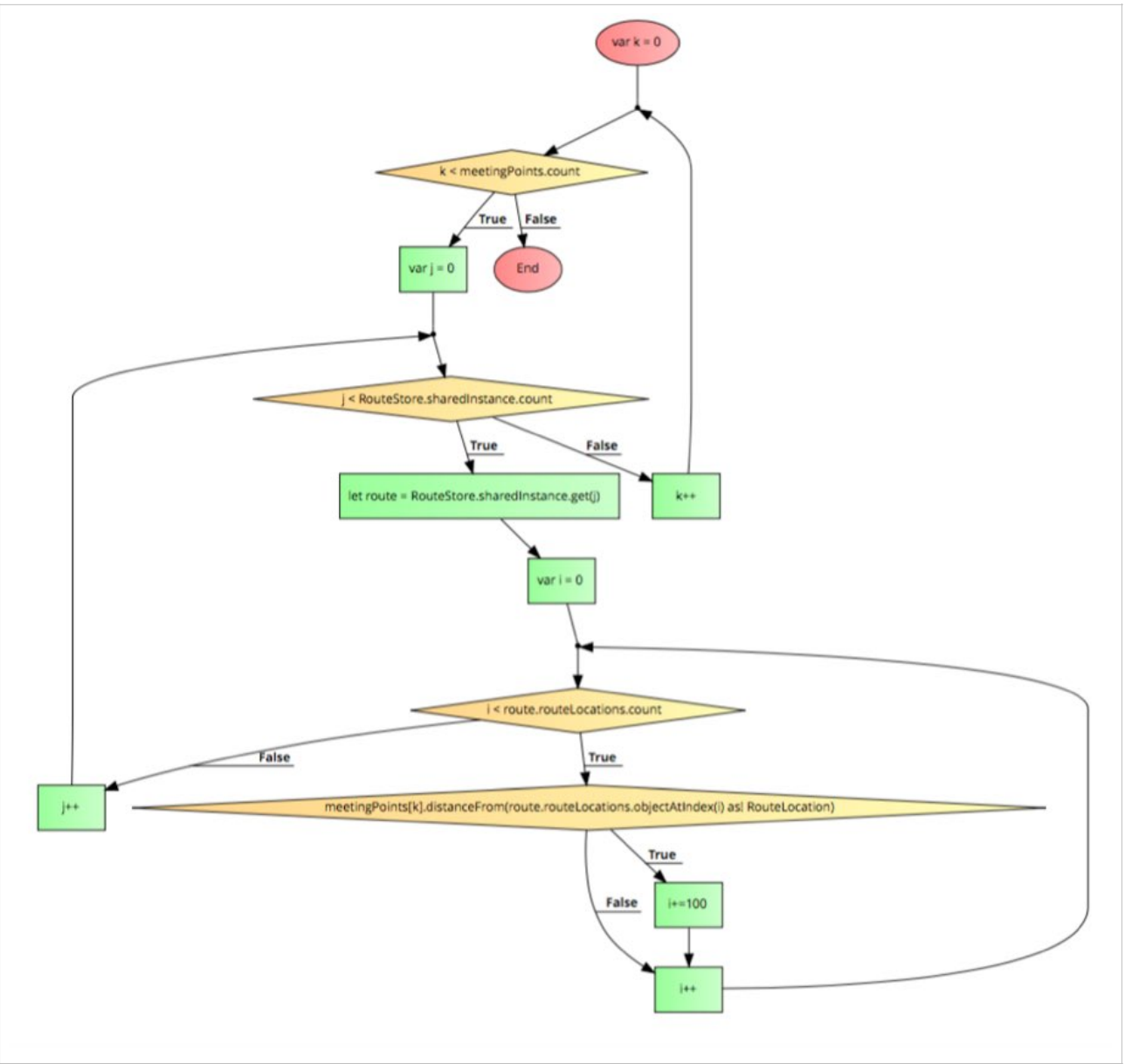
Két különböző elgondolás alapján indultam el. Az első algoritmus a rögzített GPS koordinátáknál figyelembe veszi azt is, hogy azok mely útvonalból származnak. Működése:



9. kép

Először a meetingPoints tömbbe elhelyezem az összes útvonal 50-ik mentett GPS koordinátáját. Ez egy kiindulási alapot ad a gyakori helyek megkereséséhez (9. kép).

Második lépésként a meetingPoints tartalmát súlyozom aszerint, hogy milyen gyakran halad el mellettük útvonal, figyelve arra, hogy a egy útvonal sokáig volt egy gyakori hely jelölt közelében, akkor se növelhesse meg túlságosan annak súlyát.

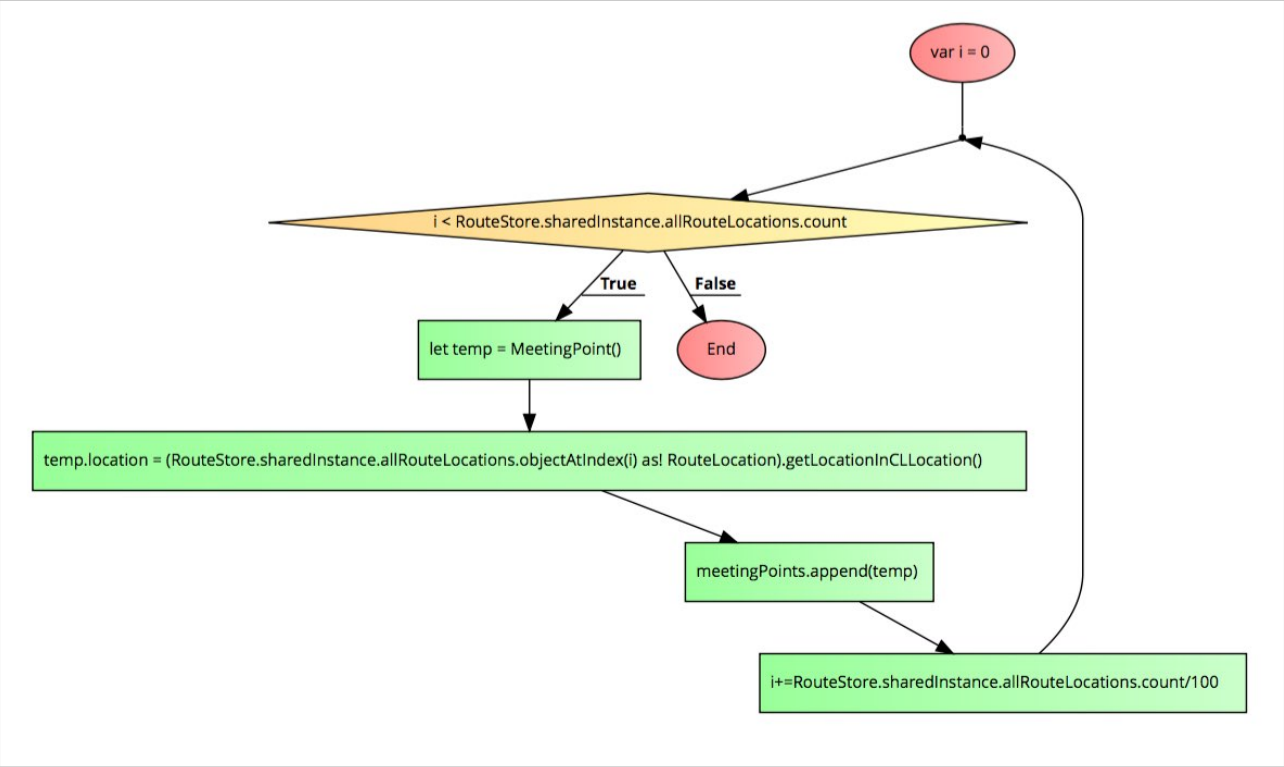


10. kép

Az utolsó teszteléskor a distanceFrom függvény igaz értékkel tér vissza, ha a kapott koordináta közel található hozzá. Ekkor az útvonal felderítésében 100 pontot előre ugrunk, amivel garantáljuk, hogy ezt a pontot ez az útvonal már nem fogja befolyásolni. Azért nem lépünk ki ekkor a ciklusból, mert egy útvonal akár több potenciális gyakori hely környékén is elhaladhat (10. kép).

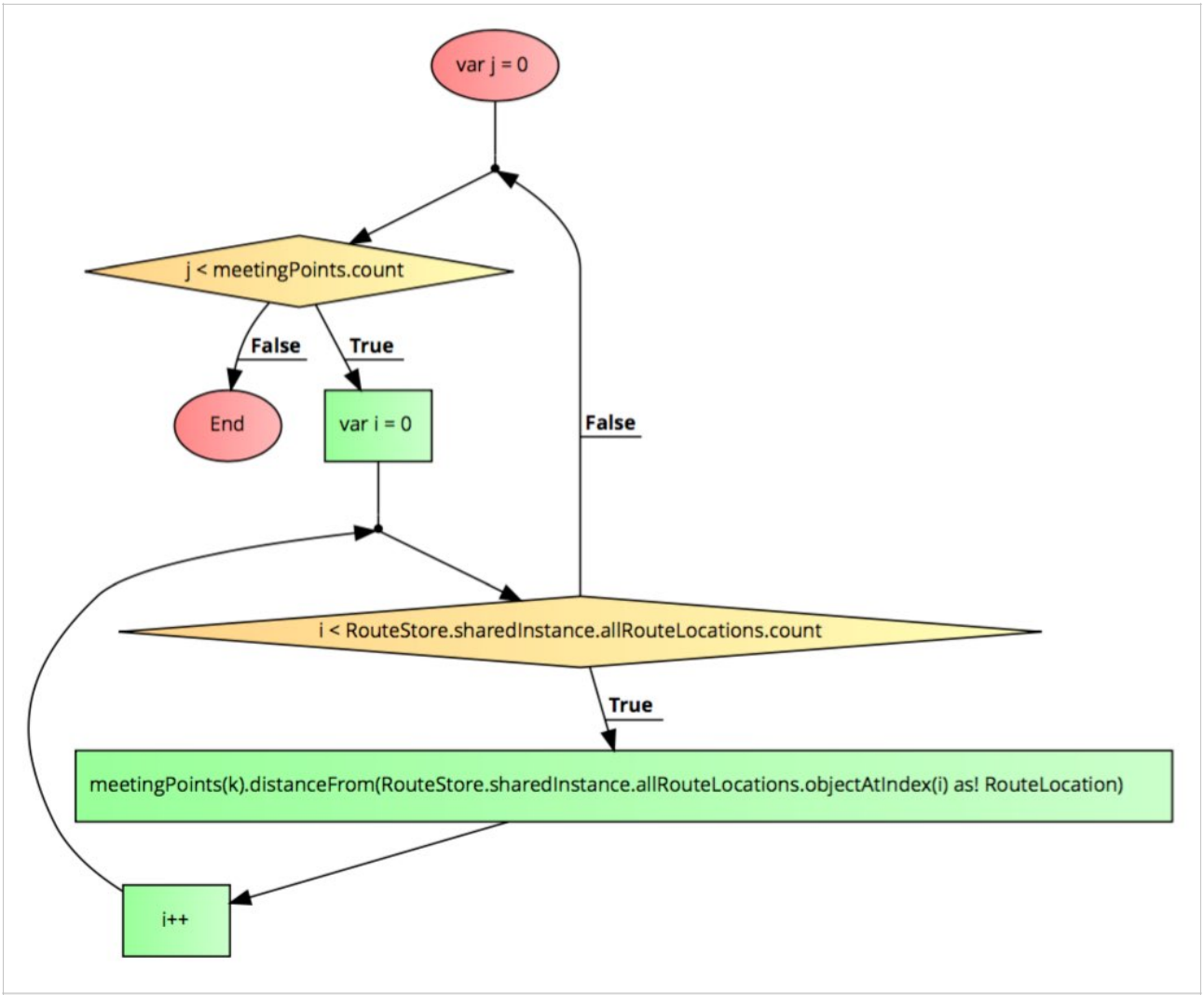
Ezzel be is fejeztük a gyakori helyek jelöltjeinek súlyozását. A második algoritmus eltekint attól, hogy a koordináták útvonalakhoz vannak kötve, valamint korlátozza a gyakori helyeket 100 kiválasztott pontra.

A gyakori helyre jelölt pontok közül mindenféleképp 100-at választunk ki, akárhány koordináta is legyen az adott városban. (11. kép)



11. kép

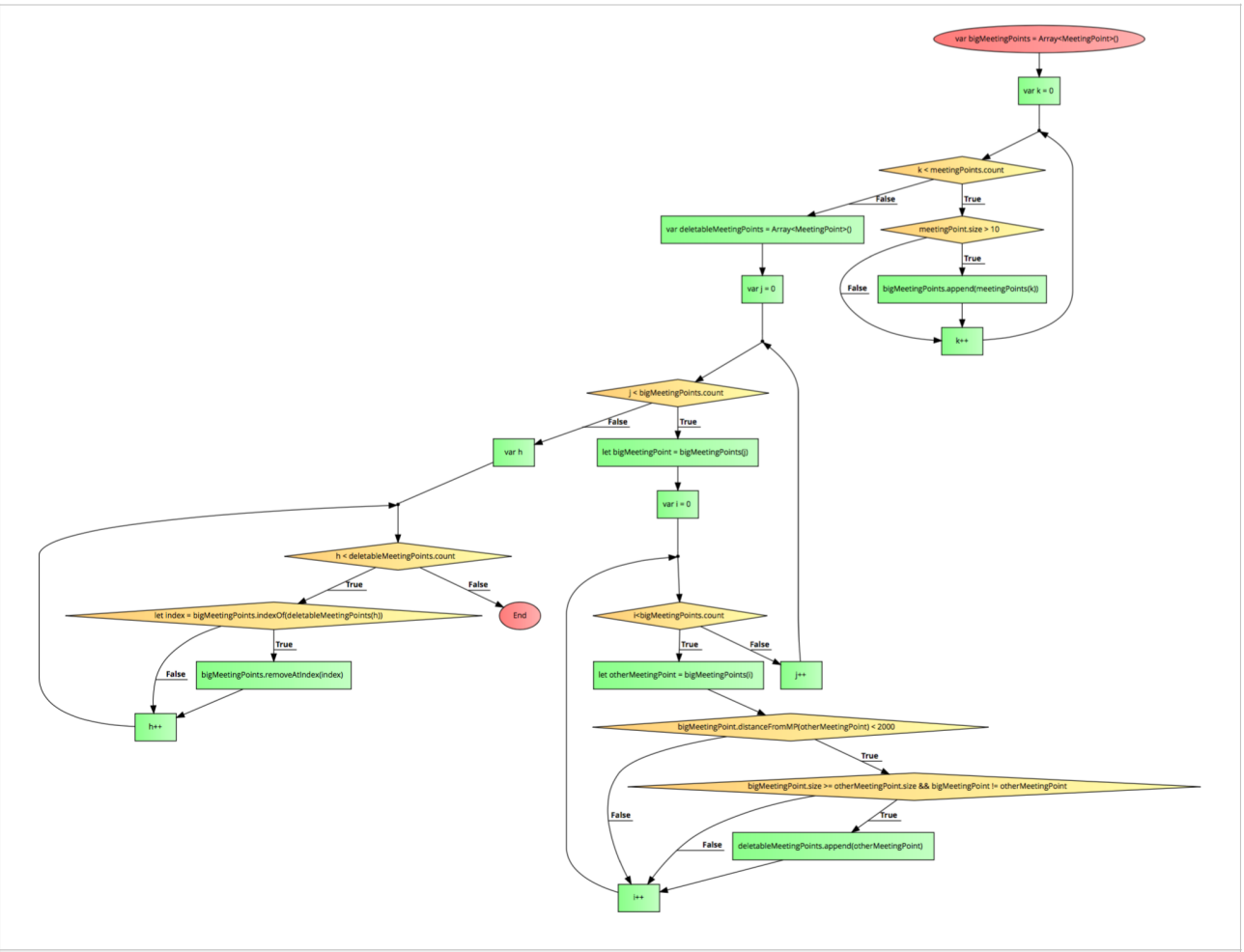
A gyakori helyek súlyozása is sokkal egyszerűbb, ha eltekintünk attól, hogy ezek a koordináták egy útvonalból érkeznek. Ekkor elég végignézni, hogy az egyes jelöltekhez hány közeli koordináta található.



12. kép

Itt kihasználom a distanceFrom függvény mellékhatását, mely közeli hely esetén megnöveli az adott gyakori hely súlyát is (12 kép).

A gyakori helyek súlyozása után mindkét esetben ugyanúgy történik a tényleges gyakori helyek felfedezése: a nagyobb súlyú helyeket megtartjuk, a kisebb súlyúakat pedig eltávolítjuk a tömbből. Ez azonban csak a feladat fele, hiszen miután megtartottuk a nagyobb méretűeket, törölnünk kell azokat a jelölteket, amik túl közel vannak egymáshoz.

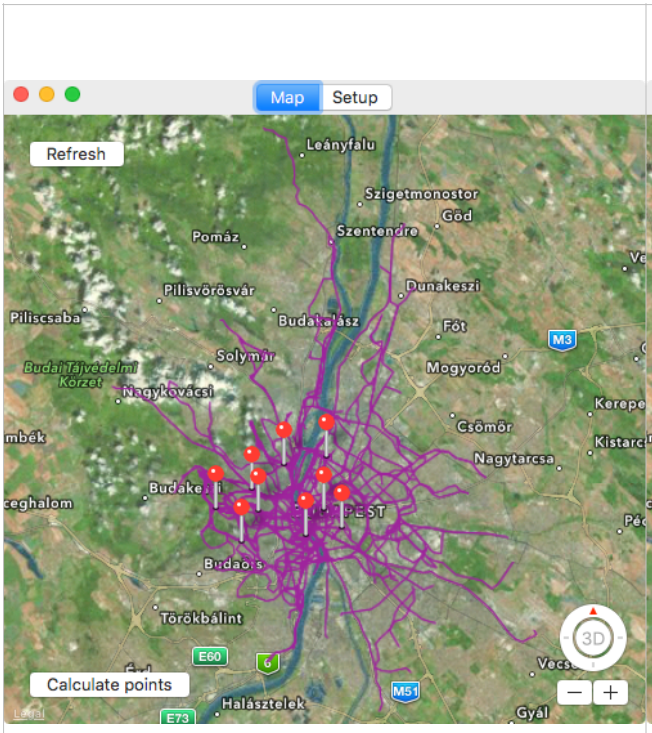


13. kép

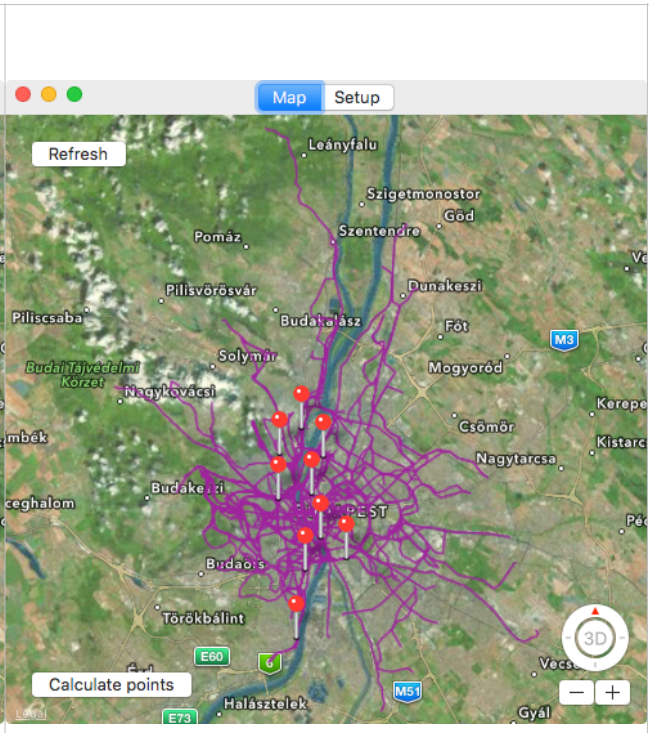
3.4 Algoritmusok összehasonlítása

Egyértelmű, hogy az algoritmusok eltérő bonyolultságúak és hatékonyságúak. A Facility location problem megoldására talált algoritmusok megvizsgálása után arra a következtetésre jutottam, hogy annak implementálásától eltekintek.

Két alapvetően különböző elgondolás alapján két algoritmust készítettem, melyek hatékonyságában és pontosságában is nagy eltérés mutatható ki, de a végeredmény mégis mindkét esetben használható lett.



14. kép: Első algoritmus



15. kép: Második algoritmus

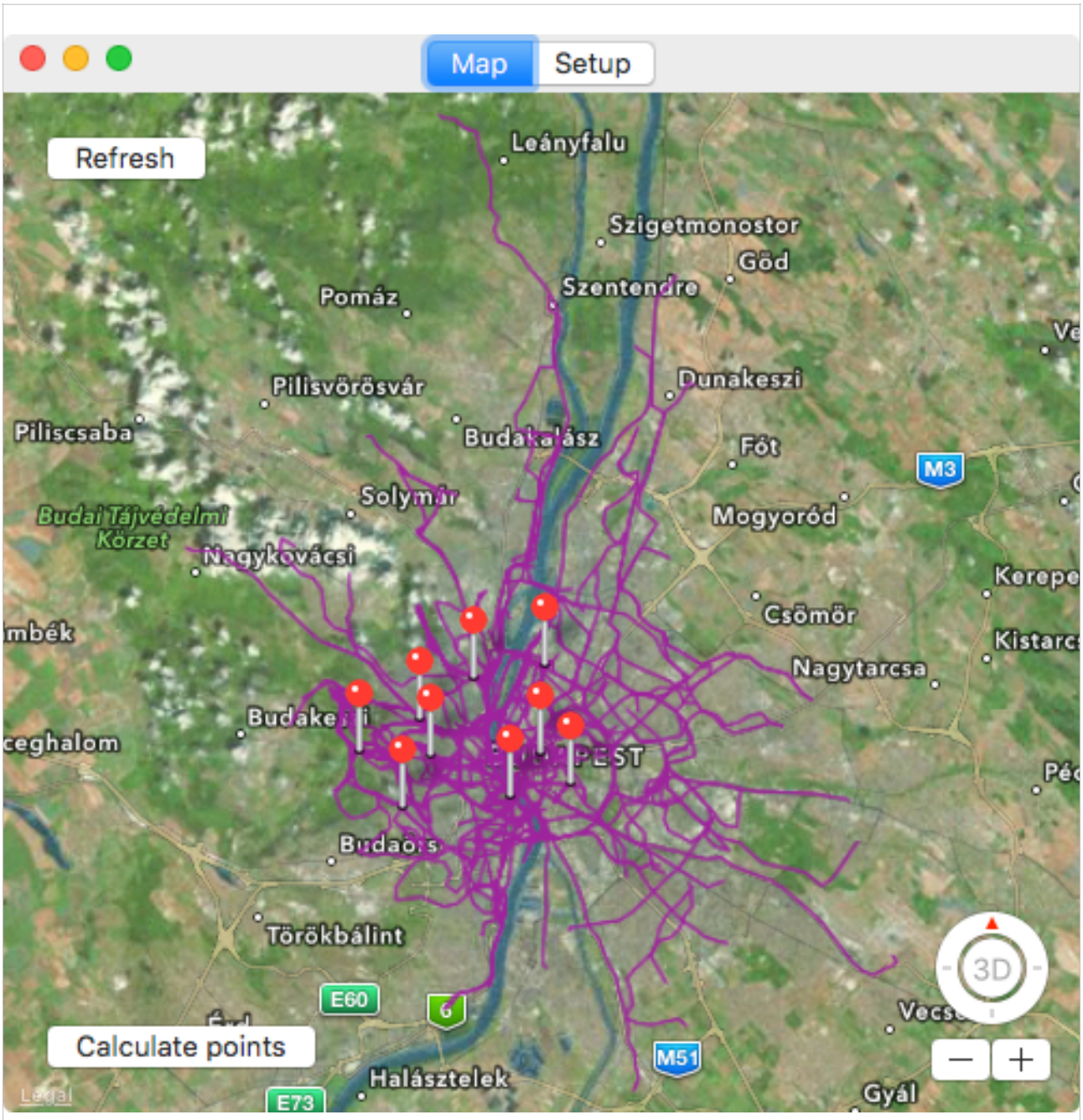
Mindkét algoritmus hasonlóan gyakori pontokat talált meg, de a másodiknak jelentősen alacsonyabb a komplexitása.

Az első algoritmus esetén a gyakori pontok súlyozása gyakorlatilag egy $O(n^3)$-ös algoritmus, míg a második csupán négyzetes komplexitású. Ez futásidőben az általam használt 188 mentett útvonal esetén 6-szoros különbséget eredményezett.

Így úgy döntöttem, hogy a másodikat hagyom meg a végleges rendszerben.

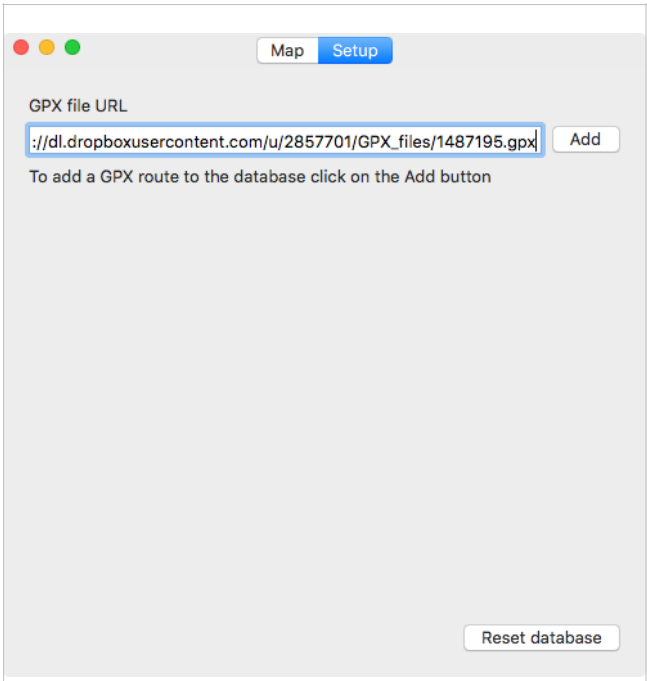
3.5 Az elkészült asztali alkalmazás

Miután a szerverről csak a roller gyártója fog végezni, így a felhasználói felület kialakítása itt közel sem volt annyira hangsúlyos, így teljes mértékben az Apple által biztosított felületi elemeket használtam fel.

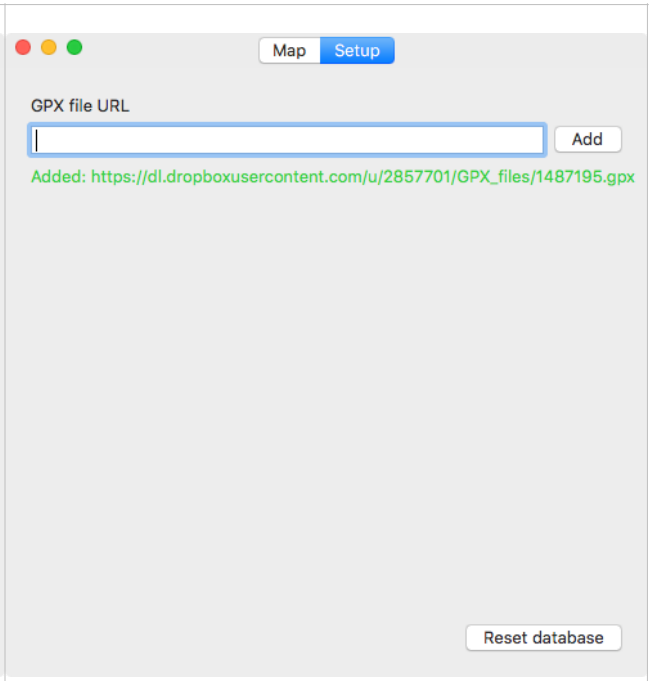


16. kép: Az alkalmazás első füle

Az alkalmazás két füllel rendelkezik, ezek közt az ablak felső részén található fülváltóval lépegethetünk. Első fülén látható a térkép, valamint itt helyeztem el a felvett útvonalak alapján gyakori helyeket kereső gomb és egy térképfrissítő gomb.



17. kép: Új útvonal hozzáadása



18. kép: Útvonal sikeres mentéséről szóló üzenet

Az alkalmazás másik fülén találhatóak a beállítások. Itt vehetünk fel új útvonalat az adatbázisba egy GPX fájllal. A GPX fájl bármilyen URL formájában megadható. A beviteli mező alatti szöveg pedig frissül, ha sikeresen vittünk be útvonalat az adatbázisba. Ez a funkció természetesen csupán a diagnosztikai célokat használja, hiszen ha lesznek az alkalmazásnak felhasználói, akkor az azok által rögzített útvonalak automatikusan bekerülnek majd az adatbázisba (17. kép).

Végül, de nem utolsó sorban, a beállítási fülön található egy adatbázis visszaállítására szolgáló gomb is (18. kép).

4. Használt szoftverek és nyelvek

Az Apple eszközökre a legegyszerűbben az Apple saját IDE-jében az Apple saját programozási nyelvén lehet alkalmazásokat és szolgáltatásokat építeni, így az Xcode elérhető legújabb változatát, a 7.1-es változatot használtam és a legújabb Swift verzióban, a 2.1-esben írtam a kliens és szerveroldali alkalmazást is.

Az iOS klienst úgy készítettem, hogy iOS 8 és 9 alatt is fusson, ezzel az aktív felhasználói bázis 91%-át lefedtem az Apple adatai szerint. Az OS X-re írt adatbázis olvasó alkalmazást OS X 10.11-re, a legújabb változatra írtam, mivel annak nem célja, hogy minél több eszközön fusson, csupán a roller gyártójának kell tudnia futtatni.

Az iOS kliens elkészítésekor több különböző szoftvert is használtam a felhasználói felület kialakításához. A felület prototípusának elkészítésére az iAd Producer alkalmazást használtam, mely bár hirdetések készítéséhez készült, nagyon egyszerű vele működő felhasználói felületet modellezni.

Az iOS alkalmazás végleges grafikai felületét az Affinity Photo vektoros képszerkesztővel készítettem el, mivel fontos volt, hogy az alkalmazás minden kijelzőméret és felbontás mellett működjön és jól nézzen ki.

Az iOS alkalmazást teszteltem az iOS Simulator 8.3-as és 9.1-es változatában, hogy mindkét fő iOS verzió legújabb változatán stabilan működjön, saját telefonomon pedig a 8.4-es és a 9.0.2-es rendszerrel.

5. Használt keretrendszerek

5.1 Használt Apple keretrendszerek

Az Apple által biztosított SDK iOS-en és OS X-en is egyaránt szinte mindent tartalmaz, amire egy komplex program elkészítéséhez szükség van. A képek átméretezésétől az animációkig minden könnyen megvalósítható az Apple integrált eszközeinek köszönhetően.

iOS-en a UIKit, CoreLocation és MediaPlayer keretrendszereket használtam fel, OS X-en az AppKit volt az egyedüli iOS-en nem használt keretrendszer, mely a megjelenítésért felel, iOS-en és OS X-en egyaránt pedig a MapKit és Foundation keretrendszereket használtam. Miután a közös keretrendszerek közel megegyeznek a két rendszeren, így nagy a közös kódbázisa a két alkalmazásnak.

A keretrendszerek neve a legtöbb esetben magától érthetődő. A Foundation tartalmazza az alapvető struktúrákat és osztályokat, melyek elengedhetetlenek egy alkalmazás

készítéséhez, a UIKit és AppKit a felület kirajzolásáért felelős, a CoreLocation felel az eszköz pozíciójának meghatározására, a MapKittel történnek a térképpel és pozícióval kapcsolatos számítások.

A MediaPlayer keretrendszert iOS-en kizárólag a tesztelés miatt használtam. Miután jelenleg még nem létezik okosroller, amihez párosíthattam volna az alkalmazást, így a hangerőgombokkal vezéreltem a képzeletbeli roller akkumulátorának töltöttségét.

5.2 Használt harmadik féltől származó keretrendszerek

Amikor a helyi adatbázis kialakítását építettem fel, több népszerű adatbázis-kezelő rendszert kipróbáltam. Az Apple Core Data megoldása túlságosan komplex lett volna a feladat méretéhez képest. Másik lehetőség lett volna az NSUserDefaults segítségével egy XML-hez hasonló property list fájlban tárolnom az adatokat az eszközön. Mindkettőnek meglettek volna a hátrányai, így egy külső adatbázis kezelőt kerestem.

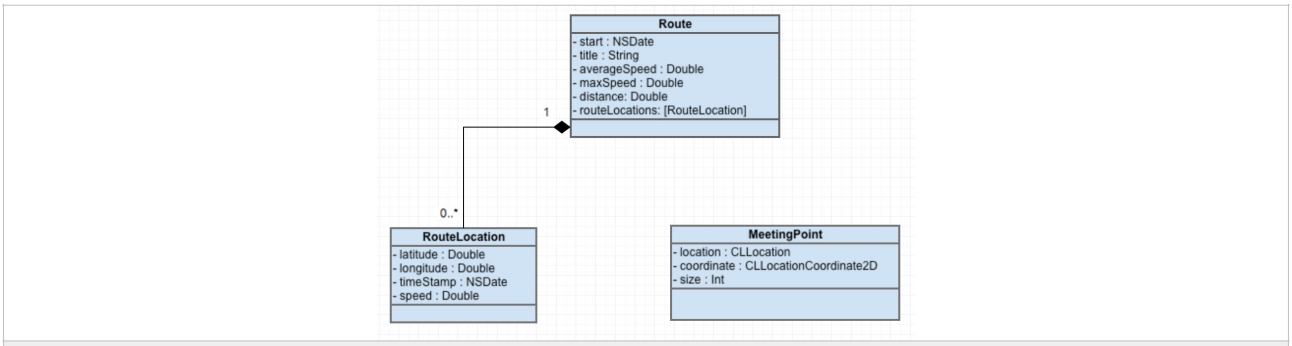
A Realm keretrendszerre esett a választásom, melynek segítségével a RouteStore singleton osztályt könnyen tudtam rekurzívan szerializálni egy adatbázisfájlba, amiből viszonylag egyszerűen, de ami még fontosabb, gyorsan lehet kinyerni az adatokat.

A Realm nem mondható komolytalannak, mivel a Siemens, az Intel, az Ebay és az Amazon is használja, régóta megbízható és kereskedelmi szoftverek számára is ingyenesen használható, Apache 2.0 licenc alatt álló keretrendszer.

A Realm keretrendszert az iOS-es kliensben és az OS X-es feldolgozó alkalmazásban is használtam, értelemszerűen a szerverre is Realm struktúráiban tölti fel az iOS kliens az adatokat.

iOS-en a Realm-en felül a statisztikák kirajzolásához használtam még harmadik féltől származó keretrendszert, méghozzá az ios-charts keretrendszert, ami a népszerű MPÉsroidChart Ésroidra írt diagram rajzoló iOS-re portolt változata. Az ios-charts is az Apache 2.0 licenc alatt áll, így szabadon használhattam az alkalmazásban.

6. Alkalmazott adatszerkezetek



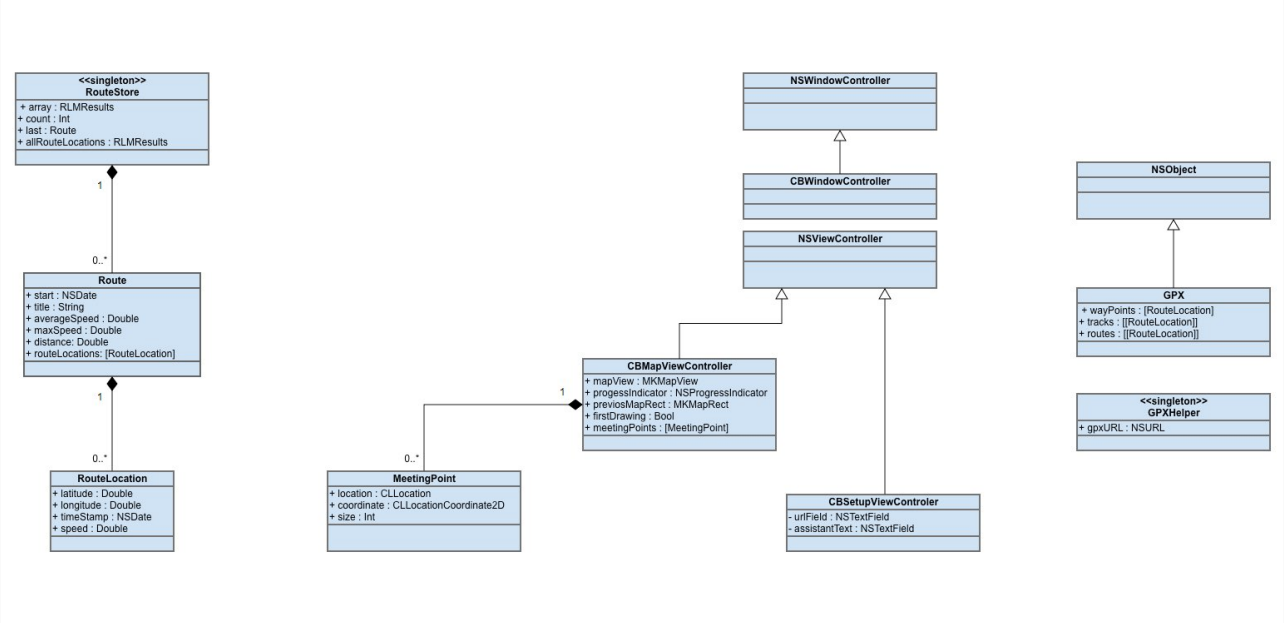
19. kép: Az útvonalak és gyakori helyek tárolásául szolgáló adatszerkezetek

A RouteStore osztály végzi el a Realm adatbázisba mentését a Route objektumoknak. A

Route objektumok lényegében a RouteLocation koordináták tárolására szolgál, kiegészítve néhány információval az úttal kapcsolatban.

A MeetingPoint objektumok nem kerülnek szerializálásra, mivel az adatbázis gyakori frissülése miatt nem érné meg eltárolni őket (19. kép).

7. A Mac-es alkalmazás osztálydiagramja



20. kép: Az allkalmazás osztálydiagramja

Az Apple az MVC mintát javasolja az alkalmazások fejlesztésére, ahol elkülönül a modell, a nézet és a nézetvezérlő. Ahogy látható, ezt a Mac-es alkalmazás teljesen betartja, a nézetért felelős komponensek középen láthatóak, míg a modell bal oldalt (20. kép).

8. Létrehozott fájlok

8.1 Az iOS kliens által használt osztályok és fájlok

Ahogy az látható lesz hamarosan, az iOS alkalmazás is követi az MVC mintát 1-2 kivételtől eltekintve.

Ezen swift fájlokból áll az iOS alkalmazás. A legtöbbjük csak egy osztályt tartalmaz, csak 3 olyan van, melyben több struktúra is található (CBYear, Colors és Route).

- AppDelegate
  – Megvalósítja: UIResponder osztályt, Implementálja: UIApplicationDelegate protokollt.
  – Fő célja: Az alkalmazás betöltése és a felhasználói beállítások betöltése.
- CBTabBarController
  – Megvalósítja: UITabBarController osztályt.
  – Fő célja: A kijelző alján megjelenő fülváltó megjelenéséért felel. Ezen felül lehetővé teszi a gesztussal történő fülváltást is.
- CBYear
  – Fő célja: A statisztikák generálásához biztosít egyszerű naptárat. A statisztika nézet számára olvasható formátumúvá alakítja az adatbázis tartalmát.
- CityBug
  – Fő célja: Singleton osztály, a csatlakoztatott okosrollert reprezentálja az alkalmazás modelljében.
- Colors
  – Fő célja: Az alkalmazás színpalettájának tárolására szolgáló fájl.
- ConnectViewController
  – Megvalósítja: UIViewController osztályt.
  – Fő célja: az okosroller csatlakoztató képernyő vezérlője.
- DashboardViewController
  – Megvalósítja: UIViewController osztályt.
  – Fő célja: A műszerfal képernyő vezérlője. A műszerfal megjelenésének nagy részéért ez az osztály felel.
- DetailsController
  – Megvalósítja: UIViewController osztályt, Implementálja: UIGestureRecognizerDelegate és MKMapViewDelegate protokollokat.
  – Fő célja: Részletes információkat jelenít meg egy útvonalról, ha rábök a felhasználó.

- DiagnosticsSender
  - Fő célja: Singleton osztály, ez felel az anonim adatok küldéséért.
- HistoryViewController
  - Megvalósítja: UITableViewController osztályt, Implementálja: UITableViewDelegate protokoll.
  - Fő célja: A korábbi mentett útvonalak böngészéséért szolgáló nézet vezérlője.
- MapViewController
  - Megvalósítja: UIViewController osztályt, Implementálja: MKMapViewDelegate protokollt.
  - Fő célja: A térkép nézet vezérlője. Megjeleníti a felhasználó jelenlegi helyzetét és kirajzolja útvonalát, ha éppen rögzít egyet. A felhasználó ebből a nézetből tudja megnyitni az alkalmazás beállításait is.
- Route
  - Fő célja: Ez a fájl két osztályt is tartalmaz: Route és RouteLocations. Ahogy az már fentebb is látható volt, a Route belsejében több RouteLocation kerül eltárolásra, valamint néhány extra információ az úttal kapcsolatban, mint például a felhasználó által csatolt kép.
  - Mindketten az RLMObject-ből származnak, mivel csak az RLMObjects-ek menthetőek a Realm adatbázisába.
- RouteStore
  - Fő célja: Singleton osztály, az útvonalak mentéséért, betöltéséért és menedzseléséért felel.
- RouteViewCell
  - Megvalósítja: UITableViewCell osztályt.
  - Fő célja: A mentett útvonalak megtekintésekor biztosít a HistoryView számára egyéni táblázatcellákat.
- SaveAssistant
  - Fő célja: a csatolt képek mentéséért és törléséért felel. Egy képelemző algoritmus is van benne, mellyel csökkenthető a tárolt képek mérete.
- SaveRouteViewController
  - Megvalósítja: UIViewController osztályt, Implementálja: UITextViewDelegate, UIImagePickerControllerDelegate, UINavigationControllerDelegate protokollokat.
  - Fő célja: A mentési képernyő vezérlője. Megjelenítheti a felhasználó galériáját és képet is készíthet benne.
- StatisticsPageViewController
  - Megvalósítja: UIViewController osztályt.
  - Fő célja: Egy diagram megjelenítéséért felelős vezérlőosztály.

- StatisticsRootViewController
  – Megvalósítja: UIViewController osztályt, Implementálja: UIPageViewControllerDataSource protokollt.
  – Fő célja: statisztikák közti lapozás ennek köszönhetően működik.
- StatisticsViewController
  – Megvalósítja: UIViewController osztályt
  – Fő célja: A StatisticsRootViewController nézetét jeleníti meg.
- UserLocation
  – Megvalósítja: NSObject osztályt, Implementálja: CLLocationManagerDelegate protokollt.
  – Fő célja: Singleton osztály, minden osztályból elérhetővé teszi a felhasználó aktuális helyzetét.

Az alkalmazás működéséhez szükséges egyéb fájlok:

- Main.storyboard
  – Fő célja: Az alkalmazás működéséhez szükséges összes nézetet tartalmazza.
- Images.xassets
  – Fő célja: Az alkalmazás működéséhez szükséges összes képfájlt tartalmazza.
- Settings.bundle
  – Fő célja: Beépíti az alkalmazást a gyári Beállítások alkalmazásba.
- Info.plist
  – Fő célja: Alapvető információkat tartalmaz az alkalmazásról, amiket az operációs rendszernek tudnia kell, mielőtt elindítja azt.
- LaunchScreen.xib
  – Fő célja: Ez a nézet jelenik meg a felhasználó előtt, amíg az alkalmazás betölt.

Természetesen más fájlok is szükségesek, mint például az integrált keretrendszerek és beágyazott projektek, de ezek azok a fájlok, melyekkel TDK munkám során foglalkoztam.

8.2 A Mac-es alkalmazás által használt osztályok és fájlok

- AppDelegate
  – Megvalósítja: NSObject osztályt, Implementálja: NSApplicationDelegate protokollt.
  – Fő célja: Az alkalmazás betöltése.
- CBWindowController
  – Megvalósítja: NSWindowController osztályt
  – Fő célja: Az alkalmazás ablakának tulajdonságainak beállítása.
- CBSetupViewController
  – Megvalósítja: NSViewController osztályt
  – Fő célja: A beállítások képernyő vezérlője.
- GPXHelper
  – Fő célja: Singleton osztály, tárolja az utoljára feltöltött GPX fájlt, elvégzi rajta a feldolgozást, majd átalakítja útvonallá.
- CBMapViewController
  – Megvalósítja: NSViewController osztályt, Implementálja: MKMapViewDelegate protokollt.
  – Fő célja: A térkép, az útvonalak és a gyakori helyek megjelenítése a nézeten.
- MeetingPoint
  – Implementálja: Equatable protokollt.
  – Fő célja: A gyakori helyek tárolására alkalmas osztály.
- Route
  – Fő célja: Ez a fájl két osztályt is tartalmaz: Route és RouteLocations. Ahogy az már fentebb is látható volt, a Route belsejében több RouteLocation kerül eltárolásra, valamint néhány extra információ az úttal kapcsolatban, mint például a felhasználó által csatolt kép.
  – Mindketten az RLMObject-ből származnak, mivel csak az RLMObjects-ek menthetőek a Realm adatbázisába.
- RouteStore
  – Fő célja: Singleton osztály, az útvonalak mentéséért, betöltéséért és menedzseléséért felel. A Mac-es kliensben az összes koordináta lekérésére is alkalmas (a második algoritmus miatt).
- GPX
  – Megvalósítja: NSObject, Implementálja: NSXMLParserDelegate protokollt.
  – Fő célja: A kapott GPX fájl parse-olása, átalakítása koordinátákká.

Az alkalmazás működéséhez szükséges további fájlok:

- Main.storyboard

  – Fő célja: Tárolja az összes nézetet és azt, hogy melyik nézethez melyik nézetvezérlő osztály tartozik.

- Assets.xcassets

  – Fő célja: Az alkalmazás által használt képek és ikonok tárolója.

- Info.plist

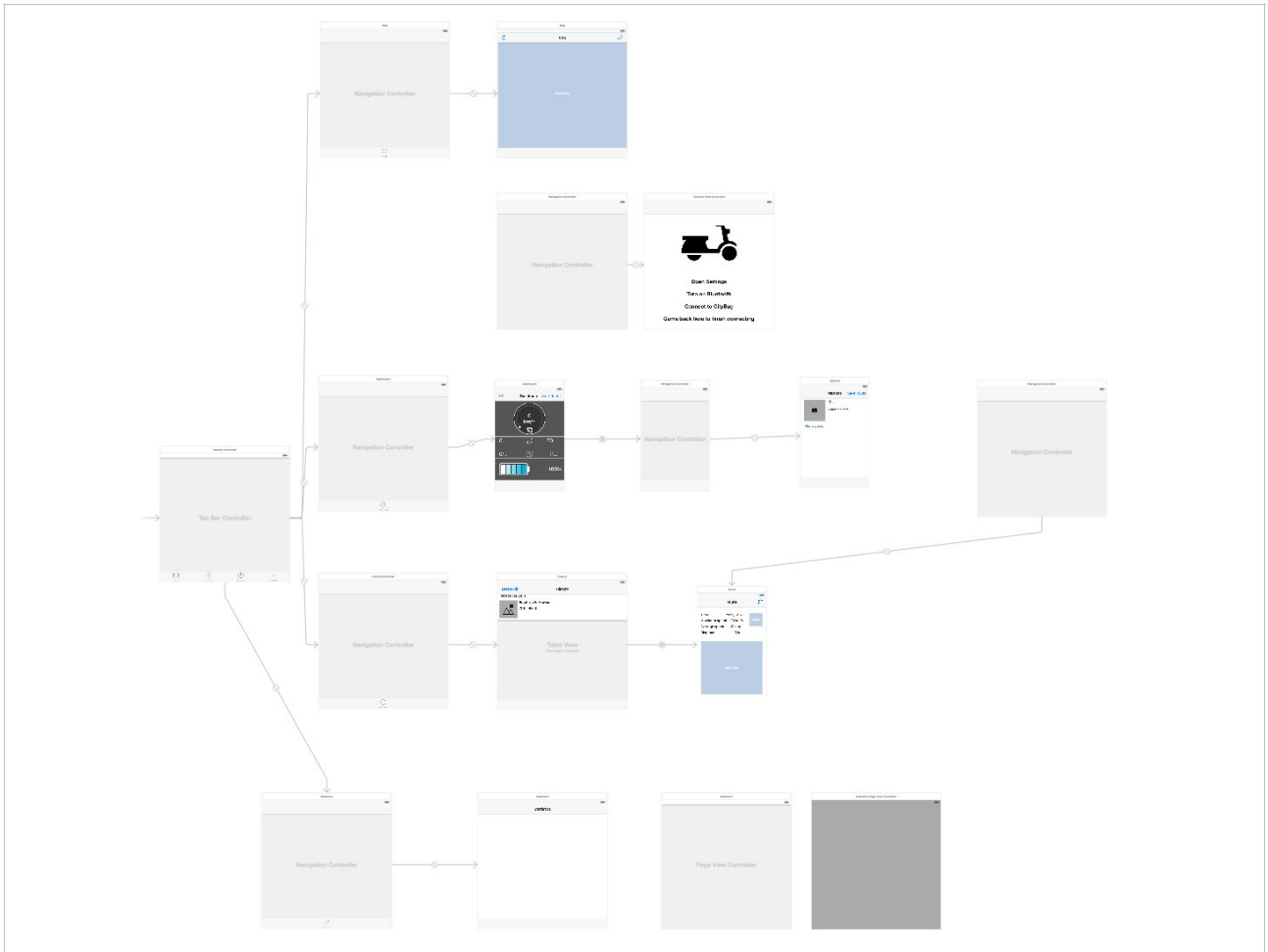
  – Fő célja: Alapvető információkat tartalmaz az alkalmazásról, amiket az operációs rendszernek tudnia kell, mielőtt elindítja azt.

Természetesen más fájlok is szükségesek, mint például az integrált keretrendszerek, de ezek azok a fájlok, TDK munkám során foglalkoztam.

9. iOS alkalmazás felhasználói felületének kialakítása

Ahogy azt már fentebb is említettem, a Mac-es alkalmazásnak nem kell szépnek lennie a rendszer szempontjából, de iOS-en ez alapkövetelmény. A platform felhasználói hozzá vannak szokva a szép animációkhoz és az egyértelmű kezelhetőséghez.

Csak a felhasználói felület kialakítása és tesztelése egy egyhetes feladat volt számomra. A felhasznált alkalmazásoknál is írtam, hogy alapvetően 3 alkalmazást használtam a felhasználói felület kialakítására: az Affinity Photo vektoros képszerkesztőt, ami biztosította számomra azt, hogy a készített grafikák jól mutassanak minden felbontáson, az iAd Producert, hogy az alkalmazás logikáját felépítsem és az Xcode-ba épített Interface Buildert (21. kép), mellyel a felület működését és kinézetét összekapcsoltam az alkalmazás logikájával.

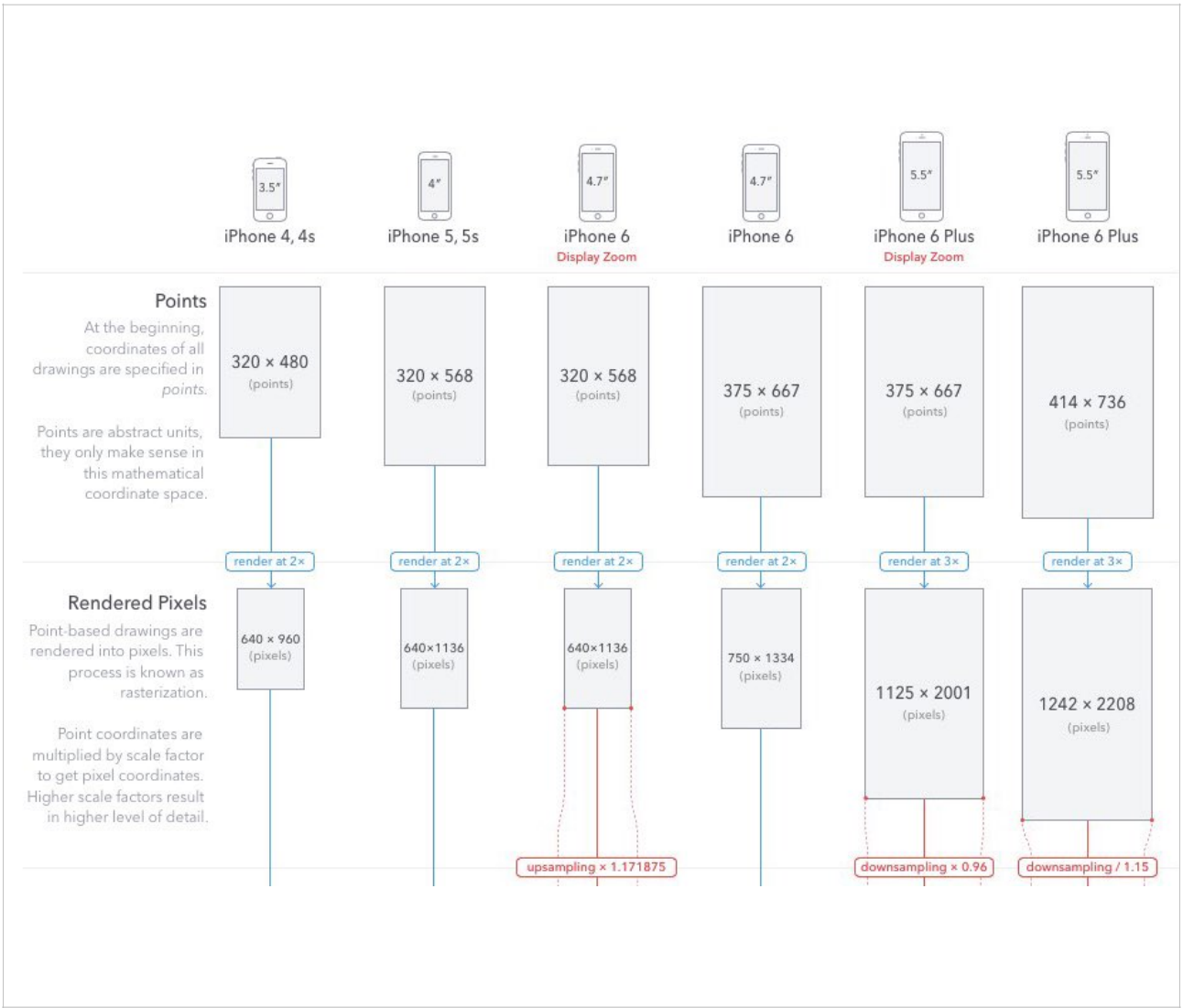


21. kép: A kialakított felhasználói felület gráfja

iOS-re közel sem annyira egyszerű már a felhasználói felületek készítése, mint 2009-ben. A végső alkalmazás 17 különböző képernyővel rendelkezik, ezek mindegyike működik az összes iOS 8-at és 9-et futtató iPhone-on. Ez azt jelenti, hogy működik az iPhone 4S-en, melynek 640 x 960 pixeles a kijelzője, az iPhone 5/s/c-n, melynek 640 x 1136 pixeles a

kijelzője, az iPhone 6/s-en, melynek 750 x 1334 pixeles a kijelzője és az iPhone 6/s Plus-on, melynek 1080p a kijelzője.

Egy olyan alkalmazás készítése, mely ezen kijelzőméretek mindegyikén jól mutat egy igazi kihívás, mert nem csak hogy eltér a kijelzők felbontása, azok pixelsűrűsége sem egyezik. Elég egy pillantást vetni az iPhone 6/s Plus kijelzőjére, mely @3x retina felbontás mellett egy 1242 x 2208 pixeles képet zsugorít le 1080p-re.



22. kép

A többi iPhone mind @2x retina kijelzőt használ, ami azt jelenti, hogy egy logikai képpont kirajzolásához 4 pixelt használ fel. Az iPhone 6/s Plus esetén egy képpontot 9 pixelből rajzolódik ki, majd egy 10%-os zsugorítás után jelenik meg a kijelzőn (22. kép). Így például az iPhone 6/s Plus esetén kerülni kell a vékony vonalakat, mert azok torzan jelenhetnek meg a kijelzőn.

10. Folyamatos működés garantálása iOS-en

Sok szempontból bonyolultabb egy alkalmazás készítése okostelefonra, mint számítógépre, hiszen rengeteg olyan faktorral is számolni kell, amik nem jelentkeznek egy asztali számítógép esetén.

Számítógépen hajlandóak a felhasználók 1 másodpercet várni, amíg betölt valami, de az okostelefon tulajdonosok türelmetlenek. Épp ezért kritikus volt, hogy a számításigényes folyamatok háttérfolyamatként működjenek az iOS alkalmazásban.

De nem ez az egyetlen indok, amiért fontos szálakban gondolkodni iOS-en. Míg számítógépünkön a legtöbb esetben az elindított alkalmazások folyamatosan futnak, mobil operációs rendszereken előfordulhat, hogy a felhasználó kap egy SMS-t, átvált egy másik alkalmazásba, esetleg kap egy telefonhívást, stb., ami miatt nem marad meg az útvonalrögzítő alkalmazás előtérben. Szerencsére egy háttérfolyamata az alkalmazásnak ekkor is megmaradhat, ami folytathatja az útvonal rögzítését vagy a diagnosztikai adatok küldését.

Az Apple által biztosított Grand Central Dispatch motor egyszerűsíti a szálak létrehozását és megszüntetését, valamint az objektumokra történő zárolást is elvégzi. Ez nagyban megkönnyíti a szálképes alkalmazás kialakítását.

Azért, hogy az alkalmazás mindig folyamatosan működjön, háttérszálon fut a pozíció meghatározása, az útvonal elmentése és a diagnosztikai adatok elküldése is. A pozíció frissítő szál értesíti a térképrajzoló osztályt, amennyiben változás történt a felhasználó helyzetében, így az alkalmazás mindig szinkronban van.

11. Egy szebb jövő ígérete

Ahogy látható, elkészítettem az iOS alkalmazást, a szerveroldali kommunikációt és egy asztali klienst, melyen keresztül az okosroller gyártója megvizsgálhatja, hogy hova érdemes töltőállomást helyeznie. Ez persze csak egy mérnökhallgató ötlete, de én őszintén hiszek abban, hogy az okosroller képes lehet átalakítani azt a világot, amit mi ismerünk.

De ha mégsem képes erre, akkor is érdemes megnézni, hogy jelenlegi munkámnak mely része használható fel egyéb célokra. Az elektromos autók most kezdenek csak elterjedni, egyik legnépszerűbb a kategóriában a Tesla.

A Tesla autóit (23. kép) a Tesla saját töltőállomásain és otthon is lehet tölteni, hasonlóan az általam felvázolt okosrollerhez. Felmerülhet hát, hogy az általam készített algoritmusokon milyen változásokat kell eszközölni ahhoz, hogy jól működhessen egy Tesla hálózat kiépítésén is.

23. kép: A Tesla Model S elektromos autó

Míg a rollert az emberek többsége csak rövidtávra használja, addig egy elektromos autóval akár az ország másik felére is átutazhat. Épp ezért egyáltalán nem mindegy, hogy a mögöttes algoritmus fel van-e készítve erre.

Jelenlegi algoritmusomnál éltem azzal a korlátozással, hogy egy útvonal csak egy városba számít bele. Azt, hogy egy adott útvonal melyik városba számít bele, úgy döntöttem el, hogy az útvonal utolsó koordinátája hol található. Azért döntöttem így, mert az útvonalak rögzítésének megkezdésekor még nem feltétlenül pontos a navigáció, de az útvonal végén már minden bizonnyal kellően jó részletességgel megállapítható a roller és a felhasználó helyzete.

Értelemszerűen egy országon átívelő utazásnál nem valószerű, hogy egy utazás csak egy város gyakori helyeinél számítson be. A második algoritmusnál eltekintettem attól, hogy a koordináták útvonalakból érkeznek, így az módosítás nélkül alkalmazható lenne egy Tesla töltőállomás kialakításánál is, bár minden bizonnyal szükség lenne pontosításokra, hogy a nagy forgalmú útvonalaknál figyelembe vegye azt, hogy autópályán például a legtöbben nem fognak csak 2-300 kilométerenként megállni tölteni és pihenni.

12. Összefoglalás

Feladatom elvégzése közben rengeteget tanultam arról, hogy miként működik az iOS és OS X, megismertem a Swift programozási rendszert és az Xcode fejlesztői környezetet is. Mindeközben alkottam egy rendszert, ami képes lehet megreformálni azt, ahogy mindennapjainkban közlekedünk.

Már ezért a kalandért is megérte végigcsinálnom ezt a projektet.

13. Források

- iPhone 6 Plus: mi a fene történik a kijelzőn? (http://beszeljukmac.com/index.php/weblog/comments/iphone_6_plus_mi_a_fene_toertenik_a_kijelzn/) 2015.10.25-én megtekintve
- Bikemag (http://www.bikemap.net/hu/) 2015.10.24-ei adatbázisa
- Facility location problem (https://en.wikipedia.org/wiki/Facility_location_problem) 2015.09.16-án megtekintve
- iOS verziók eloszlása (https://developer.apple.com/support/app-store/) 2015.10.19-ei megtekintés alapján
- CityBug (https://www.citybug.com/) 2015.10.01-ei megtekintés alapján
- Az Apple fejlesztői oldala (https://developer.apple.com/library/ios/navigation/) 2015 augusztustól október 26-ig