EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
TÉRKÉPTUDOMÁNYI ÉS GEOINFORMATIKAI TANSZÉK

Épületen belüli navigációs segéd

DIPLOMAMUNKA

Készítette:
Eszényi Krisztián
térképész mesterszakos hallgató

Témavezető:
Dr. Gede Mátyás
adjunktus

Budapest, 2018

Tartalomjegyzék

1. Bevezetés

2012 szeptemberében kezdtem meg egyetemi tanulmányaimat az ELTE Lágymányosi Campusán. A regisztrációs héten jártam először az egyetem épületében: ekkor volt a beiratkozás, majd mentoraink segítségével a közös tárgyfelvétel. Az elkészült órarendemet nagy kíváncsisággal nézegettem, ismerkedtem az új tantárgyamimmal, valamint ezek időpontjaival és helyszíneivel. A tárgyak nevét és időpontját hamar megjegyeztem, azonban a teremszámok kissé aggodalommal töltöttek el: nem tudtam, hogyan fogom időben megtalálni a termeket két ekkora épületben. Az első héten ezért minden reggel kicsit korábban indultam el otthonról, hogy még időben odataláljak az első órám helyszínére. Azóta eltelt jó néhány év, és közben alaposan megismertem az egyetem területét. Azonban manapság is elég gyakran találkozom a folyosókon tanácstalanul bolyongó hallgatókkal, akik a következő órájuk helyszínét keresik. Számukra lenne hasznos egy olyan beltéri útvonaltervező alkalmazás, mely segítene nekik eljutni egyik pontból a másikba.

Egy útvonaltervezőhöz mindenekelőtt szükséges egy olyan adatbázis, mely tartalmazza az összes olyan helyet, ahová az emberek el szeretnének jutni, valamint az ezeket a helyszíneket összekötő útvonal-hálózatot. A felhasználó pontos helyzetének megállapítása is kulcsfontosságú, hiszen a tervezéshez a kiindulási pont ismerete is szükséges. Mivel a rendszer egy épületen belül fog működni, a műholdas navigációs rendszerek, mint például a GPS vagy a GLONASS, alkalmatlanok a feladatra. Léteznek beltéri helymeghatározást lehetővé tévő, általában wifi, vagy Bluetooth alapú technológiák is, ám ezek kiépítése költséges lenne. Szerencsére az egyetem területén minden ajtónak van egy egyedi azonosítója, így a felhasználó helyzetének megállapításához elegendő, ha megadja a hozzá leközelebb található ajtó számát, amely ezután a kiindulási pontként fog szolgálni.

Diplomamunkám célja, hogy létrehozzak egy, az ajtók azonosítószámain alapuló útvonaltervező rendszert az ELTE Lágymányosi Campus Északi Tömbjére.

2. Az épület felmérése

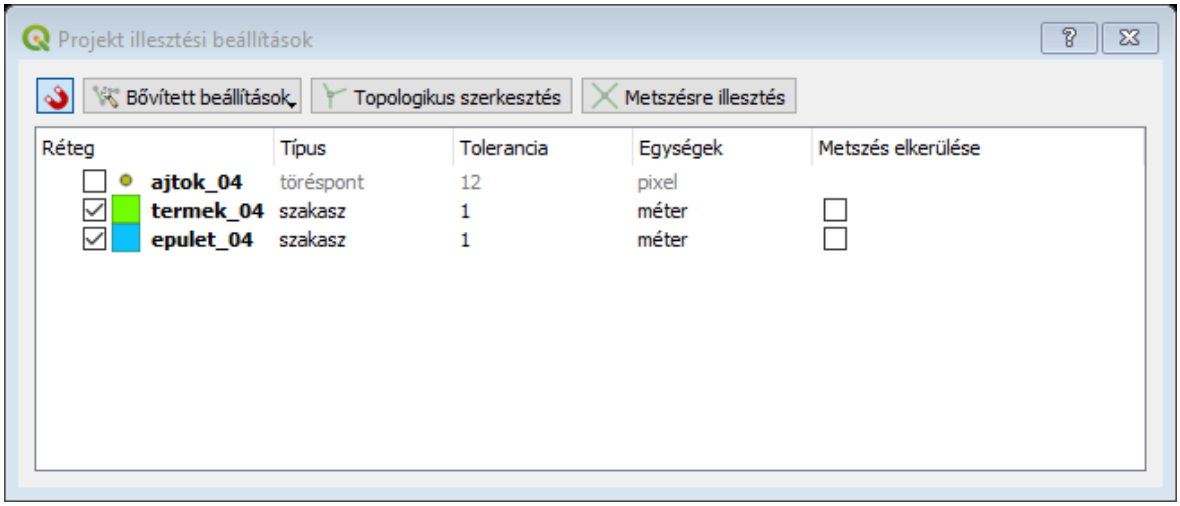
2.1. A felmérés előkészítése

Az útvonaltervezőhöz mindenekelőtt szükség volt egy olyan adatbázisra, mely tartalmazza az egyetem épületét, a termeket, valamint az összes olyan pontot az egyetem területén, amelyből kiindulva és amelyhez útvonalat lehet tervezni. Szerencsére a Térképtudományi és Geoinformatikai Tanszék két doktorandusza, Barancsuk Ádám és Szigeti Csaba már korábban készítettek egy vektoros adatbázist a lágymányosi kampuszról, melyet a rendelkezésemre bocsátottak. Az adatbázis szintenként külön shapefájlokban tartalmazta az egyetem alaprajzát, így az épület és a termek megrajzolásával nem kellett foglalkoznom. Minden egyes szinthez két EOV vetületű poligon típusú shapefájl tartozott: egy *epulet.shp* és egy *termek.shp.* A shapefile egyik fontos tulajdonsága, hogy valójában nem egyetlen fájlból áll, hanem legalább háromból, esetünkben pedig hatból, melynek ugyanaz a neve, csak más a kiterjesztése (dbf - az adatbázis, shp - a geometria, shx – a shape index, cpg - a karakterkódolás, prj - a vetület, qpj - a QGIS saját vetület fájlja EPSG kóddal kiegészítve). A fájlokat a QGIS nevű, nyílt forráskódú térinformatikai szoftver 3.0-ás verziójával nyitottam meg, és szintenként külön QGIS projektfájlokat hoztam létre. Ezeket a projektfájlokat használtam az északi tömb felmérésekor.

A következő lépés egy új, üres adatbázis megtervezése és kialakítása volt, amelyet csak a későbbiekben, az északi tömb bejárásakor töltöttem fel adatokkal. Mivel a készülő alkalmazás pontok közötti útvonaltervezésre lesz alkalmas, ezért a QGIS-ben egy pont típusú shapefájlt hoztam létre a *Réteg/ Réteg létrehozás/ Új shape fájl réteg* menüpontban. A -1. emelet adatbázisával kezdtem, így a fájl neve *ajtok_-1* lett, a karakterkódolást *UTF-8*-ra, a vetületet pedig *EPSG:23700 – HD72/EOV*-ra állítottam. Ezek után meg kellett adni, hogy milyen mezők szerepeljenek az attribútum táblában. A rendszert úgy terveztem, hogy minden ponthoz maximálisan három attribútum tartozhat: a pont típusa (*type*), azonosítója (*id*) és neve (*name*). Mindhárom mező típusának *Szöveges adat*ot állítottam be, és végül elmentettem. Mivel az északi tömb kilenc szintjére készült az útvonaltervező (-1.-től a 7.-ig), az üres shapefájlt még nyolcszor lemásoltam és a fájlnévben lecseréltem a szintazonosítót, így már minden szinthez tartozott egy *ajtok* shapefájl is.

Az *epulet, termek* és *ajtok* fájlokat szintenként külön mappákba rendeztem, majd ezeket a QGIS-ben megnyitottam. A projektbe betöltött három shapefájl megjelenési sorrendjét beállítottam: legalulra az épület, rá a termek, és legfölülre az ajtók kerültek. A *Projekt/ Illesztési beállítások* pontot választva egy ablak ugrott fel, ahol az úgynevezett *snappinget*

lehetett megadni. Ez azt jelenti, hogy digitalizáláskor a kurzor, ha egy megadott távolságon belül van az egyik réteg egy objektumához képest, akkor automatikusan az objektum egyik töréspontjához, vagy két töréspont közti szakaszhoz ugrik. Az aktuális projekt esetében a következőket adtam meg: a termek és az épület réteg, csak szakaszokhoz, 1 méteres tolerancia. *(1. ábra)* Ez a beállítás azért volt fontos, mert a felmérés közben az ajtópontok elhelyezésekor a kurzor a terem vagy az épület falához ugrott, így az ajtók pontosan a fal vonalában jöttek létre.



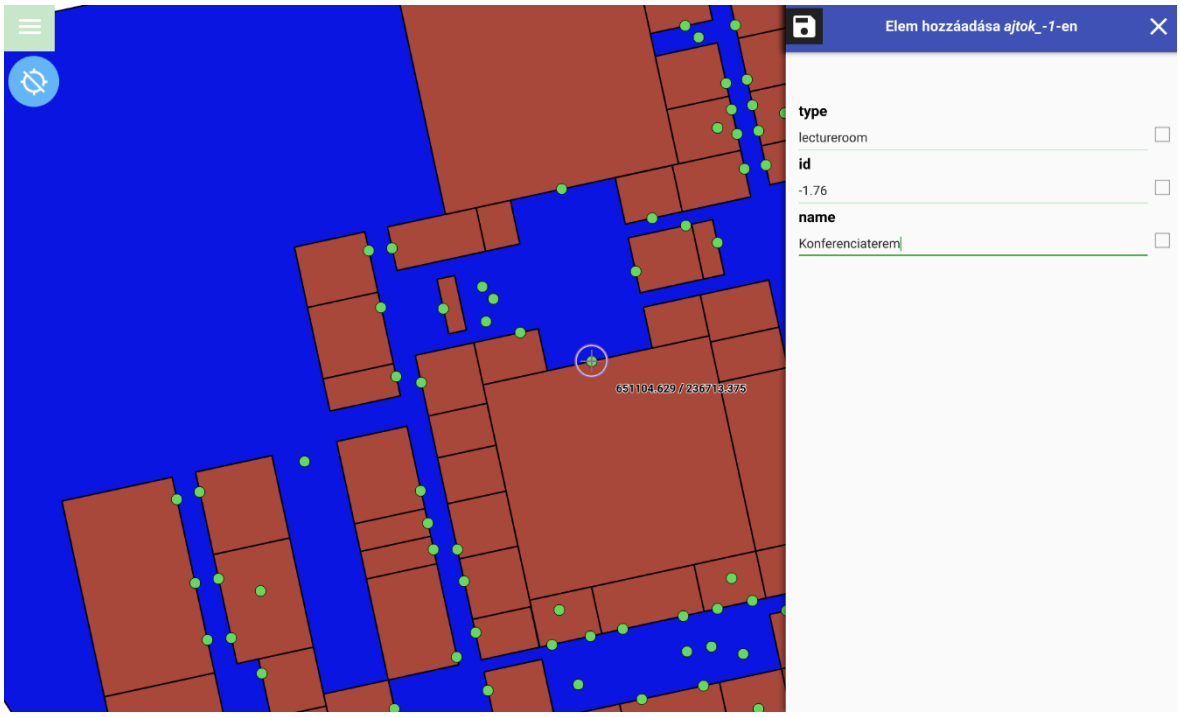
1. ábra
A 4. emelet projektjének illesztési beállításai

A *Mentés* gombbal szintenként *qgs* kiterjesztésű projektfájlokat hoztam létre. A fájlnevek ez esetben is mindig tartalmazták a szintazonosítót. Végül az összes létrehozott fájlt és mappát átmásoltam a tabletemre.

2.2. A felmérés menete

Az északi tömb felmérését egy Samsung Galaxy Tab A 2016 típusú tablettel végeztem. Az eszközön Android fut, így használhattam a QGIS Androidra optimalizált verzióját, a QField for QGIS-t. A programmal csak *qgs* kiterjesztésű fájlokat lehet megnyitni, és a projekthez tartozó shape állományokat böngészni, illetve szerkeszteni. A felmérést a -1. emeleten kezdtem, így a *-1.qgs* projektet nyitottam meg. A szerkesztéshez kijelöltem az *ajtok_-1* réteget, majd a ⚙ / *Üzemmód* / *Digitalizálás* pontot választottam ki. Ekkor megjelent középen a kurzor, valamint a képernyő jobb alsó sarkában egy sárga, kör alakú, ceruzát ábrázoló gomb, amivel a pont rögzítését lehetett megkezdeni.

Mivel a felmérés zárt térben történt, a tablet beépített GPS vevőjét használni nem lehetett, ezért a kurzort kézzel kellett beállítani a rögzíteni kívánt pont helyzetére. Ekkor megnyomtam a sárga gombot, és felugrott egy újabb ablak, amelynek segítségével a ponthoz tartozó három attribútumot lehetett megadni: típus, azonosító és név. *(2. ábra)*



2. ábra
A Konferenciaterem ajtajának hozzáadása az adatbázishoz a QFieldben

A név (*name*) mezőbe került a pont neve, például: Eötvös terem, Büfé, Bankautomata, WC. Az *id* a pont egyedi azonosítóját tartalmazta (teremszám, lépcsőház- és liftazonosító) A típus, azaz a *type* mező kötelezően megadandó volt, ugyanis ez alapján történik az egyes pontok kategorizálása. A *type* mező lehetséges értékeit a következő táblázat tartalmazza:

	Típus	**Magyarázat**
1.	classroom	tanterem
2.	lectureroom	nagy előadóterem
3.	computerlab	számítógépterem
4.	lab	labor
5.	office	iroda
6.	library	könyvtár

7.	other	egyéb helyiség
8.	poi	érdekes pont
9.	stairs	lépcső
10.	elevator	lift
11.	noentry	lezárt folyosó bejárata
12.	exit	kijárat
13.	2exit	kijárat a 2. emeleti teraszra

Az 1.-7. sorszámú pont-típusok mindig valamilyen termet jelölnek, ezen belül az *other* kategóriába kerültek a WC-k, teakonyhák, raktárak és egyéb technikai helyiségek. A *poi*-hoz tartoznak a bank-, étel- és italautomaták, a büfék, valamint az ülőhelyek. A *stairs* és az *elevator*, azaz a lépcső és a lift két fontos elem, ugyanis akárcsak a valóságban, az adatbázisban is ezek biztosítják a kapcsolatot a szintek között. Ezt úgy tudják megtenni, hogy az egyes lépcsőknek és lifteknek minden szinten ugyanaz az azonosítójuk. Így például az E1 ugyanazt a liftet, az L1 pedig ugyanazt a lépcsőt jelenti minden szinten. A *noentry* típusú pont egy olyan folyosószakasz kezdetét vagy végét jelöli, amelyen az átjárás meg van tiltva. Emiatt az útvonaltervező ha érzékeli, hogy a megtervezett út ilyen pontokon vezet át, inkább keres helyette egy másik útvonalat. Ez alól kivétel, ha a célpont egy ilyen lezárt folyosón található. Ekkor természetesen engedélyezett az áthaladás a ponton. Az *exit* az épület kijáratait jelöli, a *2exit* pedig a második emeleti teraszajtókat.

Az egyetem bejárása közben kénytelen voltam kialakítani egy 14. féle típus-kategóriát, a hibát. Kiderült ugyanis, hogy a *termek* shapefájlokban számos hiba található: csak az első három szinten (-1, 0, 1) pontosan 50 esetben volt valami pontatlan az alaprajzon, ami a későbbiekben helyesbítésre szorult. Ezt felmérés közben úgy rögzítettem, hogy a hibás területen egy pontot helyeztem el, a *type* mező értéke *hiba* lett, a *name* mezőbe pedig a leírás került. A hiba-pontok azonban nem sokáig maradhattak az adatbázisban, ugyanis később, a felmérési eredmények feldolgozása közben az alaprajzot is javítottam, így a hibajelölések is törölhetővé váltak. A legtöbb hiba abból fakadt, hogy sok termet időközben kettéválasztottak vagy éppen egybenyitottak, de több esetben komplett lépcsőházak vagy egymás mellett lévő helyiségek hiányoztak az adatbázisból. Előfordult, hogy hibás volt a geometria: átfedtek vagy nem értek össze a termek poligonjai, egy-egy objektum pedig duplikálva volt.

A felmérést az alaprajzi hibákon kívül több dolog is nehezítette. A leggyakoribb a nehézkes teremazonosítás volt: az oktatási-kutatási célokra használatos termek és irodák

megkülönböztetése meglehetősen problémás volt amiatt, hogy az ajtók legtöbbször zárva voltak. A terem funkciójáról pedig csak a folyosón az ajtón lévő kiírásról, vagy az ajtó feletti kis ablakon betekintve lehetett némi információhoz jutni, és ez alapján besorolni a tanterem, számítógépterem, labor, iroda, vagy az egyéb helyiség kategóriába. Az ajtókon, vagy az ajtók mellett lévő táblákon olvasható szövegekkel is körültekintően kellett bánni, ugyanis sokszor elavult, már nem aktuális információkat tartalmaztak a terem nevéről vagy típusáról. A kémia épületben ráadásul több olyan tábla is található, amelyen átragasztották a betűket. Az eredmény néhol értelmetlen betűhalmaz lett, máshol olyan új, értelmes szavak keletkeztek, amelyek már-már valódi szakkifejezések is lehetnének. *(3. ábra)*

3. ábra
Egy átragasztott tábla: abrosz és házi trolifent

A kémia és fizika tanszékeken még egy további nehézség akadt. Ezeken a tanszékeken elég gyakori, hogy a tanárok és doktoranduszok irodái egyben laboratóriumok is, emiatt viszont egyszerre tartozhatnának a *lab* és az *office* kategóriába is. A felmérés közben úgy döntöttem, hogy az ilyen esetekben következetesen a *lab* kategóriába sorolom az adott helyiségeket. Előfordult olyan eset is, hogy kívülről ténylegesen semmi sem utalt a terem funkciójára. Ekkor az ott dolgozó tanároktól vagy diákoktól szereztem információt.

A pontok elhelyezése a termek esetében nem okozott nagy problémát: a QFieldben látható alaprajzon oda került node, ahol a valóságban is a helyiség ajtaja található. Az

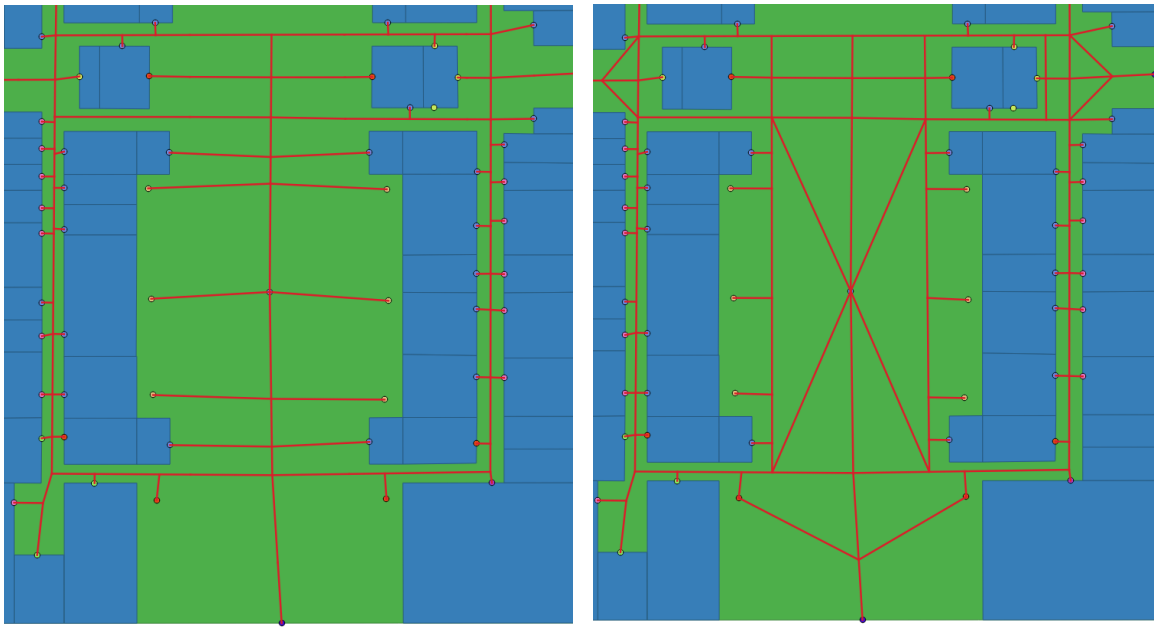
egyetem felmérése közben azonban olyan liftek, kijáratok és érdekes pontok is felvételre kerültek, melyeknél a pontos helyzetmeghatározás nélkülözhető, már-már fölösleges. Ennek oka, hogy ha egy adott területen egyszerre több ugyanolyan objektum található, nem szükséges mindegyiket külön felvenni az adatbázisba. Az ülőhelyek esetében (attribútumai: *type: poi, id: - , name: Ülőhelyek*) nem került felvételre az egyetem összes padja és széke, ez az alkategória egy olyan területet jelöl, ahol pár méteres körön belül néhány ülőhely található. Problémát a Gömbaula és az északi tömb északi kijáratának környezete okozta, ahol viszonylag nagy területen sok ülőhely van, így az nem lehetett megoldás, hogy csak egyetlen pontot helyezek el a Gömbaula közepén, vagy a bejárat mellett. Egyetlen pont esetén az útvonaltervezéskor bizonyos irányokból indulva akár több tíz méteres útvonaltöbblet is adódhatott volna, ezt viszont el kellett kerülni. Megoldásként a Gömbaulában hat, az északi kijárat és a Campus Faloda közötti folyosón négy ülőhely-pontot helyeztem el. A kijáratok, valamint a lépcsőházak bejáratai is az ülőhelyekhez hasonlóak: kis távolságon belül több ajtó, ráadásul ugyanoda vezetnek, így nem kell mindegyiket külön jelölni. Ugyanez igaz a liftekre is, azonban ez esetben még csak nem is mi döntjük el, hogy a három egymás mellett lévő lift közül melyik ajtaja fog kinyílni.

Az egyetem felmérését több részletben végeztem 2018. február és április között. Egy nap alatt kényelmes tempóban egy szintet tudtam felmérni. Miután elkészült az első három szint felmérése, nekiálltam egy próba-útvonaltervező elkészítésének, hogy még időben kiderüljenek az olyan alapvető hibák, amelyek esetleg meghiúsítanák az útvonaltervező működését, és az egyetem felmérését elölről kellene kezdenem. Mivel ilyen jellegű súlyos hibába nem ütköztem, folytattam az egyetem felmérését.

2.3. Az adatok feldolgozása

Miután elkészültem egy szint felmérésével, következett a felmérési adatok feldolgozása. Először is a *termek* shapefile hibáinak kijavításával foglalkoztam. A QGIS-ben az *ajtok* réteget a *tipus* oszlop alapján kategorizáltam és csak a *hiba* típusúakat jelenítettem meg. Az *Elem azonosítás* gomb segítségével kiolvastam az attribútum táblából az adott hiba-pont adatait, melyet a felmérés során a *name* mezőbe vettem fel, majd a *termek* réteget szerkeszthetővé állítottam és módosítottam az alaprajzot. Egy-egy hiba kijavítása után a hiba-pontot töröltem az ajtókat tartalmazó réteg attribútum táblájából. A javítások után az összes pont láthatóságát visszaállítottam és ellenőriztem, hogy minden teremhez tartozik-e legalább egy ajtó-pont, valamint megnéztem, hogy a liftek és lépcsők azonosítói szintenként megegyeznek-e.

A hibajavítást és ellenőrzést követően nekiláttam az útvonalak megrajzolásának: egy olyan vonalhálózatot kellett létrehoznom, mely az *ajtok* réteg összes node-ját összeköti egymással. Az útvonaltervezés ezeken a vonalakon fog megtörténni, és a térkép is ezeket fogja majd megjeleníteni, így hát fontos volt mindent pontosan és életszerűen megrajzolni. Első lépésben létrehoztam egy új shapefájlt, csak most a geometria típusa nem pont, hanem vonal lett. A réteg neve *utvonalak_[szintazonosító]* lett, a vetület maradt EOV. A megrajzolandó vonalakhoz semmilyen attribútumnak nem kellett tartoznia, így új mezőket sem hoztam létre. Azonban minimum egy mezőt kötelező hozzáadni a shapefájlhoz, így a program által automatikusan létrehozott *id* mezőt meghagytam és elmentettem. Az illesztési beállításokat úgy módosítottam, hogy az épület és a termek rétegekét nullára, az az ajtókét és az útvonalakét pedig 1 méteresre állítottam. Bekapcsoltam a *topologikus szerkesztés*t, hogy a későbbiekben a vonalak felrobbantásánál ne legyen probléma. Elindítottam az *utvonalak* réteg szerkesztését, és a folyosók közepére hosszú egyeneseket helyeztem el. Miután a fő közlekedési útvonalakat megrajzoltam, bekötöttem az ajtó-pontokat a folyosók egyenesébe. Ezzel az egy szinten található összes pont összeköttetésbe került, azonban az útvonalhálózatból a már korábban említett életszerűség hiányzott. Kis területeken, például szűk folyosókon nem volt gond, azonban nagy terek esetén, például a Gömbaulában, vagy a liftek és bejáratok előtti területeken az egyetlen megrajzolt útvonal meglehetősen kevésnek bizonyult. Bizonyos pontok ugyanis csak nagy kerülővel lennének így elérhetők. *(4. ábra)*



4. ábra
A Gömbaula és környékének lehetséges útvonalai

A 4. ábrán a Gömbaula és környékének útvonalai láthatók: a bal oldali képen csak egyetlen áthaladó útvonallal, a jobb oldalin már alternatív utakkal kiegészítve. A több vonallal pontosabb tervezés válik lehetővé: több fordulási lehetőség, valamint egyenes és átlós útlevágások segítségével egyszerűbb, rövidebb és ténylegesen életszerűbb útvonaltervek alakulhatnak ki.

Ahhoz, hogy az útvonaltervező algoritmus működni tudjon, nem volt elegendő csupán az útvonalak megrajzolása. Ahol két vagy több vonal találkozott, a metszéspontjukba egy-egy útvonal-csomópontot kellett elhelyezni. A QGIS-ben erre a célra található egy külön algoritmus, mely a *Vektor/ Geometria eszközök/ Csomópont kivonat* menüpontban érhető el. Ez az algoritmus egy vonalas, vagy egy poligon típusú réteg minden elemének kezdő-, törés- és végpontjában node-okat helyez el. Mivel a szerkesztéskor figyeltem arra, hogy két vonal találkozásánál mindig legyen egy töréspont (ezért kellett korábban a *topologikus szerkesztést* is bekapcsolnom), a csomópont kivonat algoritmus alkalmas volt a feladatra. Az *input réteg*nek az aktuális szint útvonalait tartalmazó réteget állítottam be, a létrejövő csomópontokat pedig egy külön shapefájlba mentettem, melynek a neve *utvonalpontok_[szintazonosító]* lett. Az algoritmus azonban túlságosan is jó munkát végzett, emiatt újabb javítások váltak szükségessé. Az útvonal-pontokban pontátfedések keletkeztek, melynek alapvetően két oka volt: egyfelől mivel minden vonalról készült csomópont kivonat, így ha egy pontban több vonal metszette egymást, az összes ott találkozó vonalnak leképződött a node-ja. Másrészt a vonalak egy része az *ajtok* réteg pontjaiban végződött, emiatt az összes ajtó-pontban is keletkezett egy útvonal-pont.

A hibajavítást az ajtókkal közös pontokkal kezdtem, ehhez a *Kiválasztás pozíció alapján* algoritmus segítségét vettem igénybe. A szelektálandó réteg az útvonalpontok, az összehasonlítandó az ajtók réteg lett, az összehasonlítás alapjának pedig az elemek metszését állítottam be. Az algoritmus kiválasztotta a fölösleges elemeket, amelyeket töröltem az adatbázisból. A másik hibajelenség javítására a *v.clean* topológia tisztítót használtam. Az *utvonalpontok* lett a tisztítandó réteg, a *Cleaning tool*nál az *rmdupl* eszközt választottam ki. Az *rmdupl* eltávolítja az azonos koordinátával rendelkező elemeket, az eredmény pedig a *Cleaned* és az *Errors* ideiglenes rétegek lettek. Az eszköz helyes működése esetén a *Cleaned* rétegben csakis olyan objektumok szerepeltek volna, melyekből egy koordinátán csak egy található, az *Errors*ban pedig az esetleges hibák, azonban ez nem így történt. A *Cleaned* layerbe változás nélkül leképződött az összes útvonal-csomópont, ami így használhatatlan volt, cserébe viszont az *Errors*ba nagyrészt topológiailag helyes elemek kerültek. Szintenként változó mennyiségben, de átlagosan 2-3 útvonal-pont teljesen hiányzott, és

körülbelül 10-15 pontban duplán, 1-2 helyen triplán jelent meg ugyanaz a node. A *v.clean* ezekkel a pontokkal nem tudott mit kezdeni, hiába próbáltam az *Errors* rétegen is lefuttatni. Emiatt a QGIS belső *Topológia ellenőrző* modulját hívtam segítségül, amely önállóan javítani nem tud, de legalább a hibákat megmutatja. Ehhez az ellenőrző beállításainál meg kellett adni egy szabályt: az *Errors* layerben *nem lehetnek dupla vonalak*. A topológia ellenőrző panelen ezután a *Minden ellenőrzése* gombra kattintottam, és a térképen pirossal kiemelte a program a hibás node-okat, melyeket egyesével kitöröltem. A hibajavítás befejeződött, úgyhogy már csak szintenként egyesíteni kellett az ajtók és az útvonal-csomópontok rétegeket. Az ajtók rétegét szerkeszthetővé tettem, és bemásoltam az *Errors* tartalmát az attribútum táblába. Ezáltal kialakult a 15. *type*-kategória, a *NULL*, azaz ha egy elem *type* mezőjében nincs semmi, akkor az egy útvonal-csomópont.

Ekkor már csak egyetlen lépés volt hátra: a vonalak felrobbantása. Korábban, az útvonalak megszerkesztésekor bekapcsoltam a *topologikus szerkesztés*t, amely lehetővé tette, hogy ha egy vonalat nem csomópontban csatlakoztattam egy másikhoz, akkor a már létező vonalon is létrejöjjön egy node. A *Vonalak felrobbantása* algoritmus ugyanis csomóponttól csomópontig tartó szegmensekre bontja szét az összetett vonalakat, ez pedig lényeges az útvonaltervező helyes működéséhez. Az algoritmus eredményeként az *utvonalak_felr_[szintazonosító]* shapefájlok jöttek létre. Ezzel szintenként kialakult egy gráf, melynek csúcsait és csomópontjait az *ajtok* layer felmért, valamint a vonalak metszéspontjaiban elhelyezett pontok alkotják, éleit a megrajzolt és felrobbantott vonalak. *(5. ábra)*

5. ábra
A Térképtudományi és Geoinformatikai Tanszék egyik folyosójának útvonal-gráfja

3. Az útvonaltervező adatbázis

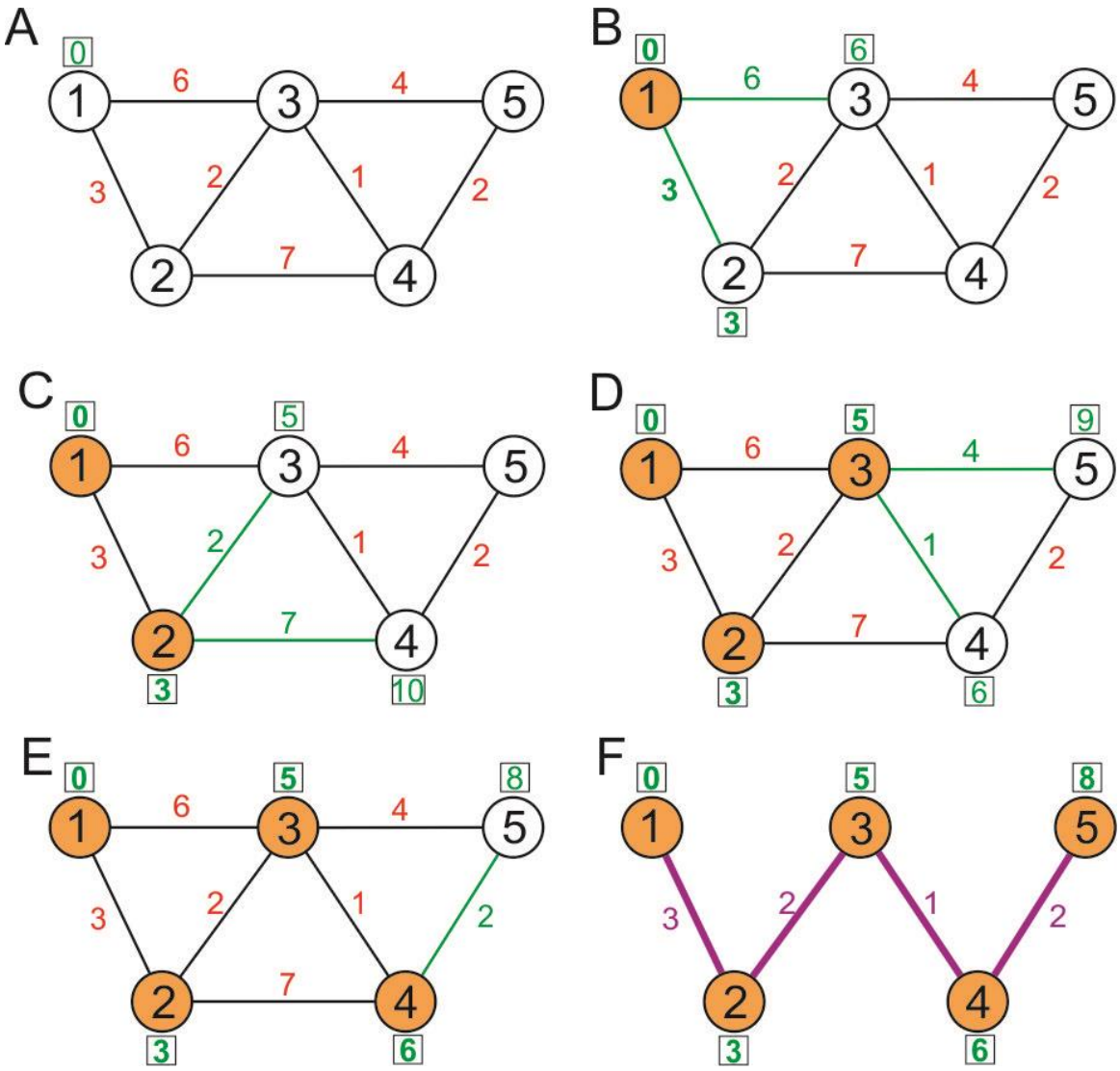
3.1. PostgreSQL, PostGIS, pgRouting

Miután az útvonaltervező alapjául szolgáló adatbázis elkészült, szükség volt egy olyan adatbáziskezelő szerverre, ahová az adatokat feltölthettem. Szerencsére a Térképtudományi és Geoinformatikai Tanszéknek van egy ilyen szervere, amely a wms.elte.hu címen érhető el. A wms-en a PostgreSQL (más néven Postgres) fut, amely egy nyílt forráskódú relációsadatbázis-kezelő alkalmazás. A Postgres sokféle adattípus tárolására alkalmas, még egyszerű geometriára is, azonban földrajzi adatokkal rendelkező objektumok kezelésére önmagában alkalmatlan. Az alkalmazáshoz több kiegészítő csomagot is kifejlesztettek, az egyik ezek közül a Refractions Research által 2000-ben elkészített PostGIS, mely a Postgres geometria típusú adattárolási lehetőségét kihasználva térbeli adatok tárolására és analizálására szolgál. A PostGIS-szel számos térbeli funkció érhető el, például felületek, távolságok, pufferzónák számítása, vagy épp topológiai vizsgálatok, például metszések, átfedések, tartalmazások. Egy dologra azonban még a PostGIS sem képes: az útvonaltervezésre. Ehhez ismételten egy kiegészítőre van szükség, a pgRoutingra, mely útvonaltervező és egyéb hálózatelemzési funkciókkal bővíti az adatbáziskezelőt. A tanszéki szerveren található Postgres mindkét kiegészítővel rendelkezik, magához az adatbázishoz pedig a PhpPgAdmin webes adminisztrációs alkalmazás segítségével lehet hozzáférni, amely a wms.elte.hu/phppgadmin oldalon érhető el kizárólag ELTE-s IP-címről.

3.2. A Dijkstra algoritmus

Az útvonaltervezés lényege, hogy találjunk egy olyan utat, amely egyik pontból a másikba vezet. Azonban mivel az út megtételéhez idő- és energiaráfordítás szükséges, fontos kitétel, hogy a talált útvonal költsége, azaz a csomópontokat összekötő élek költségeinek az összege a lehető legkisebb legyen. Általános esetben az útvonaltervezők a távolság és/vagy az eljutáshoz szükséges idő alapján állapítják meg a legrövidebb utat. Az idő leginkább autós navigációnál játszik fontos szerepet: itt előfordulhat, hogy egy útvonal hosszabb, azonban nagyobb sebességgel lehet rajta közlekedni (például autóutak, autópályák), így mégis gyorsabban lehet eljutni a célponthoz. Az északi tömb esetében nincs szó különböző sebességgel járható útszakaszokról, így itt az egyes útvonalak költsége kizárólag a két pont közötti távolságtól függ. A legrövidebb útvonalak megtalálását különböző algoritmusok teszik lehetővé.

A pgRouting egyik útvonalkereső algoritmusa a Dijkstra-algoritmus. Az algoritmust Edsger Wybe Dijkstra fejlesztette ki 1956-ban, majd három évvel később publikálta. Az útvonaltervező működtetéséhez én is ennek az algoritmusnak a segítségét vettem igénybe. A Dijkstra-algoritmus ugyanis az egyik leghatékonyabb megoldás az útvonaltervezéshez: egy adott csúcspontból kiindulva megkeresi a lehető legrövidebb útvonalat a gráf összes többi pontjába. Mivel az algoritmus előre nem tudja, hogy egyik pontból a másikba mely útvonalakon lehet eljutni, ezért szükséges a teljes gráf bejárása. A Dijkstra-algoritmus működését a *6. ábra* szemlélteti.



6. ábra
A Dijkstra-algoritmus működése

A kezdőcsúcs az 1-es, ott az útvonalköltség nulla (A). Ezt követően az algoritmus kiszámítja a szomszédos csúcsokhoz vezető utak költségét: a 2-es ponthoz 3, a 3-as ponthoz 6 költséggel lehet eljutni (B). Mivel a 2-eshez alacsonyabb költséggel lehetett eljutni, innen folytatjuk a szomszédos csúcsokhoz vezető élek vizsgálatát. A 2-es is szomszédos a 3-assal, ráadásul köztük csak 2 költségű él van, így az összesített költség az 1-es ponttól nézve csak 5 az 1 és 3 közti közvetlen él 6 költsége helyett. Szomszédos továbbá a 4-es pont, ide 7, összesen 10 költséggel lehet oda eljutni (C). Az alacsonyabb költségű 3-as pontból keres tovább az algoritmus: az 5-ösbe 4, összesen 9, a 4-esbe 1, összesen 6, tehát a 2-esből közvetlen él helyett a 3-as ponton át alacsonyabb költségű út vezet (D). A keresés a 4-es csúcsból folytatódik: az 5-ösbe 2, összesen 8 költséggel lehet eljutni, így ez is rövidebb, mint a 3-asból közvetlen él mentén (E). Az 1-esből az 5-ös pontba tehát az 1-2-3-4-5 útvonalon 8 költséggel lehet eljutni (F).

3.3. Az adatbázis feltöltése és testreszabása

A szerveren tehát minden készen állt az adatok fogadására, így következhetett a Postgres adatbázis feltöltése, melyhez továbbra is a QGIS-t vettem igénybe. A programban adatbázis-kezeléshez több modul használható, ilyen például a *Spit* vagy a *DB kezelő* is. A *Spit*et ma már a QGIS modulkezelője nem támogatja, csakis a *DB kezelőt*, ezért a munkát az utóbbival szerettem volna végezni. Először is szükség volt egy új kapcsolat létrehozására a QGIS és a wms.elte.hu-n futó Postgres adatbázis között. A QGIS böngészőablakában a *PostGIS/ Új PostGIS kapcsolat létrehozása* menüpontot választottam ki, majd megadtam az új kapcsolat adatait (név: *PgRouting*, gazda gép: *wms.elte.hu*, adatbázis: *gis*), valamint a felhasználónevet és a jelszót. A kapcsolat létrejött, és a *DB kezelő*vel feltöltöttem minden szint *ajtok* és *utvonalak_felr* shapefájlját. A wms.elte.hu/phppgadmin oldalon beléptem a felhasználónevemmel, és a *gis* adatbázis *routing* sémájában megtaláltam a feltöltött adatokat. A pontokat tartalmazó táblák öt oszlopban tárolják az adatokat: a már korábban is látott *type, id* és *name* mellett megjelent a *gid* és a *the_geom* mező. A *gid* a tábla elsődleges kulcsa, amely egyedileg képes azonosítani a táblában tárolt sorokat. Ebben az oszlopban csakis nem *NULL* és sosem ismétlődő adatok szerepelhetnek, mert ez az elsődleges kulcs feltétele. A létrejött táblák esetében ezek az azonosítók 1-től egyesével növekvő egész számok (*integer*). A *the_geom* mező tartalma egy hexadecimális kód, mely az egyes pontok geometriáját tárolja. *(7. ábra)* Az útvonalak tábláiban csak három mező van: az *id*, amit még a rajzoláskor kellett létrehozni, de a tartalma *NULL*, valamint a *gid* és a *the_geom*.

Műveletek		gid	type	id	name	the_geom
Szerkeszt	Töröl	1	other	0.125	Porta	010100000073784CE8F0DE234102E80FB1F1E50C41
Szerkeszt	Töröl	2	exit	x1	*NULL*	0101000000E3E708A6FBDE2341168D2535D2E50C41
Szerkeszt	Töröl	3	elevator	E1	*NULL*	010100000077115F1E3DE2341FE1F9635BAE50C41
Szerkeszt	Töröl	4	other	0.115	WC (női)	0101000000C83AB6F7E3DE23414802B03A04E60C41
Szerkeszt	Töröl	5	office	0.116	*NULL*	0101000000402C8497E2DE234193349E361DE60C41
Szerkeszt	Töröl	6	other	0.124A	*NULL*	01010000000F9C8728E8DE2341D642188C20E60C41
Szerkeszt	Töröl	7	office	0.124	*NULL*	01010000006BCCAD94E6DE2341CB1DDF743DE60C41
Szerkeszt	Töröl	8	office	0.117	*NULL*	0101000000E9BDE7DCDFDE2341C28464C84EE60C41
Szerkeszt	Töröl	9	other	0.123A	*NULL*	0101000000D6A42E6AE5DE23415CDAEC6352E60C41
Szerkeszt	Töröl	10	office	0.123	*NULL*	0101000000779DCC13E4DE23414C8D9AFF6AE60C41

7. ábra
Az ajtok_00 tábla tartalma (részlet)

Az egyes oszlopok típusait vizsgálva azonban feltűnt a *DB kezelő* egy komoly hibája. A létrehozott shapefájlok geometriája az ajtók esetében pont (point), az útvonalak esetén vonal (linestring) volt, azonban a feltöltést követően a geometriát tartalmazó oszlop már multipoint és multilinestring típusra módosult. Ez azonban az útvonaltervezőhöz nem használható, mert megbonyolítaná a lekérdezéseket. A hibajavítás helyett inkább töröltem a routing séma tartalmát és újrakezdtem a feltöltést egy olyan QGIS-ben, amelynél még elérhető volt a *Spit* modul. Ez a modul szinte pontosan megegyezik a *DB kezelő*vel, ugyanazokat a lépéseket kellett végrehajtanom, mint a korábbi feltöltésnél, csak ez esetben megmaradt a sima pont és vonal geometria. Végül átneveztem a táblákat: az ajtókból *nodes_[szintazonosító]_eszak* lett, az útvonalakból pedig *utvonalak_[szintazonosító]_eszak.*

A következő lépés a szintek tábláinak egyesítése volt, külön az ajtóknak és külön az útvonalaknak. Az egyesített adatokat azonban nem egy-egy újabb táblában, hanem egy-egy nézetben *(view)* tároltam el. A nézetek annyiban különböznek a táblától, hogy nincsenek tárolva az adatbázisban, csakis más táblákra vonatkozó lekérdezésekből jönnek létre. A nodes táblákat összefogó *nodes_eszak* nézet létrehozásához a következő SQL-lekérdezést hajtottam végre:

```
create view nodes_eszak as (select gid,type,id,name,the_geom,0 as level
from nodes_00_eszak union select gid+10000,type,id,name, the_geom,1 as
level from nodes_01_eszak union select gid+80000, type,id,name,the_geom,
-1 as level from nodes_minusz1_eszak union select gid+20000,type,id,
name,the_geom,2 as level from nodes_02_eszak union select gid+30000,
type,id,name,the_geom,3 as level from nodes_03_eszak union select
gid+40000,type,id,name, the_geom,4 as level from nodes_04_eszak union
select gid+50000, type,id,name,the_geom,5 as level from nodes_05_eszak
```

```
union select gid+60000,type,id,name,the_geom,6 as level from
nodes_06_eszak union select gid+70000,type,id,name,the_geom,7 as level
from nodes_07_eszak)
```

Ez a lekérdezés minden nodes táblából kiválasztotta a *type, id, name, the_geom* és *gid* mezőket, ez utóbbit azonban szintenként módosította is. Mivel a *gid* szintenként 1-től számozódik, a kilenc egyesített tábla miatt kilencszer fordulna elő ugyanaz az azonosító. Azonban a *gid*-t továbbra is elsődleges kulcsként szerettem volna használni, így szintenként eltérő mennyiségeket hozzáadtam az azonosítóhoz. A földszinten maradt az eredeti, az első emelethez 10 000-et, a másodikhoz 20 000-et, és ezt így folytatva a hetedik emelet azonosítóihoz 70 000-et, a -1. emelethez pedig 80 000-et adtam hozzá. Ezen kívül létrehoztam egy új mezőt is a nézetben, a *level*t. Ez tartalmazza minden rekordban a szintazonosítót.

Az útvonalaknak is létrehoztam egy külön nézetet *utvonalak_eszak* néven az előzőhöz hasonló lekérdezéssel.

```
create view utvonalak_eszak as (select gid,the_geom,0 as level from
utvonalak_00_eszak union select gid+10000,the_geom,1 as level from
utvonalak_01_eszak union select gid+80000,the_geom,-1 as level from
utvonalak_minusz1_eszak union select gid+20000,the_geom,2 as level from
utvonalak_02_eszak union select gid+30000,the_geom,3 as level from
utvonalak_03_eszak union select gid+40000,the_geom,4 as level from
utvonalak_04_eszak union select gid+50000,the_geom,5 as level from
utvonalak_05_eszak union select gid+60000,the_geom,6 as level from
utvonalak_06_eszak union select gid+70000,the_geom,7 as level from
utvonalak_07_eszak)
```

Ezt követően egy újabb SQL-lekérdezéssel kialakítottam az útvonaltervezés alapjául szolgáló táblát, a *routable_eszak*ot.

```
create table routable_eszak as (select u.gid as id,n1.gid as
source,n2.gid as target, st_length(u.the_geom) as cost from nodes_eszak
n1, nodes_eszak n2, utvonalak_eszak u where
n1.the_geom=st_startpoint(u.the_geom) and n1.level=u.level and
n2.the_geom=st_endpoint(u.the_geom) and n2.level=u.level order by 1)
```

Az eredmény egy négy oszlopból álló tábla lett. Az *id* az elsődleges kulcs, ez az oszlop tartalmazza az útvonal, a *source* a kiindulási pont, a *target* pedig a végpont azonosítóját. A

cost oszlopban láthatók a gráf éleinek a költségei, ezek az értékek valójában az egyes szakaszok méterben megadott hosszának felelnek meg. *(8. ábra)*

id	source	target	cost
1	187	189	2.67190855762963
2	189	188	0.705749660706931
3	188	191	3.97654656810965
4	191	190	0.51251356588399
5	190	192	2.96041223238082
6	192	193	3.27987499384728
7	193	194	3.43713755572804
8	194	195	3.19685138753819
9	195	196	3.2485358481387
10	196	197	4.91161240730092
11	197	198	1.52690297934505
12	198	199	5.52283921652171
13	199	174	3.05881206206338
14	174	200	1.65935779974704
15	200	201	3.12991830109397

8. ábra
A routable_eszak tábla tartalma (részlet)

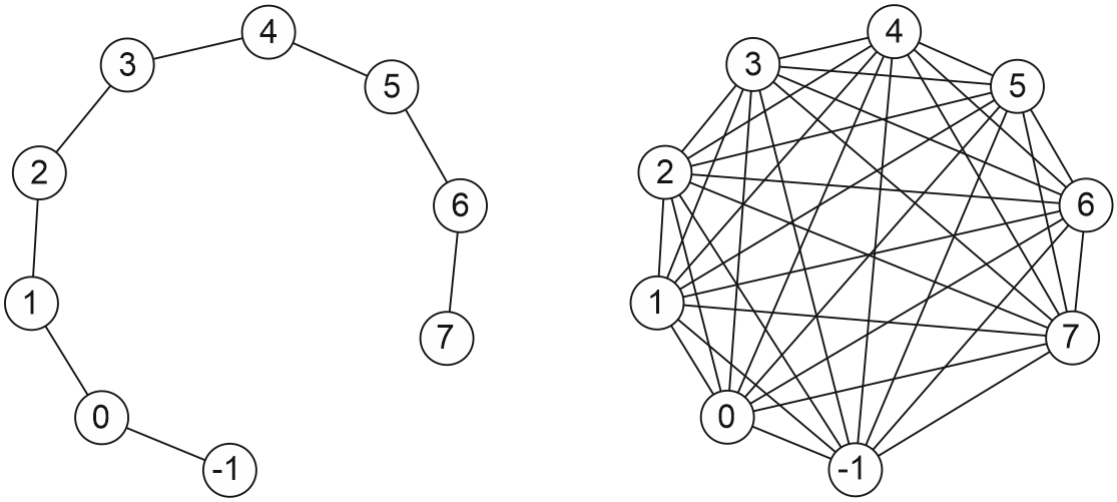
Az útvonaltervezés ezzel lehetővé vált, azonban csakis egy-egy szinten belül. Bár ekkor már minden attribútum egy táblában volt, a szintek között továbbra sem állt fent semmilyen kapcsolat. A következő feladat tehát a szintek összekapcsolása volt a lifteken és lépcsőkön keresztül. A liftek és lépcsők alapvetően más-más típusú kapcsolatot teremtenek két szint között, és ennek a különbségnek az útvonaltervezőben is meg kellett jelennie. A liftek bármelyik szintet képesek viszonylag hamar, ráadásul az ember számára nem megterhelő módon elérni. Az utazással töltött idő is alig különbözik több szint esetén, a legtöbb időt gyakran a liftre várás teszi ki. A lépcsőzéssel töltött idő és a megterhelés ugyanakkor szintről-szintre nő, azonban egy-két emeletnyi különbség esetén még reális alternatívát nyújt: nem olyan megterhelő, ráadásul nagy valószínűséggel még gyorsabb is, mint elmenni egy távolabbi lifthez és azt megvárni. Ezt a különbséget az útvonaltervezőben a lépcső- és lift-node-ok közti élekkel és költségeiknek beállításával lehetett modellezni. Lépcsők esetén a következő SQL-parancsot hajtottam végre:

```
insert into routable_eszak (select (row_number() over ())+100000 as
id,n1.gid as source,n2.gid as target, 10 as cost from nodes_eszak n1,
nodes_eszak n2 where n1.type='stairs' and n1.id=n2.id and
n2.level=n1.level+1)
```

Ezzel az *insert* paranccsal új elemeket helyeztem el a *routable_eszak* táblában. Ez a lekérdezés megkereste a lépcsőket, azaz minden olyan node-ot, amelyeknél a *type* mezőben *stairs* szerepelt, és egy-egy 10 költségű élt szúrt be azok közé, amelyeknek az azonosítójuk megegyezett (*n1.id=n2.id)* és csak egy emelet különbség volt köztük (*n2.level= n1.level+1*). A létrejövő éleknek új azonosítók is kellettek: a 0 és 90 000 közötti *id*-k már foglaltak voltak az útvonalak miatt, így a lépcsők a 100 000-rel kezdődő azonosítókat kapták. Ezáltal a lépcsőkön keresztül összeköttetésbe került az északi tömb összes felmért szintje. Következett a liftek összekapcsolása:

```
insert into routable_eszak (select (row_number() over ())+110000 as
id,n1.gid as source,n2.gid as target, 20 as cost from nodes_eszak n1,
nodes_eszak n2 where n1.type='elevator' and n1.id=n2.id and
n2.level>n1.level)
```

A parancs hasonló az előzőhöz, csak most *elevator* típusú elemeket kerestem, és 110 000-rel kezdődő azonosítójú, 20 költségű élekkel kötöttem össze a lifteket. Az összekötés módja azonban alapjaiban különbözik a lépcsőktől: az azonos *id*-vel rendelkező liftek nemcsak a következő, hanem az összes szinttel össze vannak kötve. *(9. ábra)* Emiatt egy emelet megtétele lépcsőn 10, lifttel 20 költségű, azonban például öt emelet esetén lépcsőn már 10+10+10+10+10=50, míg lifttel továbbra is csak 20.



9. ábra
Az épület szintjeinek összekapcsolása lépcsővel (balra) és lifttel (jobbra)

4. A webes felhasználói felület

4.1. Útvonaltervezés és megjelenítés

Az útvonaltervező interneten történő megjelenítéséhez és kezeléséhez alapvetően három fő, valamint számos kisebb jelentőségű fájl együttes működése szükséges. Először is szükség volt egy szerverre, melyen ezeket a fájlokat tárolhattam: a választás a terkeptar.elte.hu-ra esett, mely a mercator.elte.hu-n futó virtuális gépek egyike. A virtuális géphez hozzáadtunk egy új felhasználót, a *campusrouting*ot, ennek a nyilvános mappájába, a *public_html*-be kerültek az útvonaltervező alkotóelemei.

A három fő fájl az *index.html*, mely maga a html nyelven írt weboldal, a *route_eszaki.php*, amely az adatbázis-kapcsolatért és az útvonaltervezésért felel, a harmadik pedig az *eszaki.js*, amely kétirányú kapcsolatot teremt a html és a php fájl között, valamint megjeleníti a térképet és az útvonaltervet. Az útvonaltervező működése a következő: az *index.html*-ben van egy *panel* azonosítójú *div*, azaz tárolóelem. Ebben található két szövegdoboz (*from, to*), egy checkbox, azaz jelölőnégyzet (*enableRestricted*), és egy gomb (*Tervezés*).

```
<div id="panel">
      Honnan: <input type="text" id="from"/>
      Hova: <input type="text" id="to"/>
      <input type="checkbox" id="enableRestricted">
      <input type="button" value="Tervezés" onclick="route()" />
</div>
```

A szövegdobozokba kell beírni a kiinduló- és a célpont nevét vagy azonosítóját, a jelölőnégyzetben a pipa eltüntetésével vagy elhelyezésével lehet beállítani, hogy a tervezéskor a tiltott, vagy nem ajánlott szakaszokat (például lezárt folyosók, teherliftek) figyelembe vegye-e a program. A *Tervezés* gombra kattintva meghívódik a *route()* függvény az *eszaki.js* fájlban. A *function route()* egy összetett, több részből álló függvény. Elsőként az a feladata, hogy a html-ből megkapott információkat az *id1*, *id2* és *restr* változókba lementse, majd ezeket továbbítsa a *route_eszaki.php*-nak.

```
var id1=document.getElementById('from').value;
var id2=document.getElementById('to').value;
var restr=document.getElementById('enableRestricted').checked?0:1;
```

```
var x=new XMLHttpRequest();
x.open('get','route_eszaki.php?id1='+id1+'&id2='+id2+'&restricted='
+restr,false);
x.send();
```

Ekkor a php fájl lép működésbe és megnyitja a kapcsolatot az útvonaltervező adatbázishoz, majd készít egy SQL lekérdezést a kezdő- és végpontokról.

```
$db = pg_connect("host=wms.elte.hu port=5432 dbname=gis user=routing
password=*********")
    or die("Could not connect");
if (strpos($id1,'[gid:')!==false) {
     $id1=substr($id1,strpos($id1,'[gid:')+5,-1);
    }
else {
    $r=pg_query($db,"select gid from nodes_eszak where id='$id1' or
    name='$id1' limit 1;");
    $l=pg_fetch_assoc($r);
    $id1=$l['gid'];
}
if (strpos($id2,'[gid:')!==false) {
    $id2=substr($id2,strpos($id2,'[gid:')+5,-1);
    }
else {
    $r=pg_query($db,"select gid from nodes_eszak where id=
   '{$_GET['id2']}' or name='{$_GET['id2']}' limit 1;");
    $l=pg_fetch_assoc($r);
    $id2=$l['gid'];
}
```

Ezt követően egy feltételes utasítás miatt kétféle irányban folytatódhat a program. Ha nem engedélyeztük a tiltott útvonalakon történő tervezést, azaz nem volt pipa a jelölőnégyzetben, akkor a program először létrehozza vagy frissíti a *rest_routes* nézetet az adatbázisban a következő paranccsal:

```
create or replace view rest_routes as (select * from routable_eszak
where not exists (select 1 from nodes_eszak where (type='noentry' or id
in ('E3','E4','E5','E7','E8')) and gid in (source,target)));
```

Ebben a nézetben a *routable_eszak* tábla tartalmának legnagyobb része megtalálható, kivéve a *noentry* típusú útvonalpontokat és az *E3, E4, E5, E7, E8* azonosítójú lifteket. Ezután következhet a tényleges útvonaltervezés a *pgr_dijkstra* függvény segítségével. A lekérdezés hozzákapcsolja a függvény eredményéhez a geometriát, és közben egy vetületi transzformációt is végrehajt EOV-ból (EPSG: 23 700) Web Mercatorba (EPSG: 3857), így az eredmény térképen is megjeleníthetővé válik:

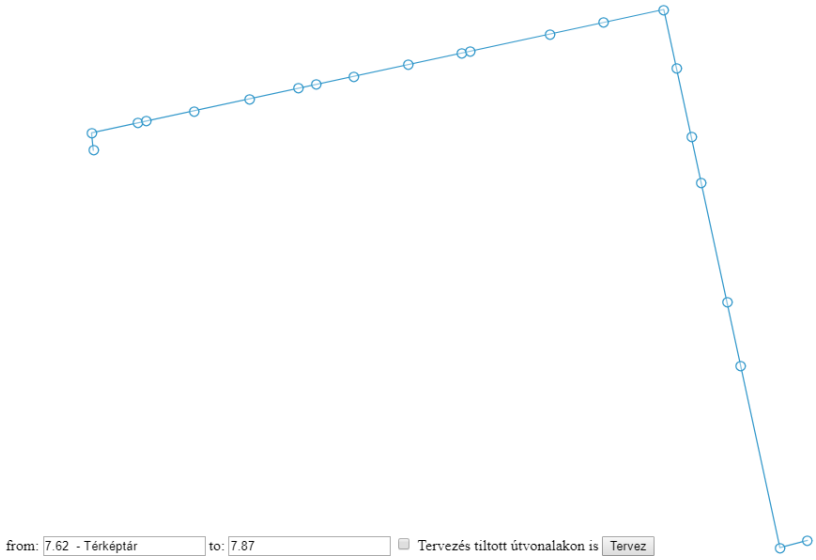
```
select st_x(st_transform(n.the_geom,'init=epsg:23700',3857)) as x,
st_y(st_transform(n.the_geom,'init=epsg:23700',3857)) as y, n.level as
level, n.type, n.id as id, n.name as name from pgr_dijkstra('select *
from rest_routes',$id1,$id2,false,false) plan, nodes_eszak n where
n.gid=plan.id1 order by plan.seq;
```

Abban az esetben, ha a tiltott útvonalak engedélyezve vannak, a feltételes utasítás *else* ága fut le. Ekkor nem kell új nézetet létrehozni, az útvonaltervezés a teljes adatbázis tartalma alapján történhet, emiatt nincs szükség a *rest_routes* nézet létrehozására, csakis az utóbbi lekérdezés megy végbe annyi különbséggel, hogy a *rest_routes* helyett a *routable_eszak* szerepel a parancsban. A tervezés után az eredményt a következő parancs küldi vissza a JavaScript fájlnak, majd zárja a Postgres adatbázissal a kapcsolatot:

```
while($l=pg_fetch_assoc($r)) {
   print $l['x'].";".$l['y'].";".$l['level'].";".$l['type'].";".$l
   ['id'].";".$l['name']."\n";
}
pg_close($db);
```

Az *eszaki.js route()* függvénye folytatódik ezután. A visszaküldött eredményt a `var pts=x.responseText.split('\n');` parancs a *pts* tömbben az új sor jelek (\n) mentén feldarabolva tárolja el pontonként a következő módon: x koordináta, y koordináta, szintazonosító, típus, azonosító, név. Ez alapján a következő *for* ciklus hozzáadja az útvonalat az *rs* vektor réteghez, így az a képernyőn is megjelenik. *(10. ábra)*

```
var path=[];
for (var i=0;i<pts.length-1;i++) {
     var c=pts[i].split(';');
     path.push([parseFloat(c[0]),parseFloat(c[1]),parseFloat(c[2])]);
     var f=new ol.Feature({geometry:new ol.geom.Point([parseFloat
           (c[0]),parseFloat(c[1])])});
     rs.addFeatures([f]);
 }
 rs.addFeatures([new ol.Feature({geometry:new ol.geom.LineString
 (path)})]);
```



10. ábra
Útvonal a térképtár és a rajzterem között

4.2. Szöveges útvonalleírás

A térképen ábrázolt útvonal jobb megértéséhez szerettem volna egy útvonalleírást is készíteni, amely egy listában összefoglalja a navigációs instrukciókat. Ehhez először az *index.html*-ben létrehoztam egy új *div*-et a *panel* és a *map* mellett *utvonal* néven, majd a *style.css* fájlban beállítottam a kinézetét. Az *eszaki.js*-ben a *route* függvényen kívül definiáltam egy új változót: `var utvonaldoboz=document.getElementById('utvonal');` majd a függvényen belül folytattam a munkát az egyes szakaszok hosszának és egymással bezárt szögeik kiszámolásával.

```
var a=((Math.atan2(path[1][0]-path[0][0],path[1][1]-path[0][1]))
     *180/Math.PI);
var tav=0;
for (var i=1;i<pontok.length;i++) {
    d=Math.sqrt(sqr(path[i][0]-path[i-1][0])+sqr(path[i][1]-path[i-
    1][1]))*corr;
    tav+=d;
    if (i<pontok.length-1)
        var na=(Math.atan2(path[i+1][0]-path[i][0],path[i+1][1]-
        path[i][1]))*180/Math.PI;
    var da=na-a;
    if (da<-180) da+=360;
    if (da>180) da-=360;
    a=na;
```

Az elején az *a* változóban megkapjuk az első szakasz azimutját és a táv értékét nullára állítjuk. A *for* ciklusban a program kiszámolja az egyes szakaszok hosszát (*d*). A képlet végén található *corr* a korrekciós tényező, mivel a Web Mercator vetület sajátosságai miatt jelentős, a földrajzi szélességtől függő hossztorzulás lépett fel. A korrekció értékét a *route* függvényen kívül számolja ki a program és menti el egy változóba: `var corr=Math.cos(47.5/180*Math.PI);`. Végül a korrigált szakaszhosszokat hozzáadjuk a *tav* változóhoz. Az *na*-ba kerül a következő szakasz azimutja, a *da*-ba pedig a két szakasz által bezárt szög. A *da* értékének -180^{o} és 180^{o} között kell lennie, erről gondoskodik a végén a két feltételes utasítás. Végül az *a* megkapja az *na* értékét a következő ciklusra. A *for* ciklus azonban itt még nem ér véget, következik a navigációs instrukciók kiíratása az útvonaldobozba.

```
if (i==pontok.length-1||Math.abs(da)>15||['stairs','elevator'].indexOf
(pontok[i][3])>-1) {
    if (Math.round(tav)>0&&(tav>3||(i>1&&i<pontok.length-1)))
        utvonaldoboz.innerHTML+=nh+"Menj előre "+Math.round(tav) +"
        métert!</span><br/>";
    tav=0;
```

Az első *if* feltételben szerepel, hogy mely esetekben kell navigációs instrukciókat kiíratni: ha az útvonal vége jön, vagy ha kanyarodni kell, vagy ha lépcső, esetleg lift típusú

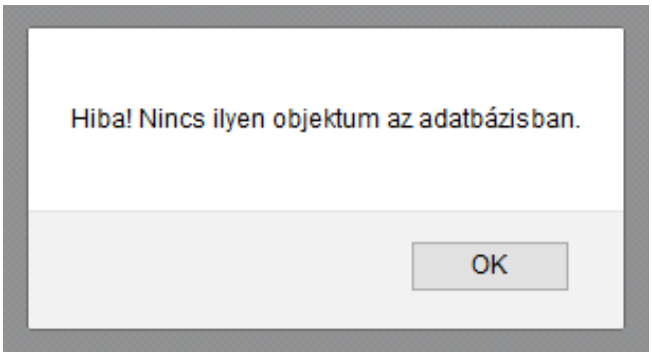
pont következik. A második feltétel az egyenesen haladás kiírásáért felel, amely csak abban az esetben megy végbe, ha a táv kerekített értéke nullánál nagyobb és nem az útvonal eleje vagy vége jön, vagy 3 méternél nagyobb. Ha megtörtént a kiírás, a *tav* értéke újra 0 lesz.

A következő lépésekben a szintváltásokat vizsgálja a program. A szintazonosítót a *path* tömb minden sorának második eleme tartalmazza (az indexelés nullától kezdődik). Két pont szintkülönbségének kiszámítása az `e=path[i][2]-path[i-1][2];` segítségével történik. Az *e* változó értéke alapján a következő esetek lehetségesek:

```
if (e!=0) {
      if (path[i][2]=="-1"||path[i][2]=="2"||path[i][2]=="3"||
          path[i][2]=="4"||path[i][2]=="6"||path[i][2]=="7") {
                utvonaldoboz.innerHTML+=nh+"Menj "+(e>0?"fel a ":"le
                a ")+path[i][2]+". emeletre! </span><br/>";
      continue;
      }
      if (path[i][2]=="0") {
          utvonaldoboz.innerHTML+=nh+"Menj "+(e>0?"fel ":"le ")+"a
          földszintre! </span><br/>";
      continue;
      }
      if (path[i][2]=="1"||path[i][2]=="5") {
          utvonaldoboz.innerHTML+=nh+"Menj "+(e>0?"fel az ":"le
          az ")+path[i][2]+". emeletre! </span><br/>";
      continue;
      }
      else {
          alert ("Hiba! Nincs ilyen objektum az adatbázisban.");
      }
}
else
     szintvaltas=false;
```

Az *e* értékének függvényében többféle üzenet jelenhet meg az útvonaldobozban. Ha a szintkülönbség nem nulla, és a *path[i][2]* szintazonosító mássalhangzóval kezdődik, akkor az első, magánhangzó esetén a harmadik üzenet jelenik meg. Ha pedig a szintazonosító nulla,

akkor a második *if* feltétel teljesül. A kiíráson belül a *fel* vagy *le* attól függően változik, hogy az *e* értéke negatív vagy pozitív-e. Abban az esetben, ha a szintazonosító nem -1 és 7 között van, valamilyen hiba történt, nagy valószínűséggel a hibásan megadott kezdő- vagy célpont azonosítója miatt üres tömböt adott vissza a *route_eszaki.php* fájl. Erre egy hibaüzenet figyelmezteti a felhasználót. *(11. ábra)* Ha pedig az *e* értéke 0, nem történt szintváltás, és a *szintvaltas* logikai változó értéke hamis, azaz *false* lesz.



11. ábra
A hibaüzenet

A program ezután a szintváltásokhoz kapcsolódóan a pontok típusának vizsgálatával folytatódik.

```
if (pontok[i][3]=="stairs"&&!szintvaltas) {
    szintvaltas=true;
    utvonaldoboz.innerHTML+="Menj a lépcsőhöz!<br/>";
    continue;
}
if (pontok[i][3]=="elevator"&&!szintvaltas) {
    szintvaltas=true;
    utvonaldoboz.innerHTML+="Menj a lifthez!<br/>";
    continue;
}
```

Az aktuális pont típusát a *pontok* tömb minden sorának harmadik eleme tartalmazza, az indexelés ez esetben is nullától történik. A program itt két feltételt vizsgál: egyfelől, hogy a típus lépcső (*stairs*) vagy lift-e (*elevator*), valamint, hogy a *szintvaltas* változó értéke hamis-e. Ha a feltételek teljesülnek, a logikai változó értéke igaz (*true*) lesz, az útvonaldobozban

pedig megjelenik egy újabb navigációs üzenet: *Menj a lépcsőhöz!* vagy *Menj a lifthez!* A *szintvaltas* logikai változó bevezetése a lépcsők összekötési módja miatt volt szükséges. Mivel a lépcsők csak a szomszédos szintekkel állnak összeköttetésben, így annyi *Menj a lépcsőhöz!* navigációs utasítás jelenne meg az útvonaldobozban, ahány emelet a különbség.

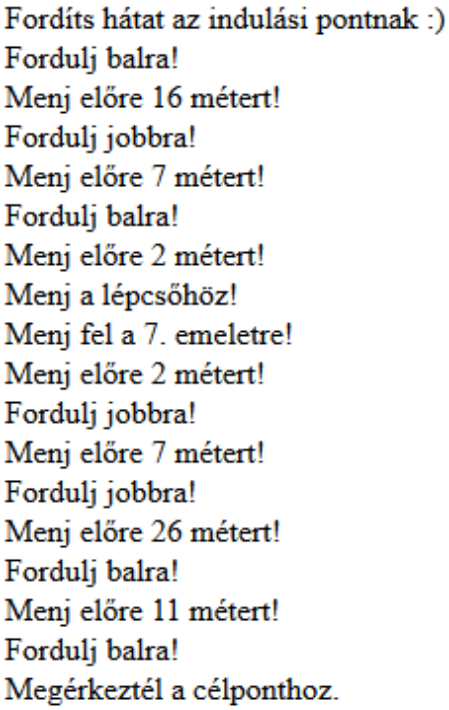
Azt, hogy az útvonalban elérkeztünk-e már az utolsó ponthoz, a következő feltétel vizsgálja. Ha teljesül, a *Megérkeztél a célponthoz* üzenettel zárul a navigációs instrukciók sora.

```
if (i==pontok.length-1&&i!=0) {
    utvonaldoboz.innerHTML+="Megérkeztél a célponthoz. <br/>";
    continue;}
```

A *route* függvény utolsó feltételsorozata a kanyarodásokat vizsgálja, vagyis az egymást követő szakaszok által bezárt szöget, a már korábban kiszámolt *da-t.*

```
if (da<-170)
   utvonaldoboz.innerHTML+=nh+"Fordulj vissza!</span><br/>";
else if (da<-45)
   utvonaldoboz.innerHTML+=nh+"Fordulj balra!</span><br/>";
else if (da<-15)
   utvonaldoboz.innerHTML+=nh+"Fordulj enyhén balra!</span><br/>";
else if (da<45)
   utvonaldoboz.innerHTML+=nh+"Fordulj enyhén jobbra!</span><br/>";
else if (da<170)
   utvonaldoboz.innerHTML+=nh+"Fordulj jobbra!</span><br/>";
else
   utvonaldoboz.innerHTML+=nh+"Fordulj vissza!</span><br/>";
```

A függvény ezen része a feltételeknek megfelelően megjeleníti az útvonaldobozban, hogy melyik irányba kell fordulni: balra, jobbra, esetleg enyhén balra, vagy enyhén jobbra. Az útvonalak megrajzolásakor elméletileg nem keletkezett 170 foknál nagyobb, vagy -170 foknál kisebb törés, emiatt a *Fordulj vissza!* utasítások fölöslegesek, de a program minden utasításra fel van készítve, ha a későbbiekben bármilyen változás történne az útvonalrajzon. A *route* függvény ezzel véget ért, így az útvonalrajz mellett már a szöveges útvonalleírás is teljesen elérhetővé vált a weboldalon. *(12. ábra)*



12. ábra
Útvonalleírás a 6.100-as és a 7.54-es terem között

4.3. A kezdő- és célpontmegadási segéd

Az útvonaltervezés legfontosabb eleme a pontos kezdő- és célpontmegadás. Ha elírás történik, vagy nem a kívánt útvonal jelenik meg, vagy egy hibaüzenetet kap a felhasználó, hogy nincs ilyen elem az adatbázisban. Emiatt szerettem volna egy olyan plusz funkcióval kiegészíteni a weboldalt, amely automatikusan ajánlásokat tesz egy listában a begépelt karakterek alapján, ahonnan pedig kiválasztható a kívánt objektum azonosítója vagy neve. Egy ilyen segédalkalmazás nemcsak pontosíthatja a keresést, hanem gyorsíthatja is.

Az alkalmazás működéséhez négy fájl szükséges: az *index.html*, a *style.css*, az *eszaki.js* és a *typing_hint.php*. Először a *style.css*-sel dolgoztam, ami a html fájlhoz tartozó CSS (Cascading Style Sheets) stíluslap, ebben tárolódnak a weboldal stílusdefiníciói: a *#map*, az *#utvonal* és a *#panel* is. Itt hoztam létre a *#typinghintbox* stílusdefinícióit is.

```
#typinghintbox {
    position: absolute;
    bottom: 30px;
    border: solid 1px #aaa;
    background: rgba(255,255,255,.6);
    max-height: 200px;
```

```
    overflow: auto;
}
#typinghintbox  div {
    cursor: pointer;
}
#typinghintbox  div:hover {
    background: gray;
}
```

A munkát az *index.html*-ben folytattam, ahol a *panel* azonosítójú *div* elemben kiegészítettem a két szövegdobozt egy-egy *onkeyup* eseménykezelővel:

```
Honnan: <input type="text" id="from" onkeyup="typingHint(this)"/>
Hova: <input type="text" id="to" onkeyup="typingHint(this)"/>
```

Az eseménykezelő minden egyes billentyűfelengedéskor meghívja a *typingHint* függvényt az *eszaki.js*-ben.

```
function typingHint(ib) {
var th=document.getElementById('typinghintbox');
if (!th) {
     th=document.createElement('div');
     th.id='typinghintbox';
     ib.parentNode.appendChild(th);
   }
 if (ib.value=='') {
     th.style.display='none';
     return;
    }
    th.style.position='absolute';
    th.style.bottom='30px';
    th.style.left=ib.offsetLeft+'px';
var x=new XMLHttpRequest();
x.open('get','http://terkeptar.elte.hu/~campusrouting/utvonal/typing_
hint.php?id='+ib.value,false);
x.send();
```

A függvény először létrehozza a *typinghintbox* azonosítójú *div* elemet, majd a szövegdoboz tartalmát (*ib.value*) elküldi a *typinghint.php*-nak. Ekkor a php fájl indul el, amely az útvonaltervezésért felelős php fájlhoz hasonlóan először csatlakozik az adatbázishoz, majd elküld egy SQL lekérdezést:

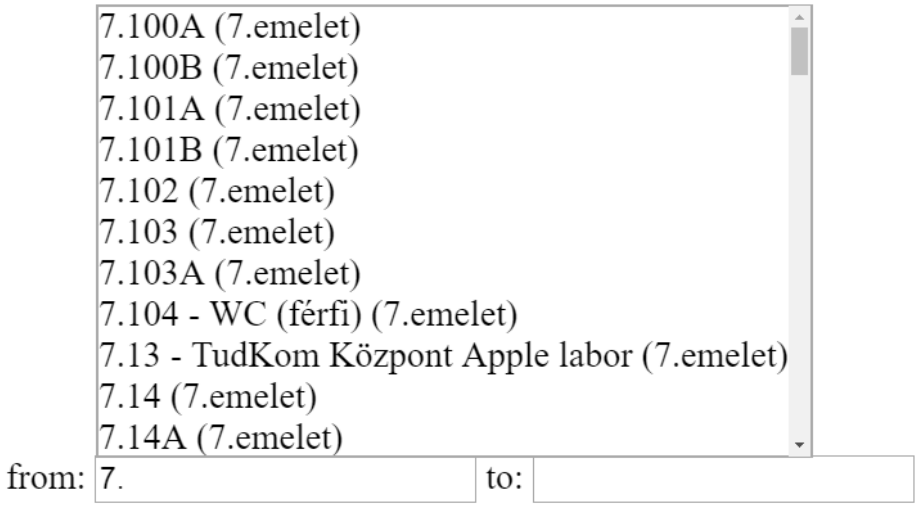
```
select * from nodes_eszak where lower(id) like lower('{$_GET['id']}%')
or lower(name) like lower('%{$_GET['id']}%') order by id;
```

A lekérdezés eredménye a *nodes_eszak* nézet összes olyan rekordja, melyben a szövegdobozba beírt karakterek az *id* mezőben tárolt érték elejének felelnek meg, vagy esetleg a *name* mező valahol tartalmazza ezeket a karaktereket. Az eredményt a *$s* tömbben tárolja el a program, és a `print json_encode($s);` paranccsal visszaküldi a JavaScript fájlnak, amely folyatja a munkát.

```
var h=JSON.parse(x.responseText);
if (h.length==0) {
    th.style.display='none';
    return;
}
th.innerHTML='';
for (var i=0;i<h.length;i++) {
    var t='';
    if (h[i].id)
        t+=h[i].id;
    if (h[i].name)
        t+=(t!=''?' - ':'')+h[i].name;
    t+=' ('+h[i].level+'.emelet)';
    var d=document.createElement('div');
    d.innerHTML=t;
    d.onclick=(function(room) {
        return function() {
            ib.value=room.id;
            th.style.display='none';
        }
    })(h[i]);
```

```
    th.appendChild(d);
}
th.style.display='block';
```

A visszaérkezett adatok a *h* változóba kerülnek, majd egy *for* ciklus feldolgozza azokat. Az eredmény egy olyan felugró táblázat, mely tartalmazza a lekérdezett adatok azonosítóját, nevét és szintazonosítóját. *(13. ábra)* Ha rákattintunk az egyik elemre, az *onclick* esemény-kezelő lép működésbe, és a kiválasztott elem azonosítóját (*id),* vagy ha az nincs, a nevét (*name*), valamint az adatbázis elsődleges kulcsát, a *gid*-t beírja a szövegdobozba. Utóbbira azért van szükség, mert az adatbázisban előfordulnak ugyanolyan nevű, azonosítóval nem rendelkező elemek, például az ülőhelyek és az automaták, ezért csak név vagy *id* alapján nem volna lehetséges a beazonosítás.



13. ábra
A kezdőpont megadását segítő táblázat

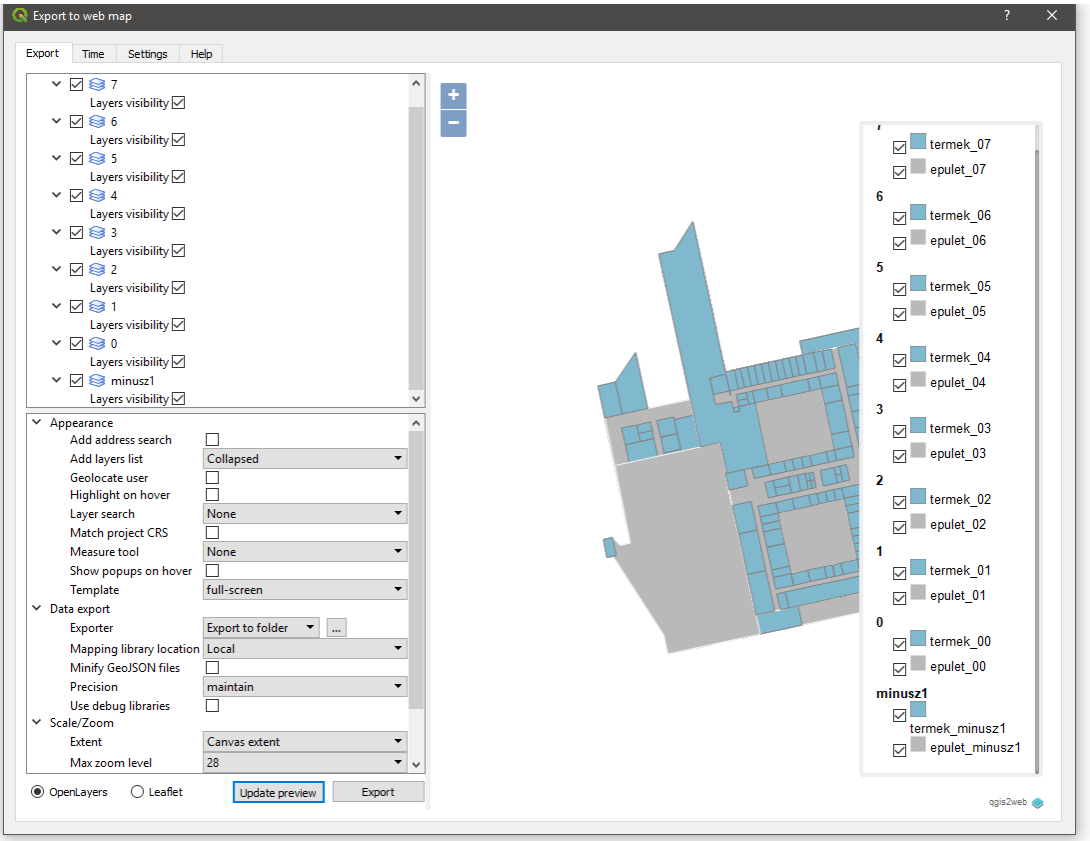
4.4. A háttértérkép

Az útvonaltervező lényegében elkészült, a megtervezett útvonal rajza és szöveges leírása is megjelent már a weboldalon, azonban a teljes megjelenítéshez és a tájékozódás segítéséhez hiányzott még a háttérből az egyetem térképe. Ezért visszatértem a QGIS-hez és létrehoztam egy új projektet *epulet.qgs* néven. A projekthez hozzáadtam az összes *termek* és *epulet* shapefájlt, és beállítottam a megjelenésüket.

A termek réteg stílusbeállításakor azonban problémába ütköztem. A qgis2web modul, amivel a webes térképet szerettem volna létrehozni, nem támogatta a réteg 2,5 D-s megjelenítését, a kilenc szint helyett csak egyetlen szint jelent meg, és az sem a beállítottak

szerint. Emiatt a 2,5 D-s stílusokat töröltem a projektből, és kategorizált megjelenítést állítottam be a különböző helyiségeknek a típusuk alapján. Sajnos ez a megoldás sem volt megfelelő, a qgis2web így már egyetlen réteget sem tudott megjeleníteni. A modul számára az egyetlen megfelelő stílusbeállítás az volt, ha a *termek* réteget kategorizálatlanul, csakis egyetlen színt használva jelenítettem meg.

Ezután már ténylegesen hozzáláthattam az útvonaltervező háttértérképének elkészítéséhez. Ehhez a qgis2web modult használtam, mely QGIS projektek egyszerű és gyors webes megjelenítésére szolgál. A modulban kiválasztottam a megjeleníteni kívánt rétegeket, vagyis minden szint *epulet* és *termek* layerét, valamint az *Add layers list* pontban a *Collapsed*-et állítottam be, mellyel egy automatikusan kinyíló és összecsukódó rétegválasztót adtam hozzá a weboldalhoz. *(14. ábra)* Végül az *Export* gombra kattintottam, és a program létrehozta a weboldalt az összes hozzá tartozó fájllal együtt.



14. ábra
A qgis2web modul

A fenti képen is látható, hogy a rétegválasztóban az elemek megjelenése nem volt megfelelő, ugyanis a *termek* és *epulet* rétegek külön is kapcsolhatók voltak, ráadásul egyszerre több szint is megjelenhetett, ami elég zavaró volt. A rétegek megjelenését a

program által létrehozott *layers* mappában található *layers.js* fájl módosításával tudtam befolyásolni. Ebben a fájlban külön-külön megtalálhatók a projekt rétegei. Itt a feladat az volt, hogy a különálló layerekből rétegcsoportokat alkossak. Először a hetedik emelethez tartozó elemeket vontam össze egyetlen csoporttá, majd ezt a többi emelettel is megtettem.

```
var group_7 = new ol.layer.Group({
            layers: [lyr_epulet_07_16,lyr_termek_07_17],
            title: "7. emelet",
            type: 'base',
            combine: true,
            visible: false});
```

A *group_7* a csoport neve, a *layers* a csoporthoz tartozó rétegeket tartalmazza, a *title* a csoport neve, amely majd a rétegválasztóban jelenik meg. A *type: 'base'* paranccsal pedig azt is elértem, hogy a jelölőnégyzetek helyett rádiógombok jelenjenek meg, emiatt egyszerre mindig csak egyetlen emelet rétegei látszódnak a weboldalon. A földszint esetében a *visible* értékét *true*-ra állítottam, így az oldal megnyitásakor alapértelmezetten a földszinthez tartozó rétegcsoport jelenik meg. Végül a `var layersList = [group_minusz1, group_0, group_1,group_2,group_3,group_4,group_5,group_6,group_7];` utasítással hozzáadtam a csoportokat a rétegválasztóhoz. A rétegcsoportok azonban nem jelentek meg a weboldalon, ott továbbra is külön szerepelt az összes layer. Végül kiderült, hogy a problémát a *resources* mappában található *ol3-layerswitcher.js* nevű fájl okozta, ezért lecseréltem a http://rawgit.com/walkermatt/ol3-layerswitcher/master/src/ol3-layerswitcher.js oldalon található JavaSript fájllal.

A következő lépés az útvonaltervező és a háttértérkép egyesítése volt, ugyanis az útvonaltervező tesztelésére használt *index.html* mellett a qgis2web is létrehozott egy *index.html* fájlt a háttértérképnek. Utóbbiból a <*script*> elemeket átmásoltam az útvonaltervezőbe a már létező `<script src="eszaki.js"></script>` elé. Ezek a <*script*> elemek tartalmazzák a weboldalhoz szükséges JavaScript fájlok elérési útvonalait. Ennek megfelelően a terkeptar.elte.hu *campusrouting* felhasználójának nyilvános mappájában is kialakítottam a mappaszerkezetet, ahová feltöltöttem az elkészült fájlokat. Szükség volt még az *eszaki.js* módosítására is, hogy a rétegválasztóban, és magán a térképen is minden megjelenjen. Alapvetően ugyanis csak az útvonalat tartalmazó *rs* réteg látszódott, a

háttértérkép nem, mivel az *eszaki.js*-ben a *map* változó csakis ezt a réteget kapta meg, a *layers.js layersList* nevű tömbjét nem.

```
var map=new ol.Map({
    target: 'map',
    controls: [new ol.control.Zoom(),new ol.control.FullScreen()],
    layers: [
        new ol.layer.Vector({
            title: 'Útvonal',
            source: rs,
        })],
```

Ezért a *layers*-t módosítottam, hogy csakis a *layersList* szerepeljen benne, ezáltal megjeleníthetővé vált a térkép, az útvonalat pedig a *push* parancs segítségével adtam hozzá a *layersList* tömbhöz.

```
layersList.push(new ol.layer.Vector({
        title: 'Útvonal',
        source: rs,
        })
);
```

4.5. Automatikus megjelenítési beállítások

Az útvonaltervező ezáltal képessé vált az útvonal térképes megjelenítésére is, azonban alapértelmezetten csak a földszint térképe jelent meg, és rajta az útvonal. A megfelelő emeletet kézzel a rétegválasztóban kellett beállítani. Az is zavaró volt, hogy hosszabb útvonal esetén nehéz volt az útvonalleírás alapján tájékozódni, problémás volt megtalálni a térképen azokat az útvonalpontokat, amelyekre az utasítás vonatkozott. Ezt csak tovább nehezítette, hogy egyedül csak a rétegválasztóban látszódott, hogy melyik szint az aktív, és többször előfordult, hogy rossz szintet választottam ki. A felsorolt problémák megoldására hoztam létre a *navihelp* függvényt, valamint ehhez kapcsolódóan további elemeket. Először az *index.html*-ben kialakítottam a negyedik *div* elemet *szint* id-vel, majd a *style.css* -ben megadtam a stílusdefinícióit. Ezután az *eszaki.js*-ben folytattam a munkát, létrehoztam a *szintdoboz* nevű változót *Földszint* kiírással, valamint egy új, *cps* nevű réteget. A *push* segítségével ezt a layert is hozzáadtam a *layersList*hez.

```
var szintdoboz=document.getElementById('szint');
szintdoboz.innerHTML='<h2>Földszint</h2>';
var cps=new ol.source.Vector();
layersList.push(new ol.layer.Vector({
            title: 'Aktuális pont',
            source: cps,
        })
);
```

Ezt követően létrehoztam a *navihelp* függvényt, mely egyszerre felel az aktuális pont és vonalszakasz megjelenítéséért, az automatikus szintváltásért és a szint kiírásáért.

```
function navihelp(x,y,level,szakasz) {
    var f=new ol.Feature({geometry:new ol.geom.Point([x,y])});
    cps.addFeatures([f]);

    if (typeof(szakasz)!="undefined") {
        var f2=new ol.Feature({geometry:new ol.geom.LineString
            (szakasz)});
        cps.addFeatures([f2]);
    }
      szintdoboz.innerHTML="<h2>"+level+ ". emelet </h2><br/>";
    var layers=map.getLayers().getArray();
    for (var i=0;i<layers.length;i++)
        if (typeof layers[i].getProperties().level!='undefined')
           layers[i].setVisible(layers[i].getProperties().level==level);
```

A függvény a *cps* nevű rétegbe berakja az aktuális pontot az x és y koordináták alapján, a *level* értékét kiírja a szintdobozban és a *for* ciklusban aktiválja az adott szint térképét. Az automatikus szintváltásért felelős kódrészlet még egyszer megismétlődik a *route* függvényben is, hogy a tervezést követően azonnal az első pont szintértékének megfelelő térképréteg aktiválódjon. A különbség annyi, hogy a második esetben a *level* változó értéke a *path* tömb nulladik elemének második helyen álló értékével fog megegyezni, ez tartalmazza ugyanis a kiindulási pont szintazonosítóját.

A *navihelp*re visszatérve, azt, hogy melyik az aktuális pont és szakasz, a *route* függvényben található *nh* segítségével tudjuk meg.

```
var nh="<span class='routehint' onmouseover='navihelp("+path[i][0]+",
"+path[i][1]+","+path[i][2]+","+JSON.stringify(szakasz)+")' onmouseout=
'hidenode()'>"
```

A *routehint* a *style.css*-ben a stílusdefiníciót tartalmazza, az *onmouseover* eseménykezelő érzékeli, ha az egér egy navigációs utasításra ér, majd meghívja a *navihelp* függvényt, melynek *x, y* és *level* értékeit a *path* tömbből olvassa ki. A *szakasz* tömb alapesetben mindig a *path* tömb nulladik elemét tartalmazza, azonban több egymást követő egyenes szakasz esetén nem történik mindig navigációs instrukció kiíratás, csakis forduló, lift, vagy éppen lépcső előtt. Azaz, amíg a *tav* változó értékét az egyes szakaszok hosszával (*d*) növeljük, addig a *szakasz* tömbben a `szakasz.push(path[i]);`-vel eltároljuk a szakaszok koordinátáit is, amelyet megkap a *navihelp*. Végül, ha az egér elmozdul a navigációs utasításról, elindul a *hidenode* függvény, amely a `cps.clear();` segítségével kiüríti a *cps* réteget. Ahhoz, hogy a navigációs instrukciókra mutatva elinduljon a *navihelp*, a *route* függvényben minden lehetséges szöveges utasítás elé be kellett szúrni az *nh* változót így: `utvonaldoboz.innerHTML+=nh+"`*[szöveges utasítás]* `</span><br/>";`.

5. A weboldal testreszabása

5.1. Az oldal megjelenése

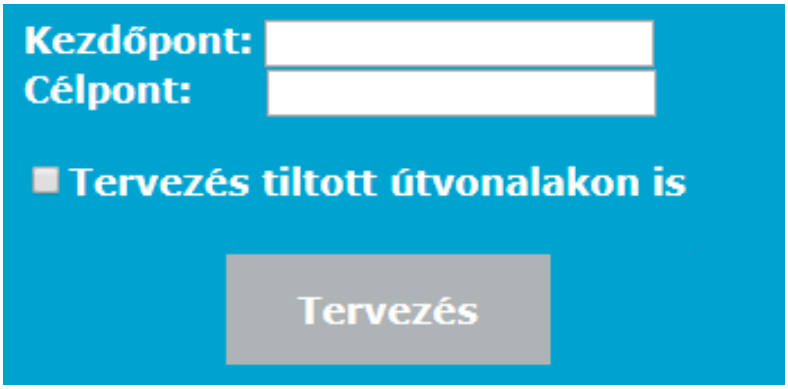
Miután az útvonaltervező elkészült, ideje volt kezdeni valamit a tervezőt futtató weboldal, az *index.html* kinézetével. Ezt a fájlt ez idáig csak a tesztelésre használtam, ennek megfelelően a megjelenése meglehetősen egyszerű volt. A munka nagy részét a *style.*css-ben végeztem, hiszen ez tartalmazza a stílusdefiníciókat. Először is a *panel* és az *utvonal* azonosítójú *div* elemeket egy oszlopba rendeztem, így a térkép a képernyő jobb, míg a tervező kezelőszervei és az útvonalleírás a bal oldalra került. Mivel a kezelőnek új helye lett, szükségessé vált a *typinghintbox* minimális módosítása, ugyanis zavaró volt, hogy kitakarta a *Célpont* mezőt, valamint az enyhe átlátszóság miatt nehezen volt olvasható a pontmegadási segéd tartalma. Két új *div* tárolóelemet is elhelyeztem az *index* fájlban *fejlec* és *lablec* azonosítóval. Előbbibe a weboldal címe, míg utóbbiba a készítői adatok kerültek. Végül a *css* fájlban minden elemnek megadtam egy háttérszínt, egy betűszínt és egy betűtípust.

```
background: #00a3cf;
color: white;
font-family: "Verdana";
```

A tesztelés során feltűnt, hogy a *Tervezés* gomb meglehetősen kicsi, főleg a telefon képernyőén nézve, ezért lecseréltem egy nagyobbra. *(15. ábra)* A gombot definiáló <*input*> elemet kiegészítettem a `class="button"` értékkel, így a *css* fájlban beállíthattam a megjelenését.

```
.button {
    position: relative;
    left: 25%;
    background-color: #aeb3b5;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
```

```
    margin: 4px 2px;
    cursor: pointer;
    font-family: "Verdana";
    font-weight: bold;
}
```



15. ábra
A „panel” azonosítójú tárolóelem a nagyobb Tervezés gombbal

5.2. A használati útmutató

Bár az útvonaltervező használata nem túlságosan bonyolult, bizonyos elemei, mint például a *Tervezés tiltott útvonalakon is*, vagy az automatikus megjelenítésért és kiemelésért felelős *navihelp* függvény működtetése némi magyarázatra szorulnak az oldal használói számára. A felhasználók kisegítésére egy 2+1 elemből álló útmutatót dolgoztam ki.

Készítettem egy képernyőképet a weboldalról, majd a CorelDraw X8 programba betöltöttem és megformáztam. Hozzáadtam két új réteget a projekthez, majd a *Callout Shapes tool* segítségével szövegbuborékokat rajzoltam a megfelelő helyekre. A szövegbuborékokat a *Text tool*lal kitöltöttem szöveggel, majd a használati útmutatót az *Export* gomb segítségével elmentettem *help.jpg* néven, és feltöltöttem a szerveren található *resources* mappába.

Az útmutató elérését kétféleképpen szerettem volna biztosítani: egyfelől egy hagyományos gomb segítségével, hogy bármikor elérhető legyen, másrészt egy automatikus megjelenést is akartam, ugyanis egyáltalán nem garantálható, hogy a felhasználók ténylegesen rá is kattintanának a segítség gombra. Azonban azt sem szerettem volna, hogy a gyakorlott felhasználók minden induláskor ezzel a felugró üzenettel szembesüljenek, ezért a weboldalak által nagyon gyakran alkalmazott technikát terveztem alkalmazni: a http-sütit, angol nevén a cookie-t. A süti egy olyan kisméretű szöveges információcsomag, amelyet a

szerver hoz létre a felhasználó gépén a böngésző segítségével. Ezt a böngésző minden kérés alkalmával visszaküldi a szervernek. Az útvonaltervező esetében ez a süti csak azért felelős, hogy ha már megtalálható a felhasználónál, azaz már legalább egyszer meglátogatta az oldalt, akkor a használati útmutató ablak nem ugrik fel többé.

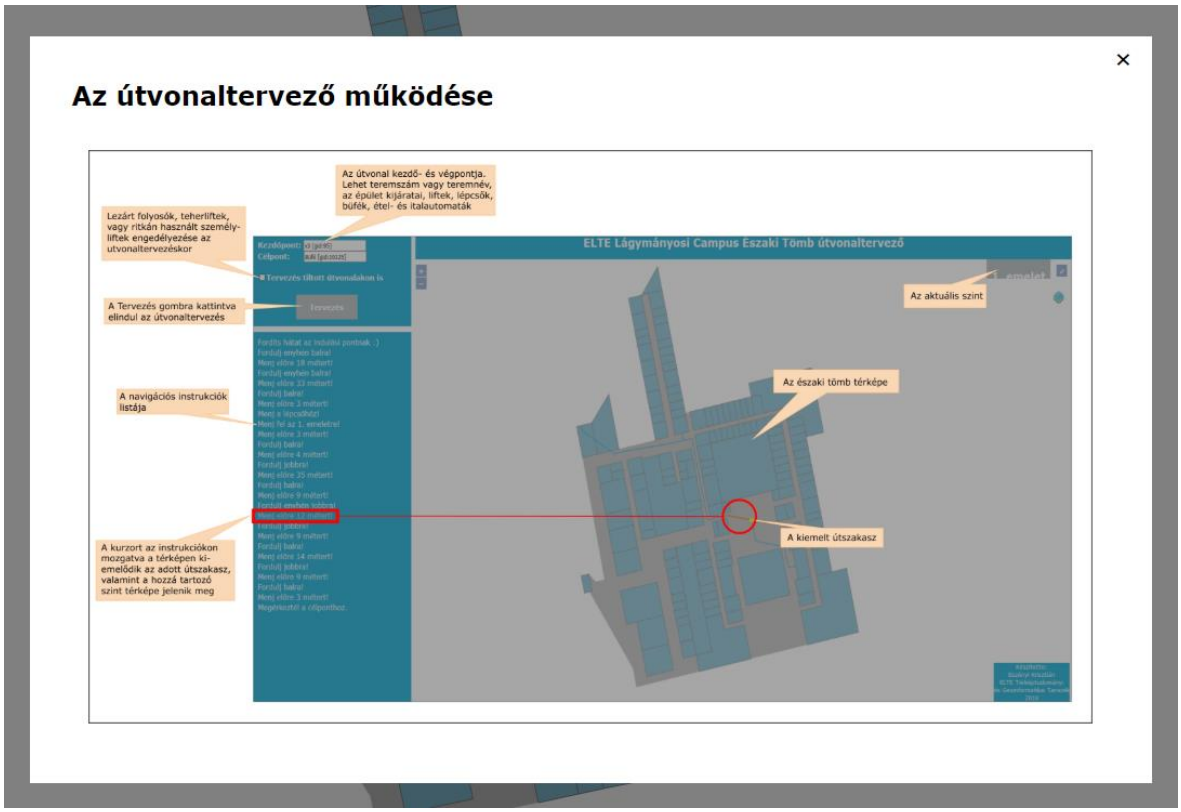
Szerencsére erre a célra már léteztek elkészített programok, mint például a First Visit Popup, melyet én is használtam. A First Visit Popup egy JavaScript fájl, amely valójában egy jQuery plugin. A jQuery az egyik legyakrabban használt JavaSript függvénykönyvtár, nagy előnye, hogy egyszerűsíti a kliens oldali programozást, ráadásul számos, már előre megírt plugin áll hozzá rendelkezésre, amit csak el kell helyezni a szerveren, és a használatához a weboldalon hivatkozni kell rá. A jQueryt még csak letölteni sem kell, elég, ha egy CDN-ről, azaz egy tartalomelosztó hálózatról hivatkozzuk be, mint például az ajax.googleapis.com. A weboldalon a következő három *script* elemet helyeztem el:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script src="resources/jquery.firstVisitPopup.js"></script>
<script>
      $(function () {
            $('#helpbox').firstVisitPopup({
                  cookieName : 'elte_utvonaltervezo',
                  showAgainSelector: '#help'
            });
      });
</script>
```

Az első hivatkozik a jQuery függvénykönyvtárra, a második a First Visit Popupra, melyet a *resources* mappába töltöttem fel, a harmadik elem pedig elindítja a jQuery függvényt. A függvényben található *#help* a *panel* azonosítójú *div*-ben lévő `<img src="resources/kerd.png" id="help" width=10% >` kérdőjelet ábrázoló képre hivatkozik, míg a *#helpbox* elem az ugyanilyen azonosítójú *div*-re.

```
<div id="helpbox">
      <h2>Az útvonaltervező működése</h2>
      <p><img src="resources/help.jpg" width=100%></p> </div>
```

Ebben a tárolóelemben található a felugró ablak tartalma, azaz a cím és a korábban elkészített *help.jpg*. Az oldal első megnyitása után az *elte_utvonaltervezo* nevű süti eltárolódik a felhasználó gépén, így ez az ablak többé nem fog automatikusan felbukkanni. Manuálisan viszont a *panel*ben elhelyezett kérdőjel képére kattintva bármikor előhívható. Végül a First Visit Popup megjelenéséért felelős stílusdefiníciókat átmásoltam a *style*.css-be, és ezzel elkészült a használati útmutató. *(16. ábra)*

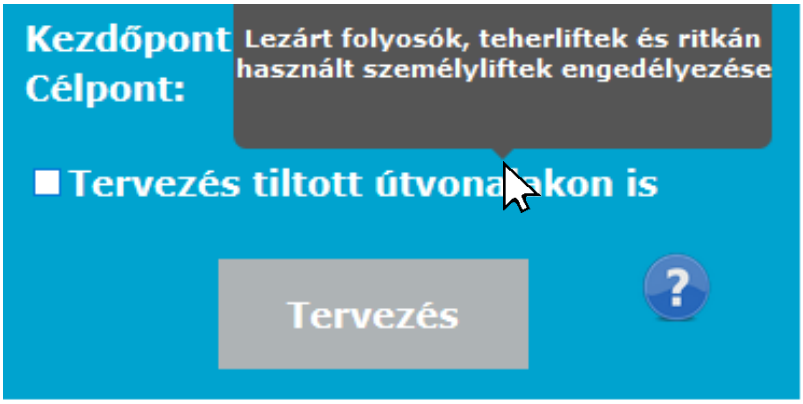


16. ábra
Az útvonaltervező használatát bemutató felugró ablak

A tesztelés során felmerült, hogy esetleg a tiltott útvonalak engedélyezése lehetőség külön magyarázó szövegbuborékot kaphatna, mivel ez egy elég fontos funkciója az útvonaltervezőnek. Pontosan erre a célra való a tooltip, vagy magyarul súgóbuborék. Az interneten számos tooltip programkódja elérhető, így a feladat csak annyi volt, hogy kisebb módosítások végrehajtása után a tervező fájljaiba a megfelelő helyekre be kellett illeszteni a kódot. Először a *style.css*-ben elhelyeztem a tooltiphez tartozó stílusdefiníciókat, majd az *index.html*-ben a jelölőnégyzetet és a hozzá tartozó szöveget a következőképp módosítottam:

```
<span style="white-space: nowrap"><input type="checkbox"
id="enableRestricted"><label for="enableRestricted">
<div class="tooltip2">Tervezés tiltott útvonalakon is
<span class="tooltiptext"><h6>Lezárt folyosók, teherliftek és ritkán
<br> használt személyliftek engedélyezése</h6></span></div></label>
</span>
```

A kód eredménye, hogy ha a *Tervezés tiltott útvonalakon* felirat fölé kerül az egér, akkor fölötte egy szövegbuborékban megjelenik a `<span class="tooltiptext">` és a `</span>` között található felirat. *(17. ábra)*

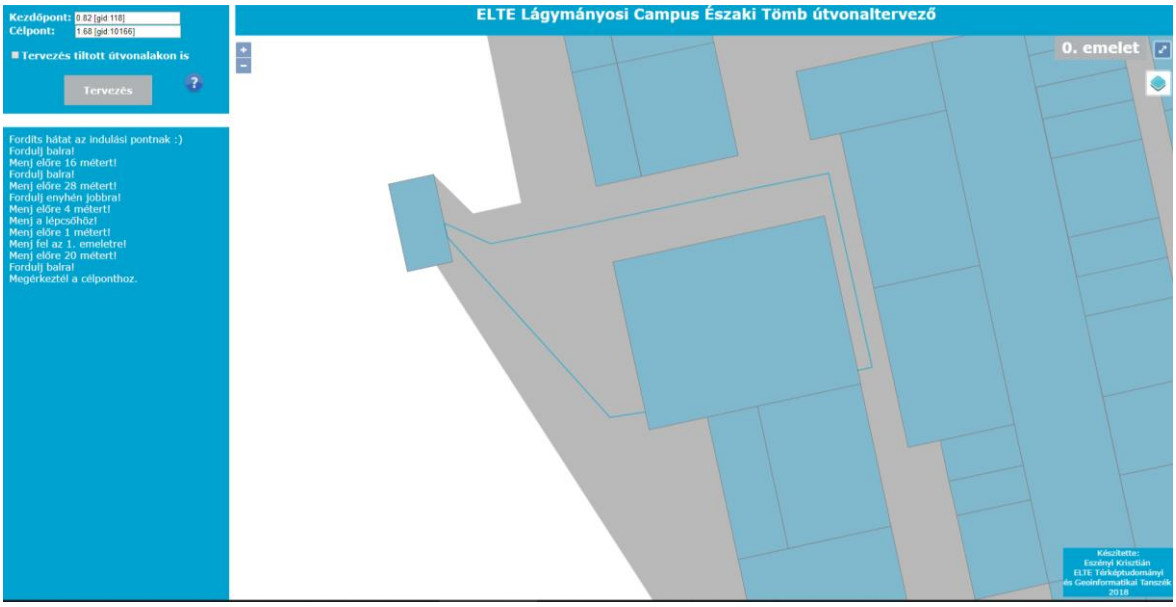


17. ábra
A tooltip, vagy súgóbuborék megjelenése

6. Továbbfejlesztési lehetőségek

A munka kezdetén felállított tervet sikerült teljesíteni: az alkalmazás elkészült, az útvonaltervezés működik. Az alap útvonaltervezőt számos plusz funkcióval elláttam, mely a program használóját segíti, például a szöveges navigációs utasítások, az automatikus térképrétegváltás, vagy a kezdő- és célpontmegadási segéd. Ez azonban nem jelenti azt, hogy az alkalmazás teljesen elkészült volna. Valójában még számtalan továbbfejlesztési lehetőség rejtőzik benne, mind az útvonaltervezés, mind a megjelenés területén.

Az útvonaltervezés területén egy nagy korlátozó tényező, hogy jelenleg csak konkrét, egyedi objektumokra lehet keresni. Ezzel hagyományos termek esetén nincs is gond, mert egy helyiséghez egy ajtó tartozik, azt pedig az útvonaltervező gyorsan megtalálja. Több bejárattal rendelkező termek ajtajai azonban nem tudnak egymásról, így akkor is a tervezéshez megadott ajtón keresztül vezet a megtervezett útvonal, ha egy másik ajtón keresztül hamarabb bejuthattunk volna a terembe. Ez leginkább a nagy előadótermek esetében jelent problémát, ahol az egyes bejáratok nem is egy szinten vannak (például az Eötvös terem: 0.82-0.83-1.67-1.68). *(18. ábra)* Az egyedi objektumok keresése az ülőhelyek, automaták és mosdók esetében sem túl praktikus, ez esetben célszerűbb lenne egy olyan új funkció, mely segítene legközelebbi objektumot megtalálni.



18. ábra

Az Eötvös terem esetén plusz 69 métert és egy emeletnyi lépcsőzést okoz a két egymásról nem tudó ajtó

Fontosnak tartanám új helyszínek bevonását az adatbázisba. Mivel a lágymányosi campuson nagyon gyakori, hogy a hallgatóknak egyszer az északi, máskor pedig a déli tömbben van órájuk, első sorban a déli tömb teljes, alapos, az északi tömbnél megalkotott feltételek alapján történő felmérése lenne szükséges. A két épületet ezután a kijárataiknál fogva össze lehetne kötni, és így máris lehetővé válna az épületek közti útvonaltervezés. Ha pedig már kimerészkedtünk a szabadba, a campus környezetét is fel lehetne mérni, leginkább a közeli busz- és villamosmegállókat, MOL Bubi gyűjtőállomásokat és a hajóállomást.

A megjelenés területén elsősorban a térkép tartogat számos fejlesztési lehetőséget: termek kategorizálása típus alapján, teremszámok feltüntetése, POI-k, kijáratok, liftek, lépcsők és mosdók ábrázolása térképi jelekkel. Sajnálatos, hogy a qgis2web modul korlátai miatt ezek az alapvető térképi elemek egyelőre nem jeleníthetők meg egyszerűen. Ha a térképen szerepelnének teremazonosítók, a kezdő- és célpontmegadás történhetne a teremszám vagy név begépelése helyett egyszerű kattintással is az adott objektumra. Az útvonaltervező kiegészülhetne egy, manapság igencsak népszerű 3D-s megjelenítési móddal is, mely az egész épületet egyben mutatná, a szinteket kissé eltolva, hogy az áttekinthetőség azért megmaradjon. Nagy kérdés, hogy ez mennyire bonyolítaná meg a térkép kezelését. Ráadásul azzal is számolni kell, hogy a felhasználók egy jelentős része a mobiltelefonján fogja az útvonaltervezőt használni, így a 3D csak tovább rontaná a térkép olvashatóságát, valamint a térbeli megjelenítés nagyobb erőforrásigénye miatt bizonyos telefonokon lassabban, vagy akár egyáltalán nem működne a tervező. Az okostelefonon történő használatot legjobban talán egy mobilapplikáció kifejlesztése tenné a legegyszerűbbé, hogy ne kelljen folyton megnyitni a weboldalt. Addig is viszont csak a telefon vagy a számítógép böngészőjén keresztül érhető el az útvonaltervező a
http://terkeptar.elte.hu/~campusrouting/utvonal címen.

7. Összefoglalás

Diplomamunkám elkészítésének az volt a célja, hogy készüljön egy olyan alkalmazás, amely az ELTE Lágymányosi Campus Északi Tömbjén belüli útvonaltervezésre használható. Mivel az épületen belül minden ajtónak egyedi azonosítója van, így a kezdő- és célpontmegadás ezen azonosítók segítségével történik, éppen ezért a rendszerhez nincs szükség semmilyen beltéri helyzetmeghatározó szenzor alkalmazására.

Dolgozatom első részében az északi tömb felmérését mutattam be: először QGIS-ben rendszereztem a kapott állományokat, és szintenként különálló projektfájlokat készítettem. A felmérést QGIS mobil eszközökön futó verziójával, a QFielddel végeztem. A munka során az épület összes ajtajának pontos helyét és adatait (helyiség típusa, azonosító, név) rögzítettem a programban. Közben több nehézségbe is ütköztem: előfordult, hogy nem volt megfelelő az alaprajz, számos esetben nehéz volt a teremazonosítás, hol a megrongált, átragasztott táblák miatt, hol pedig azért, mert kívülről semmi nem utalt a helyiség funkciójára. A felmérést az adatok feldolgozása követte: hibajavítások és a pontokat összekötő útvonalhálózat megrajzolása. A kész állományokat végül feltöltöttem a Térképtudományi és Geoinformatikai Tanszék nyílt forráskódú webGIS szerverére, a wms.elte.hu-ra. Itt a PhpPgAdmin webes adminisztrációs alkalmazás segítségével kialakítottam az útvonaltervezés alapjául szolgáló táblákat és nézeteket.

A dolgozat második felében az útvonaltervező webes felhasználói felületével foglalkoztam: bemutattam a tervezés menetét, a megkapott útvonal grafikus és szöveges megjelenítését. A könnyebb kezelés érdekében készítettem egy kezdő- és célpontmegadási segédet, valamint az egyes szintek térképrétegeit is elhelyeztem a weboldalon. Ezt követően az oldal megjelenésével foglalkoztam, majd pedig egy használati útmutatót készítettem az oldal felhasználói számára.

Végül az utolsó fejezetében bemutattam, hogy számos továbbfejlesztési lehetőség rejlik még az útvonaltervező alkalmazásban. Azonban a diplomamunka kezdetén felállított célokat sikerült elérni: elkészült egy olyan alkalmazás, amely mindenféle külső szenzor nélkül, kézi pontmegadással képes a felhasználókat az egyik helyről a másikra elnavigálni az északi tömbön belül. Remélhetőleg ez az útvonaltervező a segítségére lesz majd mindazoknak, akik jelenleg még tanácstalanul bolyonganak az épület folyosóin a következő órájuk helyszínét keresve.

8. Hivatkozások

8.1. Felhasznált források

Fekete István, Hunyadvári László: Algoritmusok és adatszerkezetek

- http://tamop412.elte.hu/tananyagok/algoritmusok/index.html
- Utolsó elérés: 2018. május 16.

Dr. Elek István: Adatbázisok, térképek, információs rendszerek

- http://mapw.elte.hu/elek/adatmodellek.pdf
- Utolsó elérés: 2018. május 16.

Fazekas László: jQuery alapok

- http://gp.estontorise.hu/archives/8131
- Utolsó elérés: 2018. június 2.

http://people.inf.elte.hu/ladqaai/adb/ADB/adatb_13%20(nezettablak).pdf

- Utolsó elérés: 2018. június 2.

https://de.wikipedia.org/wiki/PostGIS

- Utolsó elérés: 2018. június 2.

http://informatika.gtportal.eu/index.php?f0=acc_kulcs_01

- Utolsó elérés: 2018. június 2.

https://github.com/chriscook/first-visit-popup

- Utolsó elérés: 2018. június 2.

https://www.progen.hu/nm/help/tooltip.htm

- Utolsó elérés: 2018. június 2.

https://hu.wikipedia.org/wiki/HTTP-s%C3%BCti

- Utolsó elérés: 2018. június 2.

8.2. Felhasznált órai jegyzetek

- Dr. Gede Mátyás: Script nyelvek
- Cserép Máté: Térinformatikai algoritmusok
- Dr. Elek István: Térinformatika (vektoros és raszteres modellek)

9. Köszönetnyilvánítás

Mindenekelőtt szeretnék köszönetet mondani témavezetőmnek, dr. Gede Mátyás adjunktusnak a rengeteg segítségért, útmutatásért és lelkesedésért, mely nélkül munkám nem készülhetett volna el. Köszönettel tartozom Barancsuk Ádám és Szigeti Csaba doktoranduszoknak a sok-sok hasznos tanácsért és a kapott fájlokért. Köszönöm továbbá Kiss Veronika Flórának, hogy expedíciós központnak használhattam a könyvtárt, valamint Varga Csaba Gergelynek, hogy az útvonaltervező főtesztelőjeként számos problémára fényt derített.

10. Mellékletek

A diplomamunkához mellékelt CD tartalmazza:

- a dolgozatot PDF formátumban
- a felmérés eredményeit tartalmazó QGIS projektfájlokat
- a weboldalhoz és az útvonaltervezőhöz szükséges fájlokat.

11. Diplomamunka leadási és eredetiség nyilatkozat

Alulírott **Eszényi Krisztián**, Neptun-kód: **QJX2W4**
az Eötvös Loránd Tudományegyetem Informatikai Karának, Térképtudományi és Geoinformatikai Tanszékén
Épületen belüli navigációs segéd
című diplomamunkámat a mai napon leadtam.

Témavezetőm neve: **dr. Gede Mátyás**

CD-t / DVD-t mellékelek *(aláhúzandó)*: igen nem

Büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom, hogy jelen szakdolgozatom/diplomamunkám saját, önálló szellemi termékem; az abban hivatkozott szakirodalom felhasználása a szerzői jogok általános szabályainak megfelelően történt.
Tudomásul veszem, hogy szakdolgozat/diplomamunka esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

A témavezető által benyújtásra elfogadott diplomamunka PDF formátumban való elektronikus publikálásához a tanszéki honlapon

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, 2018. június 11.

……..
hallgató aláírása