Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék

Szakdolgozat

Miskolci Egyetem

Közlekedés modellezése ...

Készítette:

Kása Dániel Zoltán

Programtervező informatikus BSc

Témavezető:

Piller Imre, egyetemi tanársegéd

Miskolc, 2019

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Kása Dániel Zoltán (NEPTUN) programtervező informatikus.

A szakdolgozat tárgyköre: Szabály alapú rendszerek, számítógépi grafika, szimuláció

A szakdolgozat címe: Közlekedés modellezése ...

A feladat részletezése:

...

Témavezető: Piller Imre (egyetemi tanársegéd)

A feladat kiadásának ideje: 2018. ???.

.....
szakfelelõs

Eredetiségi Nyilatkozat

Alulírott **Kása Dániel Zoltán**; Neptun-kód: `NEPTUN` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős programtervező informatikus szakos hallgatója ezennel büntetõjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *CÍMET ÁT KELL ÍRNI!* címû szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

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

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ... év ..... hó ..... nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

2. fejezet

Témaköri áttekintés

2.1. Hasonló célú szoftverek

2.1.1. Road Traffic Library

Az eddigi közlekedés-szimuláló szoftverek közül kiemelném először az AnyLogic által készített Road Traffic Library-t. Ezen szoftver segítségével létre lehet hozni egy úthálózatot különböző elemekből, mint például egyenes, kanyar, útkereszteződés, híd, zebra, lámpás útkereszteződés, valamint buszmegállók és parkolók. A szoftver képes különböző közlekedési szabályok alkalmazására, valamint a járművek intelligens módon választják meg a haladáshoz szükséges útvonalat. Valós-idejű szimuláció futtatására is van lehetőség, 2D-ben és 3D-ben is.

2.1. ábra. Kép a Road Traffic Library-ből

A szimulációkat felhasználva hőtérkép is előállítható, amely megmutatja mennyire

voltak forgalmasak az adott területek, így egyértelművé válik hol alakul ki könnyedén forgalmi dugó.

Lehetőség van a geoinformációs rendszerekben használt vektorgrafikus formátum, azaz shapefile importálására is. Ezen fájl alapján a szoftver automatikusan előállítja az úthálózatot. A szoftver bővíthető még gyalogos- és vasútforgalomra vonatkozó könyvtárakkal is.

A gyalogos könyvtár segítségével szimulálható a gyalogosforgalom sűrűsége különböző környezetekben. Ilyen környezet lehet például egy buszmegálló, vagy vasútállomás területe. Az egyes épületek felépítése is befolyásolja a forgalom sűrűségét, a gyalogosok minden akadályt, ebbe beleértve egymást is, próbálnak elkerülni.

A vasút könyvtár segítségével egy pályaudvar vasútforgalma komplex módon szimulálható. A forgalmat befolyásolja a tehervonatok rakodási ideje, a karbantartás, üzleti folyamatok lefolyása, és sok más további esemény is.

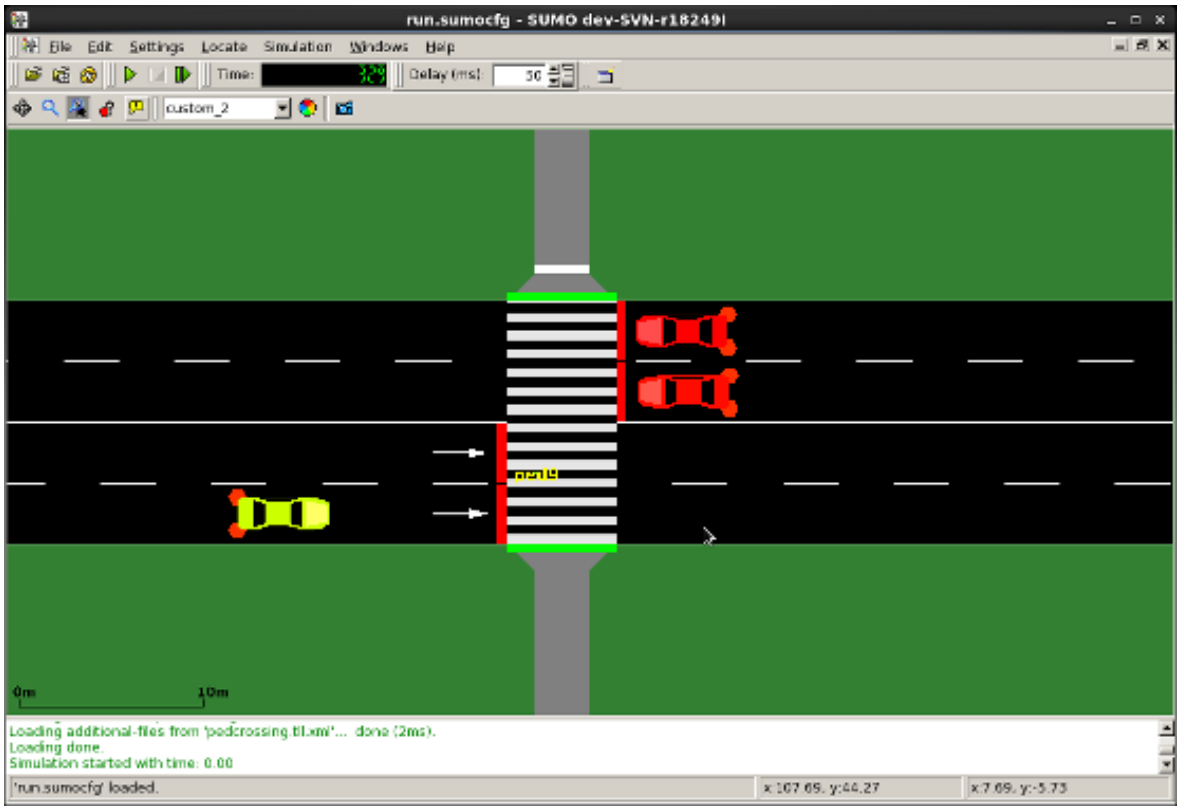
2.1.2. SUMO

A SUMO, azaz Simulation of Urban MObility egy ingyenes, nyílt, közlekedés-szimuláló C++-ban írt szoftver. Az első verzió 2001-ben került kiadásra, azóta már az 1.1.0-ás verziószámnál jár. Ez a szoftver lehetőséget ad több közlekedési forma (tömegközlekedés, gyalogosok, stb) modelezzésére a személygépjárművek mellett. Továbbá nagy eszköztárral bír az olyanokhoz mint útvonaltervezés, káros-anyag kibocsátás, valamint vizuális megjelenítés. Lehetséges továbbá meglévő úthálózat importálása fájlból, valamint személyre szabott modellek használata. A SUMO nem tartalmaz semmilyen megkötést az úthálózat méretét, vagy az egyszerre szimulálható járművek számát illetően.

A program lehetőséget ad a szimuláció online történő kezelésére is, a TraCI, azaz Traffic Control Interface használatával. Ilyen esetben a SUMO tulajdonképpen egy szerver, amelyet egy vagy több kliensen keresztül tudunk vezérelni. TCP alapú kliens/-szerver architektúrát használ.

A közlekedési lámpák viselkedését egyénileg be lehet állítani, vagy akár előre megírt TLS programból is lehetséges importálni. Az importáláshoz számos python szkript áll rendelkezésre. Ha nem áll rendelkezésre TLS program, a SUMO képes automatikusan generálni egyet. Ilyenkor alapértelmezett értékeket rendel minden paraméterhez amelynek értéke nem volt kapcsolókkal definiálva. Ezek közé tartoznak:

- –tls.cycle.time - lámpák váltási ideje.
- –tls.yellow.time - mennyi időre sárga a lámpa, ez alapértelmezetten az utak sebességhatára alapján történik kiszámításra.
- –tls.minor-left.max-speed - az útkereszteződésben balra kanyarodást megengedő sebességhatár (ha az út sebességhatára ezen érték felé esik, nem engedélyezett a balra kanyarodás.)
- –tls.left-green.time - ha egyszerre kap zöldet az egyenesen haladó, és a vele szemből balra kanyarodó, ezen érték idejéig a balra kanyarodó kap elsőbbséget.
- –tls.allred.time - minden zöld lámpa előtt ezen értéknek megfelelő ideig piros az összes lámpa (alapértelmezetten 0.)



2.2. ábra. Kép a SUMO programból

- –tls.default-type - actuated értékre állítva változó hosszúságú ideig tartanak a zöld lámpák.

A szoftver alkalmas a közlekedési lámpák teljesítményének vizsgálatára, olyan útvonal-tervezésre amely igyekszik a káros-anyag kibocsátást és a légszennyezést minimalizálni. A gyakorlatban történt híresebb alkalmazásai tartozik például a 2006-os labdarúgó világbajnokság-, valamint a Pápa 2005-ös Kölni látogatása során a közlekedés előrejelzése.

Maga a programcsomag tartalmazza a parancssori SUMO szoftvert, a grafikus felülettel ellátott GUISIM-et, egy úthálózat importáló NETCONVERT szoftvert, úthálózat generáló NETGEN szoftvert, egy OD2TRIPS nevű alkalmazást amely O/D mátrix alapján generál útvonalat, számos útvonal generátort, valamint egy grafikus úthálózat-szerkesztő alkalmazást.

2.2. Meglévő térkép-adatbázisok modellje

A Google Maps az egyik legelterjedtebb úthálózat-vizualizálásra alkalmas szoftver. Ez egy JavaScript alapú webalkalmazás, mely a geoinformációs adatok megjelenítése mellett valós-idejű forgalom elemzésre is képes.

Ehhez az adatokat több féle forrásból állítják elő. A Base Map program keretében rengeteg különböző forrásból szereztek vektoradatokat a létező úthálózatokról. Később ennek a nevét Geo Data Upload-ra váltották, viszont a lényege ugyan az maradt. Továbbá szereznek adatokat műholdképekből, Android rendszerű okostelefonok szolgáltatásai

által, valamint a nemrég bevezetett "Helyi Idegenvezetők" funkcióval, amin keresztül bárki tölthet fel adatokat.

A valós-idejű forgalomelemzéshez kezdetekben a helyi önkormányzatok által szolgáltatott adatokat használták, melyeket az egyes helyeken felszerelt szenzorok segítségével gyűjtöttek be. Viszont ezek csak a legforgalmasabb utakról adtak információt, ezért később áttértek a "crowd-source" módszerhez. Ezzel a módszerrel egy Google Maps-et futtató okostelefon GPS adatait felhasználva ad becslést egy út forgalmasságára. Lényegében azt elemzi, mennyien küldtek GPS adatot egy adott útszakaszon azonos időben.

Az utak és más objektumok kirajzolásához a böngésző leegyszerűsített vektoradatokat kap, melyek alapján felépíti az azoknak megfelelő formákat. Minden egyes objektumhoz tartoznak különböző metaadatok, mint például a sebességkorlát, haladási irány, út típusa (főút, stb), valamint egy név. Az objektumok JSON formában vannak tárolva, maga a Google Maps API pedig támogatja a GeoJSON forrásból történő adatok betöltését térképekre. Ezen információk alapján épül fel a modell amit a böngésző 2D vagy 3D formában ábrázol egy térképen.

2.3. GeoSpatial adatbázisok

A GeoSpatial adatbázisok adják az előbbi szolgáltatások hátterét. Ezek az adatbázisok kifejezetten a térbeli objektumokat leíró adatok tárolására, azok egyszerű lekérdezésére vannak optimalizálva. A leggyakoribb adatok pontok, vonalak, vagy poligonok formájában fordulnak elő.

A hagyományos adatbázisokhoz képest sokkal nagyobb teljesítményt nyújtanak ilyen területen, valamint gyakran az ilyen típusú adatok feldolgozásához bővíteni kell azok funkcionalitását. Ennek érdekében az Open Geospatial Consortium 1997-től kezdődően definiálta az ISO 19125 szabványt ezen funckionalitás megvalósításához, melynek második része leírja az SQL-ben történő megvalósítást is. Az OpenGIS szabvány ugyan magába foglalja a CORBA és az OLE/COM implementációkat is, ezek nem részei az ISO szabványnak.

A szabvány többek között a következő műveleteket definiálja:

- Térbeli számítások, azaz objektumok közti távolság, vonalak hossza, stb
- Térbeli predikátumok, objektumok közötti kapcsolatokat leíró logikai lekérdezések
- Geometriai konstruktorok, alakzatot leíró adatok alapján új geometriai elem létrehozása
- Egy tetszőleges objektumhoz tartozó információk lekérdezése

3. fejezet

Matematikai modell részletezése

- Modellezési probléma specifikálása - Pontosan milyen elemekből kellene felépíteni az adott város/térkép közlekedését? - Mi lenne a modell célja? - Irányított élek/egyirányú utak, többsávos utak modellje.

3.1. Modell specifikálása

A közlekedésre irányuló modell fő tulajdonságai közé kell hogy tartozzon az elegendő információ tárolása ahhoz, hogy alapvetően tudjanak a szimulált járművek rajta közlekedni. Ezt rengeteg féle képpen meg lehet valósítani, sokféle eleme lehet egy adott térképnek, akár minden egyes előforduló úttípusra, szabályra, táblára tekinthetünk úgy mint a modell egyedi része. Ez a megközelítés viszont lehetségesen túlbonyolítaná a modellt, ezáltal a procedurális generálás nem adna annyira elfogadható úthálózatot, lehetségesen életszerűtlen helyzetek épülhetnek fel a sok elem megléte, azok elhelyeződése miatt. Sokkal inkább célszerű a modell részeként tekinteni a főbb építőelemeket, melyek elegendőek magához az úthálózat generálásához, valamint néhány alapvető szabályt megvalósító elemet, mint például a lámpás útkereszteződés, az egyirányú út, valamint a többsávos út.

3.2. A modellben használt elemek

3.2.1. Egyenes út

Alapvető egyenes útszakasz, ennek lehet több paramétere is. Ennek az elemnek elsősorban tartalmaznia kell a rá vonatkozó sebességhatárt, valamint az úton létező sávok számát. Egyszerűen jellemezhető az elem a kezdő és a végpontjával, ugyanis az út szélessége annak paramétereiből (sávok száma) már adódik.

3.2.2. Egyirányú út

Hasonló az egyenes úthoz, viszont egyértelműen meg kell határozni a két végpont közül melyik a kiindulási, melyik a célpont. Ugyan azok a megkötések vonatkozhatnak rá, mint az egyenes út szakaszra, azaz sebességkorlát, valamint sávok száma.

3.2.3. Kanyar

Vehető tulajdonképpen külön elemnek, de megvalósításában akár sok egymást követő egyenes szakasz, melyek közt kis folyamatos közelítések vannak a kanyar ívére. Ezen esetben elég a kanyar kezdő és végpontját, valamint a használt egyenes szakaszok számát megadni annak jellemzésére. Alternatívaként a kanyar szöge is felhasználható jellemzésre. Különleges eset ha a kanyar egy egyirányú úton történik, ilyenkor az egyetlen változtatás az, hogy a kanyar közelítésére használt útszakaszoknak egyirányúnak kell lenniük.

3.2.4. Útkereszteződés

Három vagy több út találkozásánál használatos elem. Tulajdonságai közé tartozhat az hogy lámpás útkereszteződés, vagy sem, valamint ha nem lámpás akkor van-e megállási kötelezettség. Lehetséges valamint egyszerű esetben, azaz betorkolló útnál eljelölni a felülrendelt és alárendelt utat is. Az útkereszteződés jellemezhető annak középpontjával, valamint az azt érintő utak halmazával.

3.2.5. Személygépjármű

Legalapvetőbb közlekedési jármű. Az autókról nyilván kell tartani azok jelenlegi pozícióját, ebbe beleértve hogy melyik objektumon tartózkodnak éppen, és annak pontosan milyen pontján. Továbbá azt is hogy merre tartanak éppen, lokálisan a jelenlegi útszakasz végpontját, globálisan pedig a városban azon csomópont, ahova el szeretne jutni. Ezen elem jellemzői közé tartozik a sebessége, valamint többsávos úton hanyadik sávban közlekedik éppen. Tudnia kell a vele egy úton közlekedő járművekől is, azok pozíciójától függ a viselkedése. A járművek kezdetben véletlenszerű helyen kezdenek, majd ha sikeresen eljutottak a célpontjukhoz eltűnnek a modelből (leparkolt kocsinak tekintve). Paraméterezést tekintve ilyenkor új jármű jelenik meg ha jelenleg kevesebb van mint a beállított maximális érték.

3.2.6. Autóbusz

Kissé különlegesebb járműtípus, vonatkozik rá néhány további szabály. Többek között elsőbbsége van megállóból elindulva, útkereszteződésnél nagy ívben kell fordulnia, tehát többsávos út esetén ilyenkor beljebb kell venni. A buszmegállók között megkötött útvonalon kell haladnia, mindegyiknél pedig meg kell állnia. A szimuláció teljes ideje alatt az útvonalat követi, kezdetben egyik véletlenül kijelölt megállóból indul.

3.2.7. Buszmegálló

Autóbusznak elhelyezett megállási pont. Útkereszteződésen kívül bármely csomópont eljelölhető buszmegállónak. Szükséges információ a buszmegállók közötti minimum távolság, valamint az összes buszmegálló száma. Ezek az adatok paramatérezhetőek is. Köztük az útvonal kiszámítása történhet egy véletlenszerűen választott kezdőponttól a kialakult gráf mohó algoritmussal történő bejárásával.

3.2.8. Épületek

Az utak között kialakult üres területek feltöltésére szolgáló objektumok. Jellemzői közé tartozik az épület négy csúcsa, valamint a magassága. Az épület pozíciójának valamint méreteinek meghatározása a változatosság érdekében véletlenszerű.

3.3. A modell célja

3.3.1. Generálási szempont

Magának a városnak, úthálózatnak a generálására vonatkozóan a következők a célok:

- Viszonylag kevés elemből álljon össze a generált város
- Ne legyen életszerűtlen az összeállított úthálózat
- Tartalmazza a felvetett közlekedési helyzetek szimulálására szükséges elemeket
- Vizuálisan emlékeztessen egy átlagos városra
- Bizonyos mértékig paraméterezhető legyen, főleg a méreteket érintve

3.3.2. Szimulációs szempont

A már legenerált úthálózaton a szimulációra tekintve ezek a célok:

- A járművek kövessék a modellben definiált alapvető szabályokat
- A szimuláció bizonyos mértékben paraméterezhető legyen, főleg a járművek számát illetően
- Reagáljanak egymásra a járművek
- Ne legyen életszerűtlen a járművek viselkedése
- Teljesítmény szempontjából elfogadható legyen egy adott méretig
- Felhasználható statisztikai eredményeket le lehessen menteni

4. fejezet

Generálás

- Térkép, úthálózat és egyéb elemek generálása - Alapelemek megjelenítése HTML Canvas-en. - Elsősorban gráfokról van szó.

4.1. Úthálózat Generálása

Maga a generáló algoritmus megalkotásakor első lépésben a nagyvonalakban leírt elvárt működést vizsgáltam. A hálózatnak középponttól kifelé haladva ritkulnia kell, ezzel közelítve egy valódi várost. A paraméterezéssel kapcsolatos elvárásokat könnyen meg lehet valósítani. Egy gráfként képzelhető el a megalkotott úthálózat, melynek csomópontjai jelölik az egyes útelemek elejét, élei pedig azoknak tartományát. A gráf 0. szintjét jelöltem el a hálózat középpontjának, innen mindig 4 irányba indulnak élek. Minden további csomópont generálódásakor kapja meg annak értékét, hogy mennyi irányba indulnak belőle élek. A csomópontok generálása a következőképpen történik:

1. A kiválasztott csomópont kap egy véletlenszerű irányt, észak, dél, kelet, vagy nyugat értékében
2. Ha a kapott érték megegyezik azzal az iránnyal amerre a csomópont szülője van, vagy már indult arra belőle él, újat kap
3. A kiválasztott iránynak megfelelő véletlen generált X és Y koordinátákat kap
4. A kapott koordinátákból alkotott csomópont felkerül a gráfra
5. Ha a jelenlegi csomópontból még kell éleket indítani, akkor kezdődik előről

Az előbbiekben említett X és Y koordinátát kissé pontosítanám. Ha a csomópontot nyugat vagy kelet irányba generáljuk, az X koordináta értéke a jelenlegi pont X koordinátája, hozzáadva egy előre definiált értékek közötti véletlen szám. Az Y koordináta ilyenkor egy minimális eltérés az út fekvésében, hasonlóan generálódik, de lényegesen alacsonabb véletlen számot kap. Ez tulajdonképpen a valóságban is látható "tökéletlenségekre" vezethető vissza, ugyanis a legtöbb esetben ott sem teljesen merőleges egymásra kettő út. Az észak és dél irányba induló pontok esetén ugyan ez az eljárás, csak a két koordináta szerepe felcserélődik. Fontos viszont az is, hogy a középponttól távolodva átlagosan kevesebb elágazása legyen egy csomópontnak, ezért ehhez felhasználható annak gráfbeli szintje. Minden csomópontról eltárolódik annak szintje (szülő

szintje + 1), ami befolyásolja az elágazások számát. Egy másik probléma az, hogy az így generált élek gyakran metszhetik egymást, amely az aluljárók és a hídak hiányában itt nem megengedett, ezért utolsó lépésnek ellenőrizni kell hogy az új él metszi-e bármely meglévő élek közül akár az egyiket is, és ha igen akkor nem kerülhet fel a gráfra. Az él viszont hozzáadható az eredetileg kijelölt csomóponthoz legközelebbi csomóponttal történő helyettesítéssel, feltéve ha így nem történik létező út metszése. A kanyarok beépítése megoldható azzal, ha az olyan csomópontoknál ahol közel merőlegesen találkozik két él, egy körívet húzunk. A buszmegállók elhelyezésével kapcsolatban a következő feltételeket adtam:

- A legcélszerűbb a hálózat két, egymástól távol lévő, a gráf mélyebb szintjein elhelyezkedő pont közötti út létrehozása
- Az útvonalnak érintenie kell a középpontot, a gráf 0. szintjét
- Lehető legkevesebbszer érintse többször ugyan azt a csomópontot
- Ne legyen két megálló két egymás melletti csomóponton

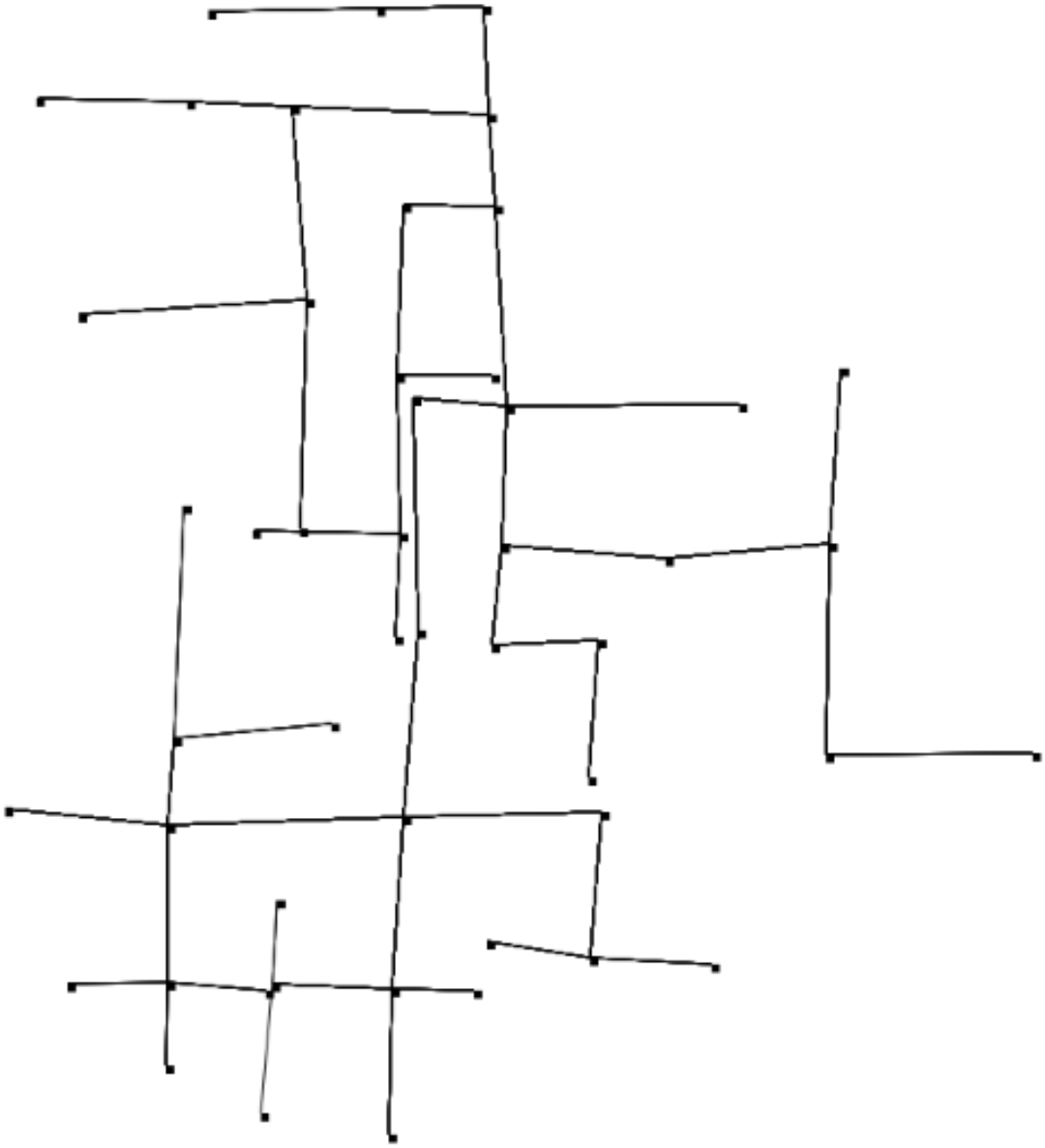
Ennek érdekében először kijelölöm az út egyik végét, mint megállót. Ezek után keresek egy utat a gráf 0. szintjéhez, melyen legfeljebb minden második csomópontot kijelölöm megállónak. A középpont elérése után kijelölöm a második végpontot, mely a középponttól az eddigi iránynak ellentétesen, a gráf mélyebb szintjei közül kell lennie. Az ehhez a középpontból vezető úton, az utóbbihoz hasonló eljárással kijelölök megállókat. Az utak sávszámának meghatározására a gráfbeli mélységüket használom. A magasabb szinten lévő élek nagyobb valószínűséggel lesznek többsávosak. Ez a generálás végén választódik ki. Az egyirányú utak kijelölése is a folyamat legvégén történik. Ennek az oka magából az utak jellegéből adódik, ugyanis egyirányú út nem végződhet zsákutcában. Éppen ezért egy olyan részgráfot kell találni amely Hamilton-kört alkot, amelyen egy részgráfot már nyugodtan egyirányúvá lehet tenni.

4.2. Alapelemek megjelenítése HTML Canvas segítségével

Az algoritmus működését, eredményét egy HTML Canvas objektumon szemléltetem. Magát a kódot JavaScript-ben írtam, annak ECMAScript 6-os verziójában. Fejlesztői környezetnek A JetBrains által készített WebStormot használtam. Először is létrehoztam egy HTML fájlt, ahol a body tag-en belül elhelyeztem a canvas elemet. Erre az elemre JavaScriptből a HTML-ben megadott id-je alapján lehet hivatkozni. Ezután létrehoztam egy JavaScript fájlt, melyben egy (document).ready() szintaktikában elhelyezett függvényhívással indítom a generálást. A (document).ready() a jQuery függvénykönyvtárnak része. Azért szükséges ebben az esetben, mert ameddig a html fájl nem töltött be teljesen, a script nem futtatható mert a canvas elemre nem létezne semmilyen referencia. Ez a részlet biztosítja hogy csak akkor kezdődjön a script futtatása, ha a HTML documentum teljesen betöltött. Ha ez megtörtént, meghívódik a DrawOnCanvas függvény. Először is ez eltárolja változóként a canvas-ra vonatkozó referenciát. szintén új változóba, utána a myCanvas.getContext("2d") létrehoz egy CanvasRenderingContext2D objektumot, mellyel a canvasra rajzolni lehet. A következő lépésben

megtörténik a paraméterek beállítása, azaz mennyi legyen a maximális csomópontok száma, átlagosan milyen hosszúak legyenek az utak. A kontextust elmozdítjuk alapértelmezett helyéről, (0,0)-ról az (1000,500) pontba. Létrehozzuk a kezdő csomópontot ezen a pozíción, és hozzáadjuk a csomópontokat tartalmazó tömbhöz. Létrehozzuk az éleket tartalmazó tömböt, majd kirajzoljuk az első csomópontot, melyet egy négyzettel ábrázolok a kontextus .fillRect() függvényével. A többi csomópont generálásához itt egy ciklust indítok, mely addig fog futni ameddig a csomópontokat tartalmazó tömb elemszáma el nem éri a paraméterként megadott értéket. Minden iterációban a csomópontok tömbjéből a következő elemet választja ki első lépésben, a 0-tól indulva. A kontextust áthelyezem a jelenlegi csomópont pozíciójába, majd beállítok négy logikai változót, melyek megadják hogy ebből a csomópontból a négy irány közül valamelyikébe mentünk-e már tovább. Kezdetben mindegyik hamis értéket kap. Ezt követően egy switch utasítással megkeresem melyik az az irány ahonnan az adott csomópontot kiterjesztettük, és ennek megfelelően arra nem mehet tovább. Akár úgy is lehet venni hogy a jelenlegi pontból arrafelé már "húztunk élet", a logikai változó értéke erre vonatkozóan hamissá válik. Különleges eset a kiindulási pont viszont, ahol egyik irányra sem igaz ez, ezért csak szimplán továbbmegy a program. Ezután következik egy újabb ciklus, mely a csomópontból húzandó éleken iterál végig, tehát a kiindulási pontnál 4 alkalommal fog ismétlődni. Legelőször is belép egy végtelen ciklusba, ahonnan csak akkor szabadulhat break utasítással, ha a ciklus elején véletlenszerűen kisorsolt irányba húzható él. Ha nem akkor újat sorsol. Ezután egy switch szerkezet következik a direction nevű változóra, mely az előbb kapott irányt tárolja. Észak és dél esetén az új csomópont X koordinátája egy előre definiált (minimális) értékek közötti véletlen szám, melyhez hozzáadjuk a jelenlegi pont X koordinátáját. Ez a szám negatív is lehet. Az Y koordináta kiszámításánál is hasonló a helyzet, de itt nagyobb számokat várunk, amelyek csak pozitívak lehetnek. Ha észak felé terjeszkedünk, a kapott véletlen számot beszorozzuk -1-el, ehhez hozzáadva a kiindulási Y értéket, megkapjuk az új pont koordinátáját. Nyugat és Kelet esetében annyi a különbség, hogy a két koordináta generálásában az eljárás felcserélt. Ezek után gráf mélységének függvényében generálódik egy szám, amely megadja hogy az új csomópont hányfelé fog szétágazni. Annak az esélye hogy négyfelé ágazik kezdetben 100Ha egy csomópont nem teljes, azaz 4-felé ágazó, akkor véletlenszerűen adódik hogy 2, vagy 3 ágú. Ez után beállítunk egy invalidedge nevű logikai változó hamis értékre, és elkezdjük végigiterálni egy újabb ciklusban az eddig létező élek tömbjét. Ha még nem léteznek élek, azaz ez az első él generálása, ezt a részt átugorja. Ellenkező esetben minden élre meghívódik egy külön függvény intersects néven. Ennek a függvénynek argumentumaiként meg kell adni mindkét él két végpontjának az x és y koordinátáit, azaz összesen 8 argumentumot. Ez a függvény igaz értéket ad vissza ha a két szakasznak van közös pontja. Működésének alapja az ezen pontok által kapott két egyenes feltételezett közös pontjára vonatkozó egyenletrendszer. Ezen egyenletrendszer mátrixának a determinánsát nézi meg elsősorban. Ha ez 0, nem létezik a metszéspont. Ha nem 0, akkor tudnunk kell hogy a metszéspont a szakaszokon van-e. Az egyenletből kapott lambda és gamma értékeknek egyaránt 0 és 1 közé kell esnie, ha a pont a két szakasz közé esik. Ha ez teljesül igaz értéket, ha nem akkor hamis értéket ad vissza a függvény. Abban az esetben ha a kapott él metszett más élt, a végpontját áthelyezzük a hozzá legközelebbi csomópontra, viszont ha így is metszi az egyik másik élt, akkor elvetjük. Ha nem metszette semelyik élt, akkor Nodes.push()-al hozzáadjuk a tömbhöz az új pontot, felrajzoljuk a canvas-ra, az Edges tömbhöz szintén hozzáadjuk a hozzá vezető élt, majd a kontextusnak a .lineto() és .stroke() metódusával felrajzoljuk

a canvas-ra. Ezen alap eljárással a következő képhez hasonló eredményhez jutunk:



4.1. ábra. Kép a generáló algoritmus eddigi eredményéről

Ezek után következik az utak sávszámának kiadása, az egyirányú utak elhelyezése, valamint a buszmegállók generálása.

5. fejezet

Tervezés, implementáció

- Bemutatni a használt eszközkészletet. - UML osztálydiagram, szekvencia diagramok, blokkvázlat és hasonló dolgok. - JavaScript implementációval kapcsolatos tudnivalók. - WebGL-ről részletesen írni.

6. fejezet

Szimulációk

- Különféle paraméterezéseket megnézni. - Statisztikai jellegű vizsgálatokat végezni a kapott eredményekre.

7. fejezet

Összegzés

Irodalomjegyzék

- https://www.researchgate.net/publication/228966705_A_Review_of_Traffic_Simulation_Software
- https://www.anylogic.com/road-traffic/
- https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

CD-melléklet tartalma

A dolgozat PDF változatát a `dolgozat.pdf` fájlban találjuk.

A dolgozat LaTeX segítségével készült. A forrásfájlok a `dolgozat` jegyzékben találhatók.

...