Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék

Szakdolgozat

Miskolci Egyetem

Interaktív közlekedési modell procedurális generálása és megjelenítése WebGL segítségével

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 (KYDK4Z) programtervező informatikus.

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

A szakdolgozat címe: Interaktív közlekedési modell procedurális generálása és megjelenítése WebGL segítségével

A feladat részletezése:

A dolgozat célja egy olyan interaktív alkalmazás bemutatása, amely egy város közlekedését képes szimulálni. Ehhez a város úthálózatát, az épületeket és az egyéb környezeti elemeket az elkészült szoftver procedurálisan generálja majd. A generálás során külön paraméterek jellemzik a város szerkezetének és a szimulációnak a komplexitását. Az úthálózat esetében ilyen például, hogy milyen jellegű útkereszteződések fordulhassanak elő, illetve azoknak milyen legyen a gyakorisága. Fontos szempont, hogy a szimuláció realisztikus legyen olyan tekintetben, hogy a közlekedés résztvevői szabályosan közlekedjenek. Ehhez részletesen meg kell adni azok viselkedésének matematikai modelljét, illetve bemutatni azon heurisztikákat, amellyel a szimuláció valószerűvé tehető. A program JavaScript programozási nyelven kerül megvalósításra. A grafikus megjelenítést a WebGL nevű OpenGL alapú megjelenítő környezet biztosítja majd.

Témavezető: Piller Imre (egyetemi tanársegéd)

A feladat kiadásának ideje: 2018. szeptember 28.

. . . . . .
szakfelelõs

Eredetiségi Nyilatkozat

Alulírott **Kása Dániel Zoltán**; Neptun-kód: KYDK4Z a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős programtervező informatikus szakos hallgatója ezennel büntetõjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Interaktív közlekedési modell procedurális generálása és megjelenítése WebGL segítségével* 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

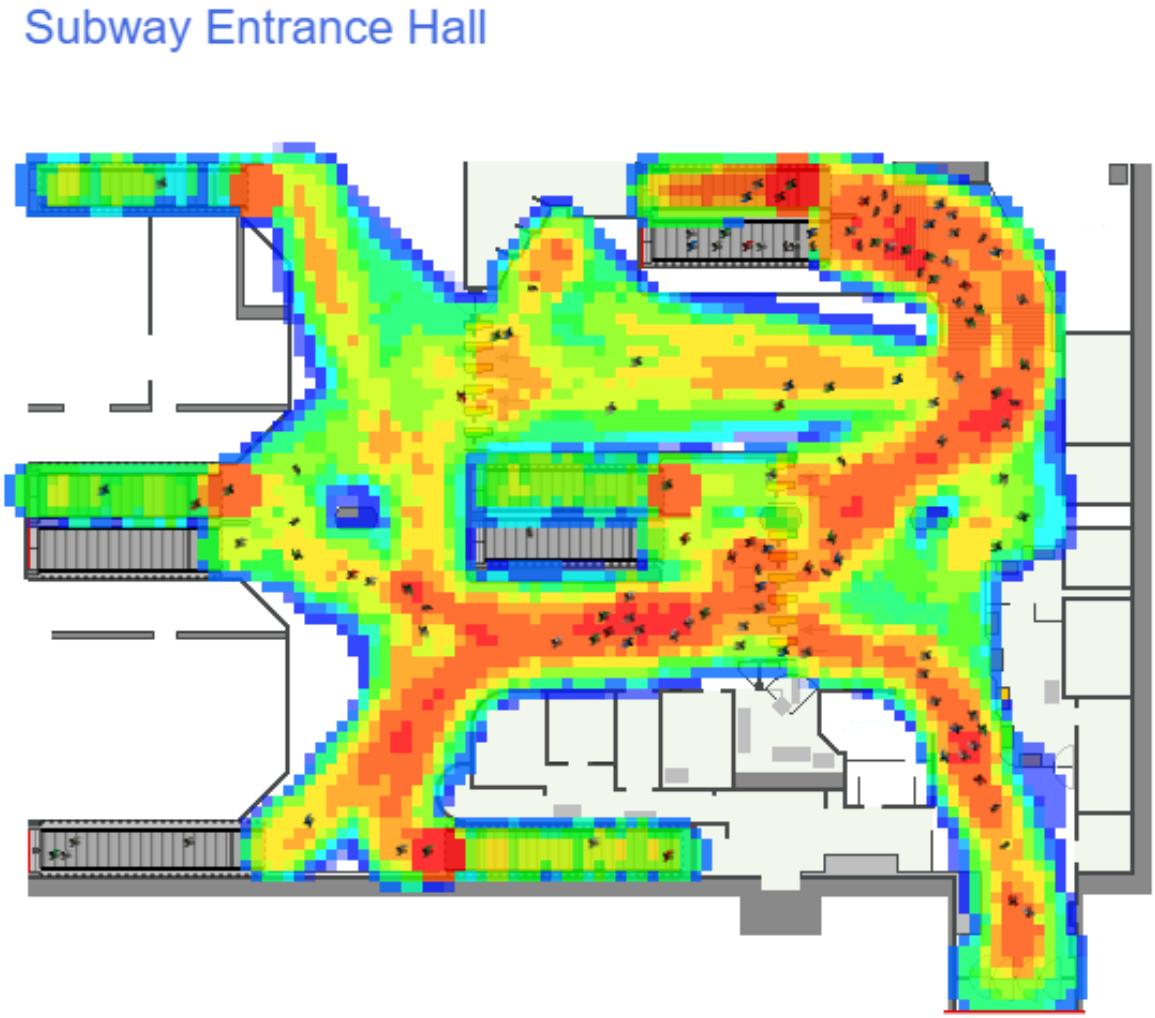
A feladatot azzal kezdtem, hogy utánanéztem milyen közlekedés modellezésére, annak szimulálására készült szoftverek készültek eddig. Ezek felépítése, valamint működése alapján fogom összeállítani a modellt, és továbbá az alapvető elvárásokat a generálást és a szimulációkat illetően.

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 útszakasz, útkanyarulat, útkereszteződés, híd, gyalogátkelőhely, lámpás útkereszteződés, valamint buszmegállók és parkolók. A szoftver képes különböző közlekedési szabályok alkalmazására, valamint a járművek intelligens módon választják meg a haladáshoz szükséges útvonalat. Valósidejű 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.



2.2. ábra. Hőtérkép egy metróállomás gyalogosforgalmáról az AnyLogic szimulációs szoftverrel

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.

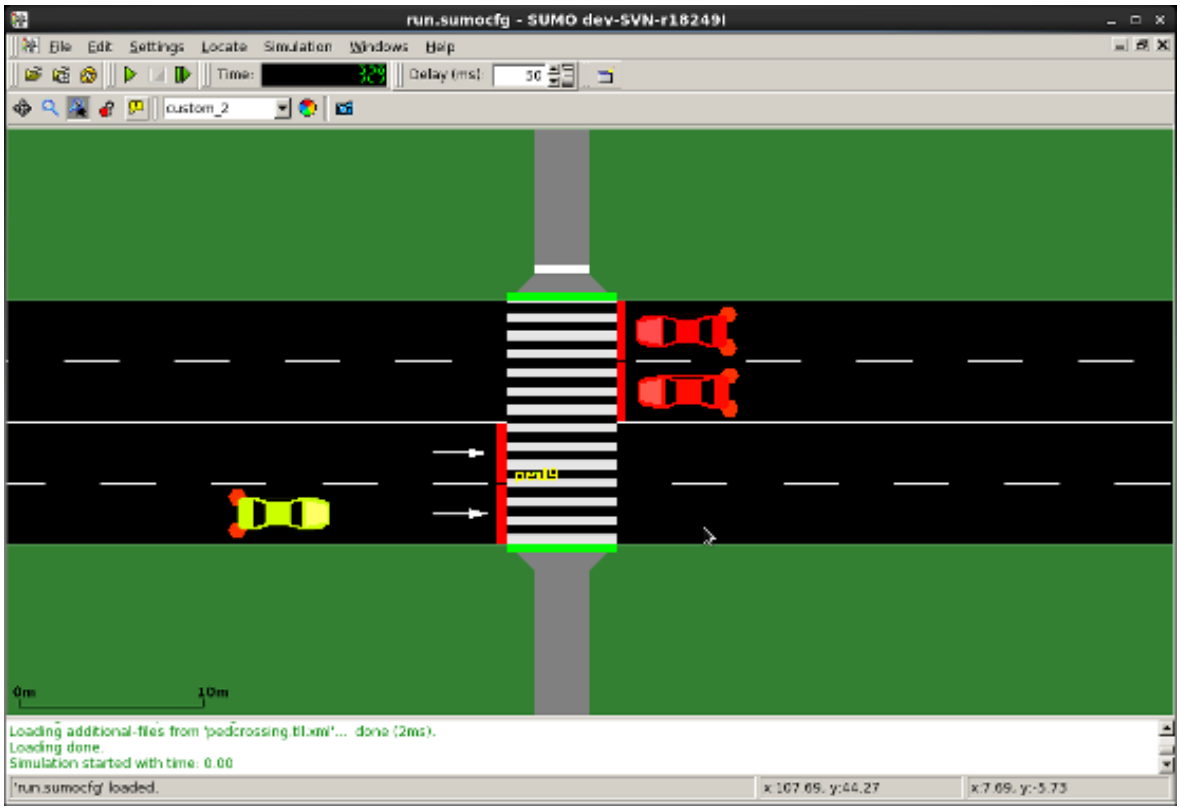
A gyalogos könyvtár segítségével szimulálható a gyalogosforgalom sűrűsége különböző környezetekben. Ilyen környezet lehet például egy buszmegálló, vagy vasútállomás területe. Az egyes épületek felépítése is befolyásolja a forgalom sűrűségét, a gyalogosok minden akadályt, ebbe beleértve egymást is, próbálnak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



2.3. ábra. Kép a SUMO programból

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

A szoftver alkalmas a közlekedési lámpák teljesítményének vizsgálatára, olyan útvonaltervezésre 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

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

Matematikailag a modell egy gráfból áll, melynek csomópontjai jelölik az útkereszteződéseket, vagy két egyenes útszakasz között az összekötő részt. Az élek jelölik magát az útszakaszokat, ezeknek az éleknek pedig számos tulajdonságai lehetnek amivel különbséget teszek több elem között.

3.2.1. Egyenes út

Alapvető egyenes útszakasz, ennek lehet több paramétere is. Ennek az elemnek elsősorban tartalmaznia kell az úton létező sávok számát. Az elem leírásához szükség van annak kezdő és végpontjára. További információt ad a modell többi részének, ha tartalmazza egy megközelítőleges égtájnak megfelelő irányt. Ebből következtetni lehet arra ha például kanyar következik, és ha igen akkor milyen irányban, csak arra van szükség rá hogy megnézzük milyen módon változik az utak hozzávetőleges iránya.

Alapvetően az út olyan szélességű, hogy elfér egymás mellett két jármű, ezért ha több sáv van csak arra kell figyelni hogy megfelelő helyen közlekedjenek az adott járművek. A gráfon való pozicíójának behatárolásához a kezdő és végpont az ő általa összekötött két csomópontot jelenti.

3.2.2. Egyirányú út

Hasonló az egyenes úthoz, viszont egyértelműen meg kell határozni a két végpont közül melyik a kiindulási, melyik a célpont. Hasonlóan az egyenes szakaszhoz, azzal a különbséggel, hogy ez az út kizárólag 1 sávos.

A gráfot tekintve itt a két összekötött csomópont között egy irányított él lesz, melynek kezdő és végpontjaként egyértelműen el kell jelölni a két pontot.

3.2.3. Kanyar

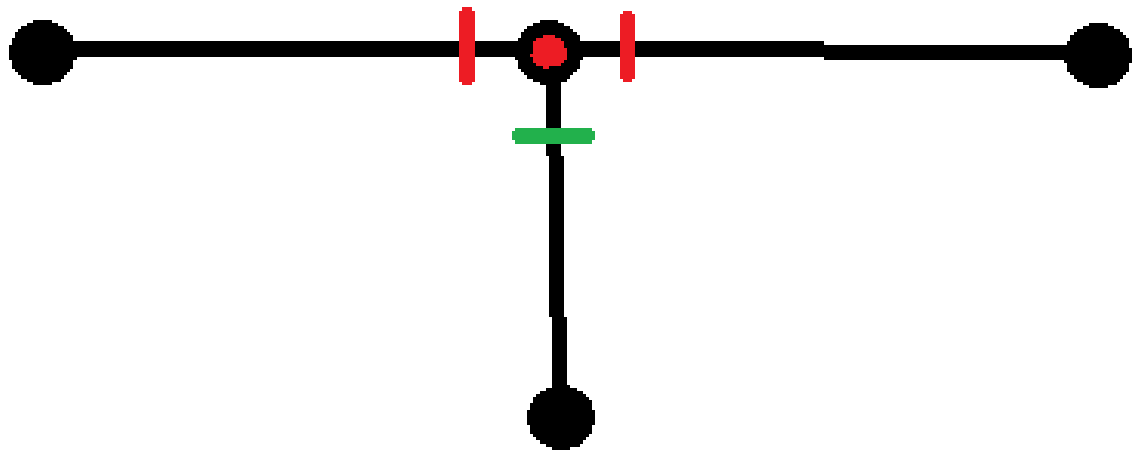
Vehető tulajdonképpen külön elemnek, de lényegében csak két egymást követő egyenes szakasz, melyek az őket összekötő elem mentén nem egyenes irányban folytatódnak. A kanyar iránya adódik az általa összekötött két egyenes útszakasz iránya által.

A gráfon kanyarnak tekinthető az olyan csomópont, melyből csak két él indul ki, ha ez a két él iránya egymástól eltérő, és nem ellentétes.

3.2.4. Útkereszteződés

Három vagy több útszakasz találkozásánál használatos elem. Tulajdonságai közé tartozhat az hogy tartalmaz-e jelzőlámpát az útkereszteződés, vagy sem. Az útkereszteződés jellemezhető annak középpontjával, valamint az azt érintő utak halmazával.

A gráfon ez azt jelenti, hogy amennyiben egy csomópontból legalább 3 él indul ki, az egy útkereszteződés.



3.1. ábra. Háromágú útkereszteződés lámpákkal

3.2.5. Személygépjármű

Legalapvetőbb közlekedési jármű. Az autókról nyilván kell tartani azok jelenlegi pozícióját, mely abból adódik hogy éppen melyik élen haladnak, melyik csomópont felé, és attól milyen távolságra vannak. Minden jármű véletlenszerűen kiválasztott útvonalat

kap, amin végig kell haladjon. Ezek az útvonalak a teljes gráfon egy-egy részgráfok. A jármű jellemzői közé tartozik annak sebessége, valamint a következő csomópontot elérve a várható haladási iránya, tehát előre, balra, vagy jobbra megy-e tovább.

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). Ezen útvonal kijelölése egy véletlenszerűen kiválasztott csomóponttól indul, ahonnan véletlenszerű irányba indulunk el, majd a következő csomópontot elérve szintén új irányt választunk addig, ameddig vagy zsákutcába nem ér az autó, vagy eléri a paraméterként maximálisan kijelölt úthosszat. Szintén a paraméterezést tekintve ilyenkor új jármű jelenik meg ha jelenleg kevesebb van mint a megadott maximális érték.

3.2.6. Autóbusz

Kissé különlegesebb járműtípus, vonatkozik rá néhány további szabály. Többek között elsőbbsége van megállóból elindulva, valamint ha lehet próbál jobbra tartani,többsávos úton használhatja a külső sávot is egyenes haladásra és balra kanyarodásra. A buszmegállók között megkötött útvonalon kell haladnia, mindegyiknél pedig meg kell állnia. Ha elérte az útvonal végét megfordul, és visszafelé halad végig rajta. A szimuláció teljes ideje alatt az útvonalat követi, kezdetben az útvonal végéről elejétől indul.

Ez az útvonal a gráfot tekintve egy legmélyebb szinten elhelyezkedő véletlenszerűen választott csomópont, valamint a tőle legtávolabb lévő csomópont közötti utat jelenti. Kijelölése könnyen megoldható ezen két csomópont szüleit végigkövetve a középen lévő kiindulási pontig.

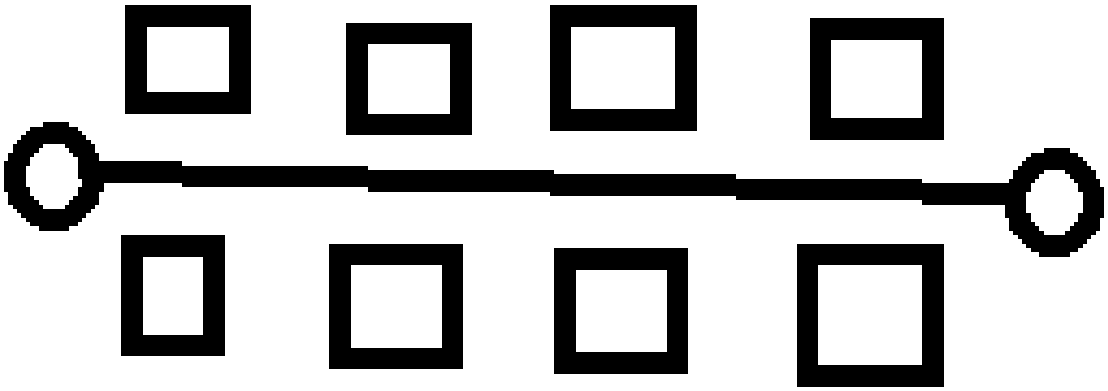
3.2.7. Buszmegálló

Autóbusznak elhelyezett megállási pont. Két csomópont közötti él felezőpontján helyezkedik el, ezért leírásához elegendő az adott élt megadni. Szükséges információ a buszmegállók közötti minimum távolság, azaz hány csomópontnak kell lennie legalább két buszmegálló között. Ezt a számot véletlenszerűen fogom meghatározni minden buszmegálló elhelyezése után újra számítva.

Ezen elem egy olyan csomópont, amelyből nem indul él, és nem is tart belé él.

3.2.8. Épületek

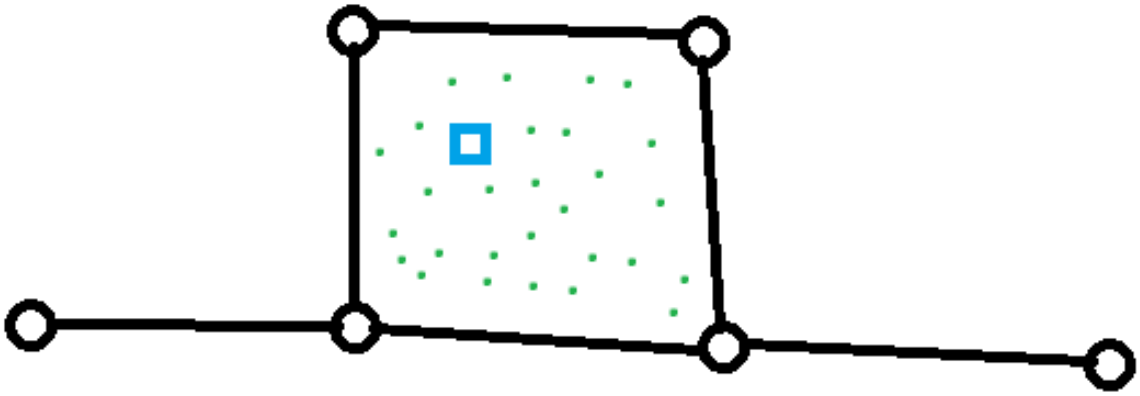
A gráf létrejötte után kijelölendő elemek. Az élek két oldalán helyezkednek el a házak, amelyek az alacsonyabb szinten lévő csomópontok között blokkházak, magasabb szinteknél pedig kertesházak. Az út két oldalán lévő elhelyezkedés szimmetrikus. A blokkok magassága változó.



3.2. ábra. Út két oldalán lévő épületek

3.2.9. Park

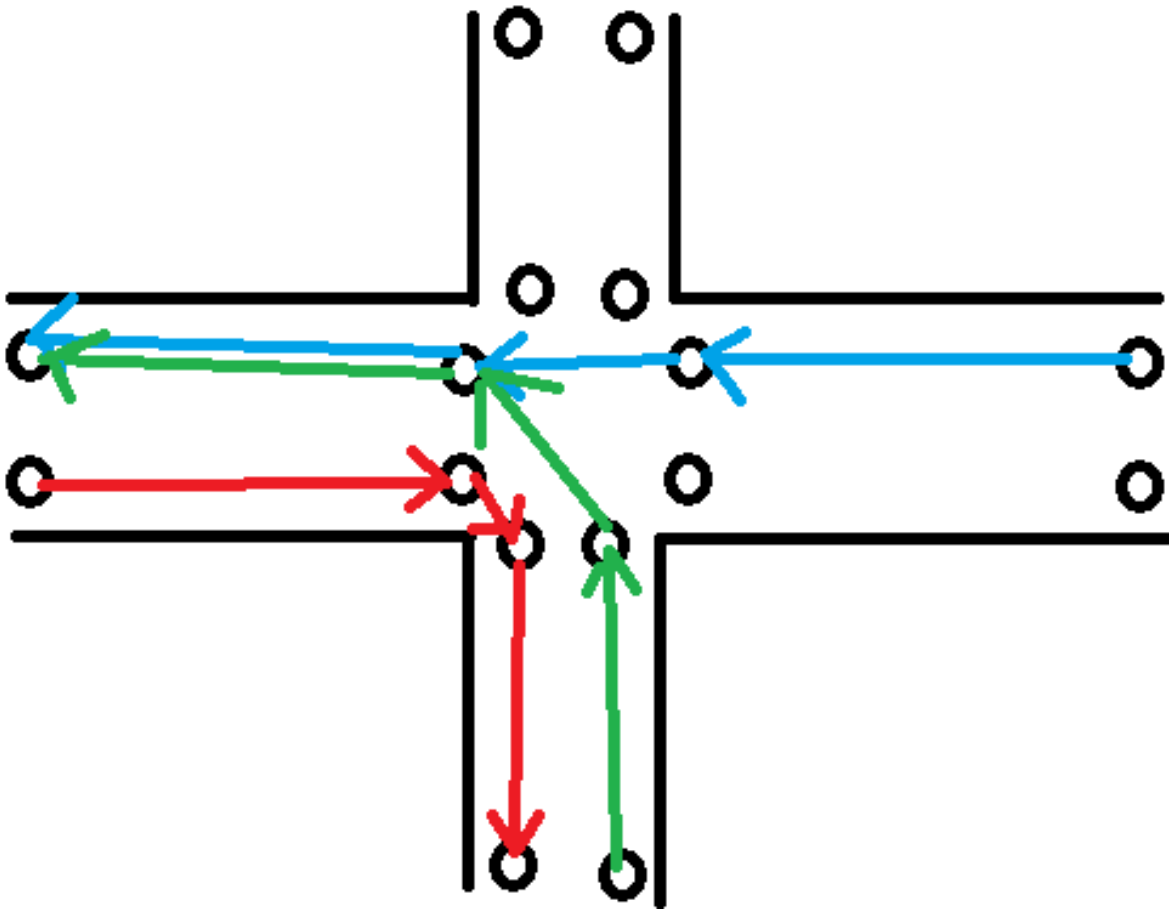
Szintén a gráf létrejötte után kerül elhelyezésre, kizárólag az utakkal teljesen közbezárt területeken. Az adott területen belülre fák kerülnek, valamint egy szökőkút.



3.3. ábra. Elkerített részen lévő park, kék négyzettel jelölve a szökőkutat, zöld pontokkal a fákat

3.2.10. Útvonal

Ez az egy járműhőz tartozó egyedi haladási vonala az úthálózaton. Nem ugyan az, mint az útvonal éleinek kezdő és végpontja, ugyanis itt egyértelműen ki kell jelölni annak a sávnak a pontjait, amelyikben a jármű haladni fog. Útkereszteződésben ez segít a helyes irányba való kanyarodásban.



3.4. ábra. Egy útkereszteződésbeli útvonalakat kijelölő pontok. Példa útvonal balra kanyarodáshoz (zöld), jobbra kanyarodáshoz (piros), egyenes haladáshoz (kék).

3.2.11. Közlekedési szabályok

Néhány alapvető közlekedési szabály amelyet a járműveknek követniük kell, általában attól az elemtől függ ahol haladnak, valamint attól amelyhez tartanak

- Egyirányú útra csakis a kijelölt útirányban hajthatnak rá a járművek
- Ha az út több sávot tartalmaz, a külső sáv csak jobbra kanyarodásra alkalmas, a belső sáv balra kanyarodásra, illetve egyenes haladásra
- Ha az útmenti buszmegállóból elindulni készül a busz, a többi járműnek elsőbbséget kell neki adnia
- Jelzőlámpával ellátott útkereszteződésbe nem hajthat be ha a lámpa piros, ha aközben vált hogy benne van akkor még áthaladhat

3.3. A modell kritériumai

3.3.1. Generálási szempont

Magának az úthálózatnak a generálására vonatkozóan a következők a célok:

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

Elemek száma

Az említett viszonylag kevés elem, vagy pontosabban kevés különbözű típusú elem azt jelentené, hogy ne legyenek túl specifikusak az elemek, hanem akár könnyedén egyik elem bővítésével elő lehessen állítani egy másikat. Ilyen például a modellben az egyenes szakasz, melyből kis változtatással előáll az egyirányú út.

Életszerű úthálózat

Röviden annyit tesz, hogy nem tartalmaz olyan területeket ahol össze van szűkítve kis területen rengeteg út. Tehát legyen minimális hely kihagyva az utak között az épületeknek. Életszerűtlennek tekinthető továbbá az is, ha egy útkereszteződésből rengeteg út indul ki, azaz 5-6 avagy annál több. Ezért a modellben az útkereszteződés maximálisan 4 utat köthet össze.

Közlekedési helyzetek

Legyen benne jelzőlámpával ellátott útkereszteződés, többsávos út, egyirányú út, valamint buszmegálló. Ezek az elemek szükségesek minden esetben, hogy a szimuláción meg lehessen jeleníteni a lámpák működését, a többsávos utak eseten a járművek sávválasztását és tartását, egyirányú út irányának a betartását, és buszmegállónál a busz megállását, neki járó elsőbbség megadását.

Átlagos város képe

Átlagos városra hasonlít akkor, ha a központtól kifelé haladva ritkul az úthálózat, valamint a többsávos utak az előreláthatólag forgalmasabb helyeken vannak elhelyezve. Ennek érdekében a modellben elhelyezett csomópontok átlagosan kevesebb élt indítanak magukból, minél távolabb vannak a gráf kiindulási pontjától, azaz a középponttól.

Paraméterezés

Ki lehessen jelölni hogy a generálás meddig tartson, azaz hány csomópontot terjesszen ki.

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

Szabályok

A már előbbiekben leírt szabályoknak megfelelően közlekedjenek a járművek, haladási útukat azok alapján válasszák meg.

Paraméterezés

Meg lehessen adni mennyi jármű legyen maximálisan egyidőben az úthálózaton, külön értve a buszokat ettől. Továbbá az is megadható legyen milyen gyakorisággal jelenik meg személygépjármű, és mekkora lehessen a neki kijelölt útvonal maximális hossza csomópontokban mérve.

Reagálás

Ha egy jármű közelít egy másik felé kezdjen lassítani. Ez a lassítás függjön a távolság mértékétől, ha elég közel kerül hozzá teljesen álljon meg. Csak akkor induljon el egy jármű, ha haladási útját nem állja el másik jármű. Közúti baleset, azaz ütközés esetén a két érintett jármű kis idő után tűnjön el, ezzel szemléltetve az elszámolás, elvontatás folyamatát.

Életszerű viselkedés

A járművek tartsák az út vonalát, ne térjenek át a szembe sávba. Gyorsítás és lassítás viszonylag realisztikusan történjen, ne pedig hirtelen. Kanyarodás előtt igyekezzenek lelassítani, kanyart pedig végig lassan bevenni.

Teljesítmény

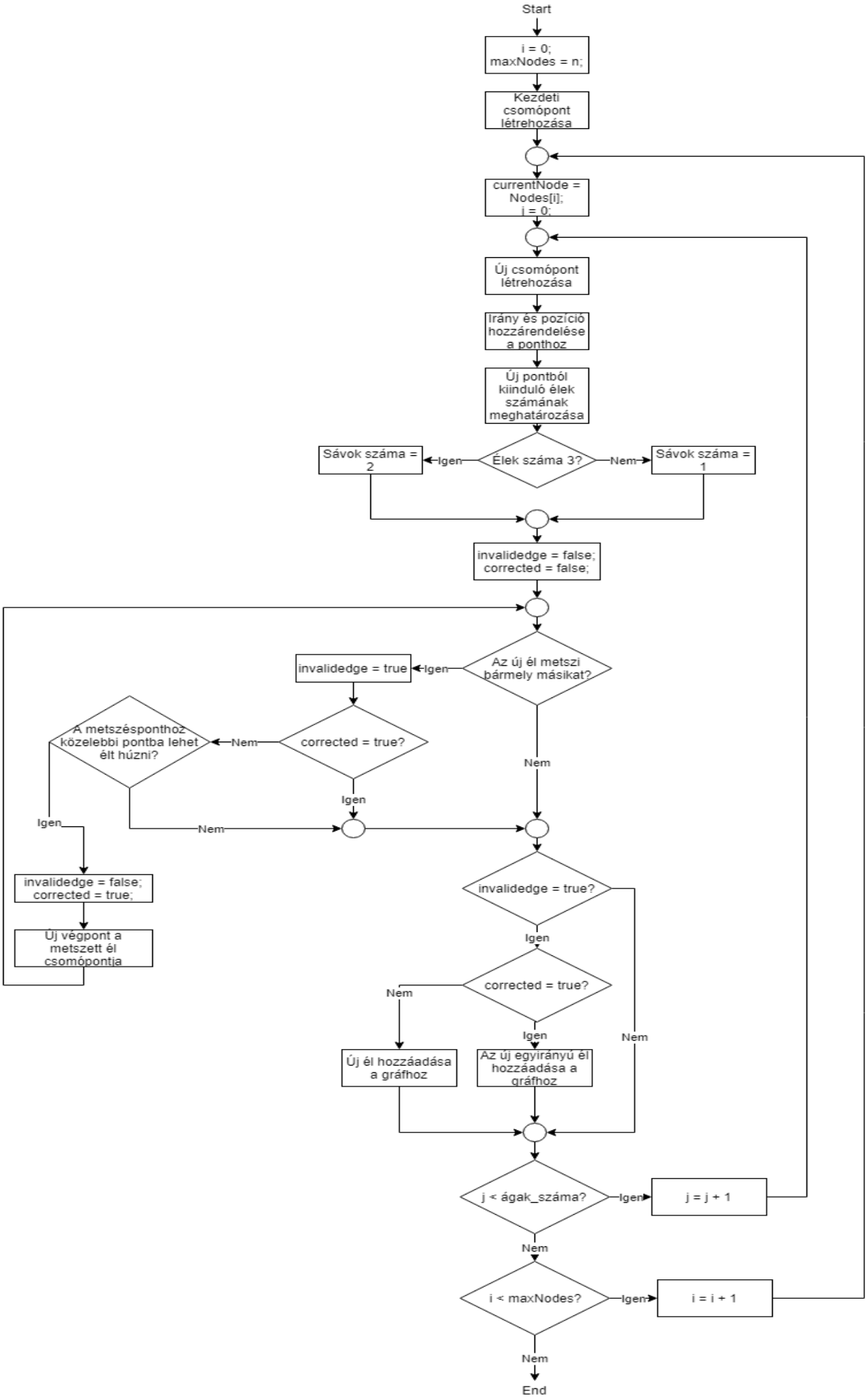
A szimuláció mérete mind az úthálózat csomópontjait, mind az egyszerre megjelenő autók számát értem. Abban az esetben ha ezek az értékek nem kimagaslóan nagyok, a szimulációnak lényegesebb teljesítményromlás nélkül kell futnia.

4. fejezet

Generálás

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. Mivel az egy csomópontból kiinduló maximális élek számának 4-et választottam, ez garantálja hogy a középpontnak kijelölt ponttól minden irányba nagyjából egyenletes mértékben terjedjen az úthálózat. Minden további csomópont generálódásakor kapja meg annak értékét, hogy mennyi irányba indulnak belőle élek.



4.1. ábra. A generálás lépéseinek egyszerűsített folyamatábrája

4.1.1. Generáló algoritmus

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 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

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, és nem is zárhatja el a hálózat egy részét a többi elől. Éppen ezért egy olyan részgráfot kell találni amely Hamilton-kört alkot, amelyen egy részgráfot már nyugodtan egyirányúvá lehet tenni.

4.1.2. Az egyirányú út

Egy könnyű módszer annak megoldására, hogy mely utak lehetnek egyirányúak az lenne, ha a generálás során egymást metszett utakból kialakult éleket jelölöm el egyirányúnak. Ezek az élek ugyanis biztos hogy egy létező Hamilton-út részei, mivel az eredetileg fagráfnak megfelelő úthálózat bármely két csomópontja közé húzott él Hamilton-utat ad.

4.2. Alapelemek megjelenítése HTML Canvas segítségével

4.2.1. Fejlesztői környezet felállítása

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. A js fájlban globálisan eltárolom egy változóba a canvasre mutató referenciát, majd egy másik váltózóba lekérem ennek a kontextusát.

4.2.2. Létrehozott osztályok

Node

A gráfon egy csomópontot leíró objektum. Egy konstruktort tartalmaz, amely beállítja az osztály 6 adattagjának értéket. Ezek az adattagok a következőek:

- x: A csomópont X koordinátája
- y: A csomópont Y koordinátája
- branches: Hányfelé kell tovább ágaztatni a csomópontot
- level: A gráf hanyadik szintjén helyezkedik el a csomópont
- connectedFrom: Milyen irányból lett kiterjesztve a csomópont
- parentNodeIndex: A gráfban lévő csomópontokat tartalmazó vektoron belül ezen csomópont szülőjének indexe

Edge

A gráfon egy él leírására szolgáló objektum. Egy konstruktort tartalmaz, amely inicializálja 4 adattagját. Ezek az alábbiak:

- from: Az a csomópont, ahonnan az él kezdődik
- to: Az a csomópont, amiben az él végződik
- lanes: Az él által jelölt úton hány sáv van
- oneway: Logikai változó, az út egyirányú-e vagy sem

Graph

Magát a gráfot leíró objektum, mely tartalmaz 2 függvényt annak canvas-on történő kirajzolására. Paraméter nélküli konstruktora 2 adattagját inicializálja üres értékkel. Adattagjai és függvényei a következők:

- Nodes: A csomópontokat tartalmazó vektor
- Edges: Az éleket tartalmazó vektor
- drawAllNodes: Függvény, mely végig iterál a Nodes vektoron, és mindegyik elemére meghívja a kirajzoló függvényt
- drawAllEdges: Függvény, végig iterál az Edges vektoron, egyirányú út esetén nyilat rajzol, ellenkező esetben egyenes vonalat

4.2.3. Segédfüggvények

intersects

Ez a függvény arra szolgál, hogy kiszámítsa két él metszi-e egymást. Ehhez két paramétert vár, edge1 és edge2 néven. Visszatérési értéke logikai típusú. Működésében a két szakaszból egyeneseket képez, majd ezek alapján az X = v1 + lambda*d1, X = v2 + gamma *d2 egyenletrendszer mátrixának kiszámolja a determinánsát. Ha ez 0, a két szakasz nem metszi egymást. Ellenkező esetben megnézi az egyenletrendszerben szereplő gamma és lambda értékeket, azaz a két egyenes mentén az irányvektor hányszorosát kell venni hogy elérkezzünk a ponthoz. Ha ez nem 0 és 1 közé esik, a metszéspont nincs a szakaszon belül.

```
function intersects(edge1,edge2) {
    let det, gamma, lambda;
    det = (edge1.to.x - edge1.from.x) * (edge2.to.y - edge2.from.y) -
    (edge2.to.x - edge2.from.x) * (edge1.to.y - edge1.from.y);
    if (det === 0) {
        return false;
    } else {
        lambda = ((edge2.to.y - edge2.from.y) * (edge2.to.x - edge1.from.x)
         + (edge2.from.x - edge2.to.x) * (edge2.to.y - edge1.from.y)) / det;
        gamma = ((edge1.from.y - edge1.to.y) * (edge2.to.x - edge1.from.x)
```

```
            + (edge1.to.x − edge1.from.x) * (edge2.to.y − edge1.from.y)) / det;
        return (0 < lambda && lambda < 1) && (0 < gamma && gamma < 1);
    }
}
```

distance

Két csomópont közötti távolság kiszámítására szolgáló függvény. Ehhez a két vektor által meghatározott szakasz hosszát számolja ki pitagorasz tétellel. Ezzel az eredménnyel tér vissza.

maxBranchesReached

Ezzel a függvénnyel megmondható hogy egy adott csomópontba már lett-e 4 él húzva. A függvény paramétereiként megkapja a keresett csomópontot, a gráfot, és a jelenleg kiterjesztés alatt álló csomópont gráfbeli indexét. Amennyiben a gráfban megtalált csomópont branches értéke kisebb mint 3, akkor növeli 1-el és hamis értékkel tér vissza, azaz még nem volt 4 él húzva. Ellenkező esetben igaz értéket ad.

getBranchCount

Ez a függvény egy egész számot kap paraméterként, mely egy csomópont gráfbeli mélységére utal. Ez alapján a szám szerint visszaadja hogy mennyi irányba ágazzon el az adott csomópont. Első szinten 80% az esélye hogy 4 irányban ágazik el, ez szintenként 20%-al csökken. Ha nem 4 irányban ágazik el, véletlenszerűen generált számot ad vissza 0 és 2 között, mivel a csomópont szülőjéhez vezető élt ilyenkor nem számoljuk.

findMaxLevelNodes

A gráf legmélyebb szintjén lévő csomópontok megtalálására szolgáló függvény. Paraméterként megkapja a gráfot. Első lépésben beállítja a max értéket a gráf első csomópontjának szintjére, és inicializálja a maximális mélységben elhelyezkedő csomópontok vektorát. Ezután maximum kereséssel beállítja max értéket. Majd a gráf Nodes vektorán végig iterálva hozzáadja az ezen max értéknek megfelelő mélységű csomópontokat az eredmény vektorhoz. Végül ezzel a vektorral tér vissza.

DrawBStops

Buszmegálló rajzolására használt függvény. Egy csomópontot kap paraméterként. A kontextust felhasználva felrajzolja a canvasra egy zöld négyzet formájában. Visszatérési értéke nincs.

makeBusStops

Ez a függvény szolgál a buszmegállók elhelyezésére. Egyelőre csak a hozzávetőleges helyüket jelöli, nem az él közepére teszi a helyüket, hanem a csomópontokra. Paraméterként megkapja a gráfot. Először meghívja a findMaxLevelNodes függvényt, ebből a kapott értéket eltárolja egy változóba.

Kijelöl az útvonal kezdetének az előbb kapott vektorból egy véletlenszerű elemet. Ezt felrajzolja a DrawBStops függvénnyel. Ezután végig iterál a maximum mélységű tömbökön, maximum keresést végezve a kiindulási ponttól mért távolságot illetően. Ehhez a distance függvényt használja. Ha megvan a végpont azt is felrajzolja. Ezután elindul a kiindulási ponttól, sorra véve a csomópontok szüleit. Minden buszmegálló kirajzolása után kap egy véletlen értéket, amely megmondja hogy hány csomópont után kell rajzolnia mégegyet. Ha eljutott a gráf kezdőpontjáig megcsinálja ugyan ezt az út végpontjától is. Visszatérési értéke nincs.

drawCanvasNode

Egy paraméterként megkapott csomópont koordinátáinak megfelelő helyre négyzetet rajzol a canvasre.

drawCanvasEdge

Egy élt kap meg paraméterként, melynek kezdő és végpontjai között egyenes vonalat húz a canvasre.

4.2.4. A generáló függvény

A generálás kódjának fő része ebben a függvényben van. Ez egy paraméter nélküli függvény, visszatérési érték nélkül. A következőkben vázolom a működését.

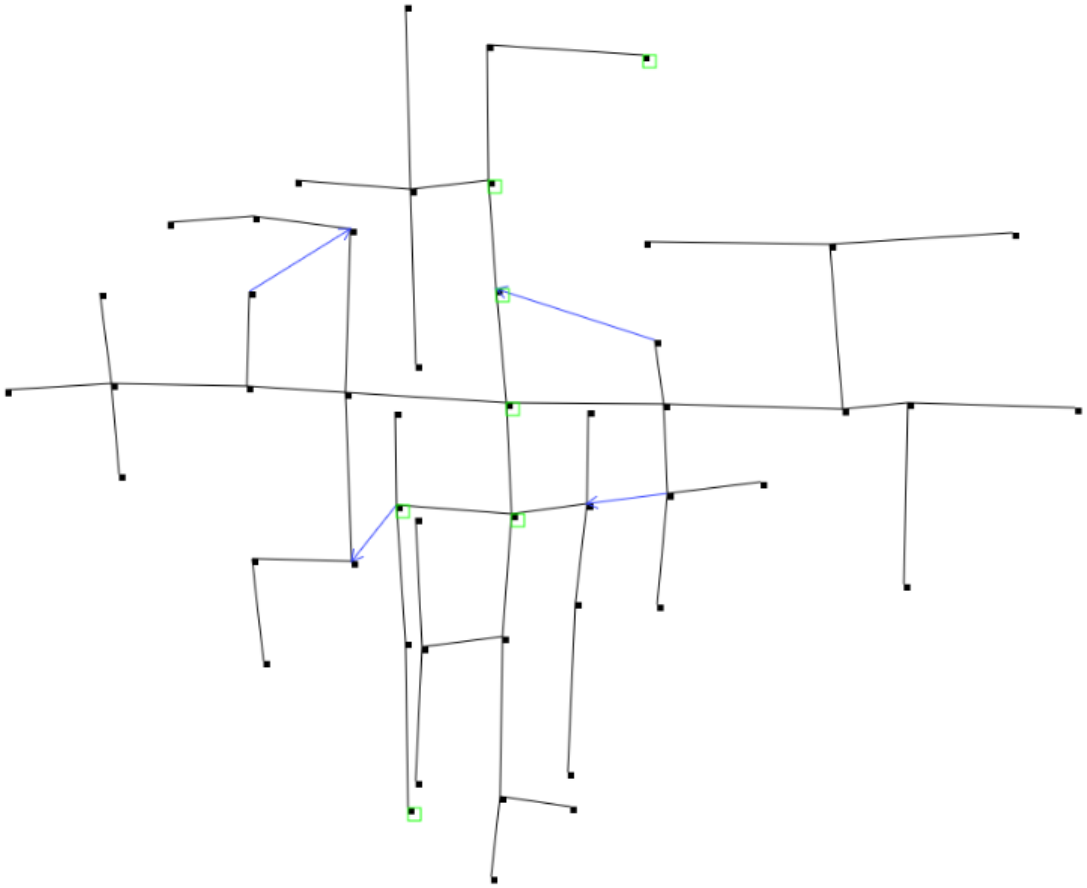
Először inicializálja a gráfot az osztály példányosításával, majd kijelöli a kezdő csomópontot, annak paramétereit beállítva fix értékekre, tehát 4 elágazású lesz, az 1000,500 koordinátákon.

Ezután egy for ciklussal végig iterál a csomópontokat tároló vektoron, minden csomóponton kezdetben megadja a connectedFrom értéket, tehát hogy az adott csomópont melyik irányból lett kiterjesztve.

Ha ez megvan, újabb ciklus kezdődik, mely a csomópont elágazásait terjeszti ki egyesével. Ezen belül megkapja minden iterációban a kiterjesztés irányát, az új csomópont elágazásainak számát a getBranchCount függvénnyel, valamint az új élen lévő sávok száma is kiszámítódik. Ilyenkor már létezik az új él és csomópont mint objektum, következő lépésben megnézi hogy az új él metszi-e bármely meglévő élt. Ez egy szimpla iteráció az éleket tartalmazó vektoron, az intersects függvény segítségével eldönti hogy történt-e metszés.

Amennyiben volt metszés, de az élt be lehetett húzni a metszett élnek a metszésponthoz közelebb álló csomópontjához, az új él megmarad, és egyirányú lesz, a csomópontot pedig elvetjük. Ha nem lehetett behúzni, vagy még így is történt metszés, akkor mindkettőt elvetjük. Ezen vizsgálat után az új él és csomópont amennyiben valamelyik is alkalmas maradt, hozzáadásra kerülnek a gráfbeli vektorhoz.

Miután a kellő mennyiségű csomópont ki lett terjesztve, a függvény meghívja a gráfon belüli drawAllNodes és drawAllEdges függvényeket, majd végül a makeBusStops függvényt. Ezzel kész a generálás, az elemek felkerültek a canvasre.



4.2. ábra. Kép a generáló algoritmus eddigi eredményéről

4.3. Az algoritmus implementálása Unity-ben

Mivel a szimulációt a Unity Enginel fogom elkészíteni, első lépésben az előbbi JavaScript implementációt ki kell bővíteni. Ez főleg a kirajzoló függvények átírását fogja jelenteni, amelyben 2D canvas helyett most már 3D-s megjelenítésben kell gondolkozni, de ezen kívül a C# programnyelvre való áttérés miatt lehetnek még minimális változtatások.

4.3.1. Osztályok

Util

Létrehoztam egy Util nevű statikus osztályt, melynek fő célja az lenne hogy külön osztályba összegyűjti az összes használt matematikai Segédfüggvényt. Ebbe az osztályba került bele a maxBranchesReached, a distance, az intersects mostmár hasIntersected néven, a getBranchCount, valamint a findMaxLevelNodes függvények.

Továbbá bővült egy újab fügvénnyel, a FindDistanceToSegment-el. Ez a függvény paraméterként egy élt és egy csomópontot vár. Visszatérési értékként a csomópont és

a szakasz közötti legrövidebb távolságot adja.

Node

Ez az osztály az eredeti implementációhoz nagy mértékben hasonló. Egy lényeges különbség, hogy a csomópont pozícióját egy Vector3 típusú adattagban tárolom. Ez a típus 3 float típusú koordinátát tárol, az x,y, és z koordinátákat.

Hozzáadtam valamint 5 új adattagot, amelyek közül 4 a csomópont négy sarkát jelőlik. Ezek az adattagok Vector3 típusúak, kiszámításukra az osztályon belül elhelyeztem egy CalculateWorldCorners függvényt, melynek paraméterül meg kell adni a létező csomópont objektum Renderer komponensét.

Az ötödik új adattag egy Edge lista, mely a csomópontból kiinduló éleket tartalmazza, nem beleértve az ide érkezőket.

A CalculateWorldCorners függvény a következő képpen használja fel a renderert: a renderer objektum képes visszaadni a hozzá tartozó 3D mesht behatároló téglatestet. Ennek a téglatestnek lekérdezhető a maximum és minimum értéke, mely rendszerint a jobb felső és a bal alsó sarkát adja vissza. Ezen két pont tudatában kiszámítható a maradék két sarok is, a koordináták vegyes felhasználásával.

Edge

Ez az osztály esett át a legnagyobb átalakításon, ugyanis az utat ábrázoló mesht dinamikusan kell kirajzolni, amelyhez először el kell végezni néhány számítást.

A konstruktora változatlan maradt, viszont az osztály rengeteg adattaggal bővült. Először is a RoadCorners Vector3 típusú tömbbel, amely az útszakasz négy sarkának koordinátáit tárolja. A mesh megalkotásához szükséges a poligont alkotó háromszögek megadása, erre létrehoztam egy egész típusú tri tömböt.

Szintén hasonlóan, az úton jelen lévő felezővonalak, sávelválasztó vonalak jelölésére is szükség van egy objektumra. A felezővonal sarkait a LaneDivider tömbbe, a bal és jobb oldali sávelválasztó sarkait pedig a MultiLaneLeft és MultiLaneRight Vector3 típusú tömbben tárolom. Ezeknek a válaszvonalaknak külön háromszög tömböt is deklaráltam triLane néven.

A meshez szükséges még normálvektorok tartalmazó, illetve uv tömböket is bevezettem.

Az osztály új metódusokkal bővült, ezek közül első a GetWorldCoords void típusú paraméter nélküli metódus. Ez a metódus annak függvényében, hogy a Direction adattagban milyen irány van megadva, kiszámolja az általa összekötött csomópontok sarkai alapján az útszakasz sarkainak koordinátáit.

Példa esetként, ha észak felé tart az út, akkor a kiindulási pont bal és jobb felső, a célpont bal és jobb alsó koordinátáira lesz szükség.

Ha az út két sávos, ezen sarkok alapján kiszámolja a felező- és sávelválasztó vonalak sarkait. Ezek megjelenésével nem kell foglalkozni, csak jelző értékkel rendelkeznek a járművek számára, ezért az útszakasz felett foglalnak helyet.

Ezek után beállítja a tri és triLane értékeit, azaz hogy a háromszögek megalkotásához milyen sorrendbe járja be az adott pontokat.

A GetRoadMesh metódus maga az utat reprezentáló Mesh objektummal tér vissza. Paraméter nélküli metódus. Első lépésben meghívja a GetWorldCoords függvényt, ezzel beállítva az út sarkainak értéket. Majd létrehoz egy új Mesh objektumot, melynek

csúcsait beállítja az előbb kiszámolt pontokra, háromszögek értéket pedig a tri tömbre. A normálvektorok itt nem kapnak nagy jelentőséget, Z tengely szerinti negatív egyenes értéket adtam nekik. Az uv koordinátákat szimplán a mesh négy sarkának jelöltem el.

A GetLaneDividerMesh, GetLeftLaneDivider, valamint a GetRightLaneDivider metódusok az előbbivel hasonló módon funkcionálnak, csak a csúcsok halmaza változik.

Ám ezekkel a vonalakkal kapcsolatban szükség volt létrehoznom három Inverse metódust is, a GetInverseLeftLaneDivider, GetInverseRightLaneDivider, és a GetInverseLaneDividerMesh függvényeket. Erre az a magyarázat, hogy ezek alapvetően síkok, azaz a hátlapjaik nem láthatóak.

Bár a Unity tartalmaz megoldást ütközők konvexé alakítására, ez ebben az esetben nem mindig működik, mert gyakran a válaszvonal négy csúcsai koplanáris pontok. Ezért a következő megoldást választottam: a létrehozott síkokkal megegyező pozícióban létrehozom a síkok ellentétes irányba néző változatát. Ezt a feladatot látják el ezek az inverse metódusok, melyek az adott háromszög tömböt visszafelé járják be.

Graph

A gráfot leíró osztály bővült néhány új adattaggal.

Külön Node és Edge listában tartalmazza mostmár a buszok útvonalát leíró csomópontokat és éleket. A buszmegállok koordinátái tárolására egy Vector3 típusú listát is bevezettem.

Továbbá ebben az osztályban tárolom a buszok számára kirendelt útvonalat is, melyet egy Path nevű osztállyal írok le. Ezt a későbbiekben részletezném a szimulációval kapcsolatban.

Az osztály konstruktora üres listával inicializálja az összes adattagot.

Egy új metódus került az osztályba, a GenerateRandomPath. Ennek egy egész szám típusú paramétert kell megadni, ez lesz az útvonal maximális hossza. A metódus visszatérési értéke Path típusú, ez fogja tartalmazni az útvonal adatait.

Működésében egy Node és egy Edge listában tárolja el az útvonal által bejárt csomópontokat és éleket. Először választ egy véletlenszerű csomópontot, majd a csomópontból kiinduló élek közül választ egyet, és az él túlsó végén lévő csomóponthoz megy tovább. Végül a Node és Edge listát felhasználva visszadja a Path objektumot.

A GenerateBusStops metódusban történt pár változtatás. Ezt is úgy írtam meg, hogy a Path objektumot használja, ezért a metódus az osztályon belüli két busz útvonalat leíró adattag értékét állítja be.

Működésében nagyjából azonos, egy fontos változtatással, miszerint a megállók elhelyezése közben bejárt csomópontokat és éleket eltárolja listában. Az útvonal vége felőli bejárt utat megfordítás után hozzáadom a kiindulás felőli úthoz, ez adja a busz teljes útvonalát.

Ezt az útvonalat megfordítom teljes egészében, így megkapom a buszok útvonalát ellentétes irányban. Mindkettő útvonal tárolódik a végén.

A CalculataActualBranchCount metódusra azért volt szükség, mert generálás után a csomópont branches adattagja nem minden esetben reflektálja a hozzá tartozó tényleges élek számát, például ha az egyik kiterjesztése érvénytelen volt. Ez a metódus megszámolja ténylegesen hány él indul és/vagy érkezik az adott csomópontba, és ez alapján beállítja a branches értékét.

RoadGenerator

Többek között ebbe az osztályba került a fő generáló függvény, valamint a kirajzolásért felelő függvények.

Adattagjai közé tartozik maga a gráf, mint Graph típus, külön Material típus három féle útra (egy sávos, két sávos, egyirányú), négy GameObject lista, melyek az útszakaszok, az útkereszteződések, a választóvonalak, és a buszmegállók tárolására vannak, valamint egy sablon objektum elhelyezésére alkalmas CrossRoad és BusStop GameObject típusú adattag.

Az osztályon belüli Start metódus az ezen osztály komponensként tartalmazó objektum létrejötte utáni legelső képkockában fut le, pontosan egyszer.

Ebben a metódusban először elhelyezem a kezdeti csomópontot, majd ameddig kevesebb csomópont létezik mint a megadott maximum, az ExpandNode metódussal kiterjesztem a soron következő csomópontot.

Ha ez megtörtént, újra számoltatom a csomópontok elágazásainak számát a gráfon belüli CalculataActualBranchCount metódussal, majd meghívom a SpawnCrossroads, spawnRoads, a gráfon belüli GenerateBusStops, majd végül a SpawnBusStops metódusokat a csomópontok, útszakaszok, és a buszmegállók kirajzolásához.

Az ExpandNode metódusba került bele a generáló algoritmus fő része, ez egy visszatérési érték nélküli metódus, mely egy egész számot vár paraméterként. Ez az egész szám annak a csomópontnak az indexét jelöli, amelyet ki fog terjeszteni. Funkcionalitásában nem történt változás.

A SpawnCrossRoads metódus felelős az útkereszteződések kirajzolásáért. Visszatérési érték nélküli, paraméter nélküli metódus. Az Instantiate metódussal létrehozza egy megadott pontban, az eltárolt sablon útkereszteződés egy másolatát. Ezután kiszámolja a sarkait a CalculateWorldCorners metódussal. Végül a jelzőlámpa irányításáért felelős CrossRoadController szkriptnek átadja a csomópontot.

A spawnRoads void visszatérésű, paraméter nélküli függvény dinamikusan képez mesheket az Edge listában tárolt élek adataiból. A materiált beállítja annak függvényében, hogy az út egyirányú, két sávú, vagy egy sávú. A választóvonalak renderer komponensét kikapcsolja, mivel azok megjelenítésére nincs szükség.

A SpawnBusStops metódus void típusú, és paraméter nélküli. Az Instantiate metódus segítségével létrehozza a sablonként eltárolt megálló objektum másolatát a buszmegállók pozíciót tartalmazó listának megfelelő helyeken.

A GenerateANew metódus a felhasználói felület kezelését látja el. A Generate Anew gombra kattintva lefut ez a metódus, mely törli az összes létrehozott objektumot, a gráf tartalmát üríti, majd meghívja a start metódust amellyel egy új úthálózatot generál.

Szintén a UI-hoz kapcsolódó két metódus a setMaxNodes, mely a beírt értékre beállítja a maximálisan generálandó csomópontokat. Valamint a setLightTime, amely átadja a jelzőlámpák szkriptjének a váltáshoz használandó időtartamot.

4.3.2. A Scene létrehozása

Ahhoz hogy a megírt metódusokat használni lehessen, hozzá kell rendelni valamilyen GameObject objektumhoz. Ezeket az objektumokat lehet dinamikusan is létrehozni C# szkripten belül, de minden esetben szükség van legalább egy GameObjectet létrehozni a Unity editorjában, enélkül nem lehet szkriptet futtatni.

A GameObjecthez rendelt szkriptek az objektum létrejöttekor meghívják a Start metódusokat, majd minden egyes képkockafrissítés után az Update metódust.

Az alap scene tartalmaz egy Main Camera objektumot, valamint egy Directional Light fényforrást. Létrehoztam ezek mellé egy üres GameObjectet, melyhez komponensként hozzárendeltem a RoadGenerator forrásfájlját. Ezt az objektumot elneveztem Manager-nek, ezt fogom használni a szimuláció kezelésére, ezen keresztül hozok létre, illetve törlök objektumokat. A Managerhez hozzárendelt szkript a Scene betöltődésekor most már meghívja a Start metódust, mely végrehajtja a generálás lépéseit.

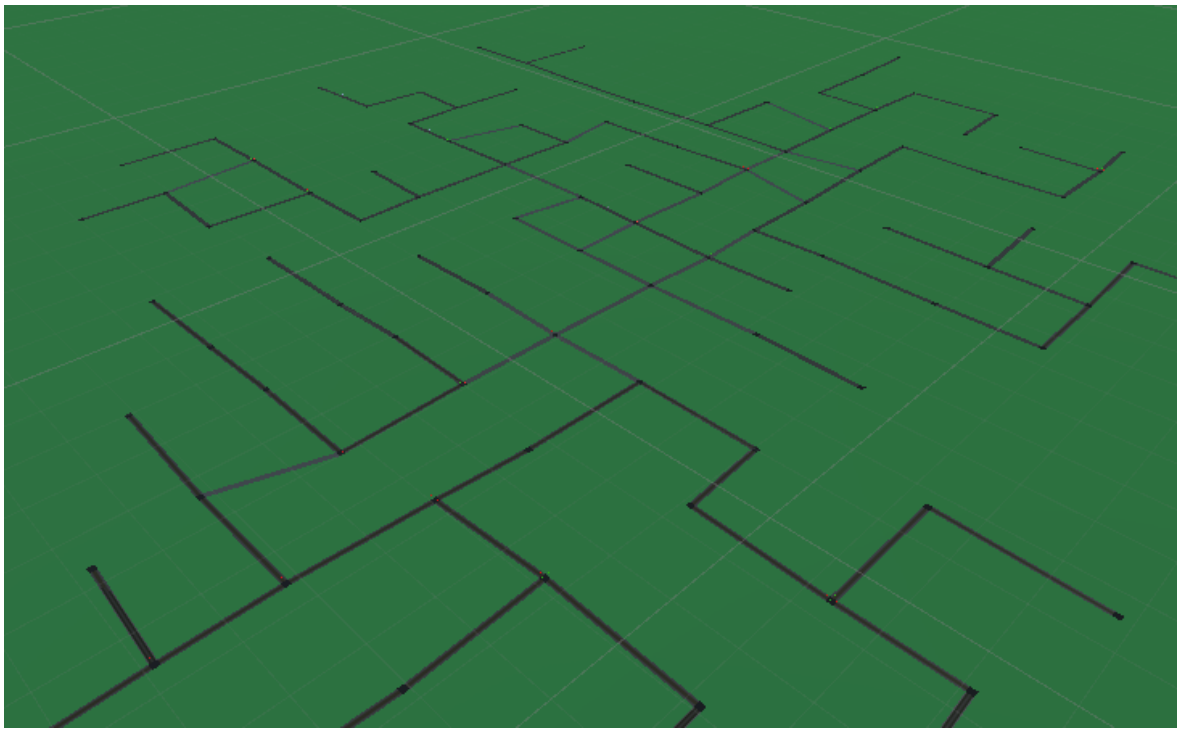
Viszont ehhez még szükség van a szkript adattagjainak feltöltésére. Amint írtam, ez az osztály tartalmaz 3 Material típust, és 2 GameObject típust amelyet az editorból fel kell tölteni.

Először is elkészítettem a materialokat, felhasználva egy alap útburkolat textúrát. Ezt a textúrát kissé átszerkesztve előállítottam a 3 úttípushoz szükséges materialokat, majd hozzárendeltem őket a szkripthez.

Ezek után létrehoztam egy új objektumot a scene-ben, mely egy egyszerű négyzet alakú síklap. Ezen síklapot fogom felhasználni a csomópont jelölésére alkalmas sablonként. Létrehoztam ennek az objektumnak négy gyerekelemét, ezek a négy oldalán lévő jelzőlámpáknak felelnek meg. Ezeket az egyszerűség kedvéért egy alap kapszula alakzattal reprezentálom. Elheyeztem rajta egy sötétszürke, útra emlékeztető materialt. Ezt az objektumot a projekt fájljai közé helyezve létrejön róla egy sablon, ezt már hozzá lehet rendelni a generáló szkripthez.

A buszmegállók sablonját is hasonlóan készítettem el, egy szimpla kapszula alakzattal jelölöm. Ehhez az objektumhoz adtam egy új BoxCollider komponenst, mellyel a megálló által behatárolt területet tudom eljelölni. Ennek a komponensnek az isTrigger tulajdonságát engedélyezve lehet a fizikai motor igénybe vétele nélkül ellenőrizni azt, hogy mely objektumok léptek a területére. Ezzel készen léve hozzáadtam a sablont a szkripthez.

A működést kipróbálva az alábbi eredményhez jutottam:



4.3. ábra. Kép a generáló algoritmus eredményéről Unityben

5. fejezet

Tervezés, implementáció

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

5.1. A WebGL-ről

A WebGL, azaz Web Graphics Library, egy 2D és 3D grafikus renderelésre képes JavaScript alapú API, amely lehetőséget ad interaktív grafika megjelenítésére a webböngészőkben, bármilyen más harmadik féltől származó bővítmény használata nélkül. Képes teljes mértékben hasznát venni a hardveres gyorsításnak, a képfeldolgozási utasításokat a GPU-val hajtatja végre.

Első verzióját 2011 március 3.-án adta ki a Khronos Group, azóta is ők fejlesztik. A Khronos Group emelett még sok más grafikai alkalmazásprogramozási felületet készít, melyek közül a két legismertebb talán az OpenGL és a Vulkan.

A WebGL az OpenGL ES-en alapszik, mely amint a nevéből is adódik, beágyazott rendszerekhez készült. Széles körben használatos főleg mobiltelefonok, tableteken, és más hordozható készülékeken. A WebGL első verziója az OpenGL ES 2.0-án alapult, azóta már kiadták a 2.0-ás verziót, amely az OpenGL ES 3.0-t vette alapul.

Az 1.0-ás verziót már szinte az összes modern webböngésző támogatja, valamint a HTML5 szabványnak is része. A 2.0-ás verziót még sok böngésző csak részlegesen támogatja, néhány pedig, mint például a Microsoft Edge, egyáltalán nem.

A WebGL API nagyon alacsony szintű természete miatt nem célszerű natívan programozni. Az alap alakzatok megjelenítése is rengeteg időt venne igénybe ilyen megvalósítással. Emiatt is készült rengeteg függvénykönyvtár hozzá, amelyek lényegesen megkönnyítik a fejlesztési folyamatot, magasszintű felületet biztosítanak egyszerű alakzatok megjelenítésére, textúra beolvasására, UV map előállításához, projekciós és modelview mátrixok kiszámolásához.

Az egyik ilyen híres függvénykönyvtár a Three.js. A Three.js nagyon bő funkcionalitással rendelkezik, egyszerűen létrehozható vele scene, renderer, kamera. Képes irányított és pontszerű fényforrások kezelésére. Megkönnyíti a materialok használatát, animációk létrehozását, és még sok más hasznos funkciót tartalmaz.

A Three.js-ben először is egy scene-t kell létrehoznunk, ebben az objektumban fog történni a kirajzolás. Ezután jön a renderer létrehozása, majd egy kamera objektum hozzáadása a scenehez. Ha ezek megvannak, a meglévő scenehez hozzá lehet adni a kí-

ván objektumokat, azaz fényforrásokat, irányítást megvalósító objektumokat, valamint mesheket. A mesheket a Three.js-ben lehetőség van JSON objektumokból betölteni, majd dinamikusan hozzájuk rendelni materialt. Az animate függvényben meghívva a renderer kirajzoló függvényét frissül a megjelenés.

5.2. Unity

5.2.1. Unity WebGL

unity webgl implementáció

5.2.2. Unity Editor

az editor működése

5.3. Szimuláció tervezése

5.3.1. Szükséges objektumok

autó, busz objektum felépítése

5.3.2. Szimulációs szkriptek

carwheel, carengine, carspawner, szenzorok, stb

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.

...