SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 397

PROCEDURALNO GENERIRANJE CESTOVNE MREŽE

Antonija Engler

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 397

PROCEDURALNO GENERIRANJE CESTOVNE MREŽE

Antonija Engler

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 397

Pristupnica:	**Antonija Engler (0036524414)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje cestovne mreže**

Opis zadatka:

Proučiti načine zapisa cestovne mreže. Posebice obratiti pažnju na zapise OSM (OpenStreetMap) i ASAM OpenDRIVE format (.xodr). Proučiti mogućnosti korištenje grafova i gramatike za proceduralno generiranje cestovne mreže. Obratiti pažnju na Lindenmayerove L-sustave u ovom kontekstu. Implementirati proceduralnu izradu topologije cesta temeljem proučenih tehnologija. Rezultate ostvariti u zapisu ASAM OpenDRIVE pogodnom za izradu simulacija vožnje, a za prikaz ostvarenih rezultata koristiti CARLA Simulator. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski jezik Python. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 28. lipnja 2024.

Hvala mentorici prof. dr. sc. Željki Mihajlović na strpljenju, predanom vodstvu i potpori kroz preddiplomski i diplomski studij. Hvala roditeljima i sestri na neiscrpnoj podršci kroz cijelo moje školovanje, vi ste me izgradili u osobu koja sam danas. I na kraju, hvala prijateljima na razumijevanju, ohrabrenju i zajedničkim trenutcima koji su mi uljepšali ovo poglavlje života.

Sadržaj

Uvod

Ljudski rod oduvijek teži napretku, inovacijama i usavršavanju ideja, od izuma kotača, preko parnog stroja do električnih i autonomnih vozila. Kako bi to bilo moguće, izumitelji i inženjeri su kroz puno pokušaja i pogrešaka došli do željenih rezultata. Nekada su se testiranja izvodila u laboratorijima ili u posebno predviđenim područjima, no danas postoje simulacije koje smanjuju troškove, osiguravaju sigurnost svih sudionika te omogućavaju ispitivanje scenarija koji su rijetki u stvarnom okruženju [36]. Jedna od takvih simulacija je CARLA. Simulator CARLA je softverska platforma otvorenog koda koja služi za stvaranje, treniranje i testiranje autonomnih vozila [1]. Cilj simulatora je pružiti modularno i prilagodljivo programsko sučelje (engl. *API*) kojeg svatko može uređivati i poboljšavati za potrebe razvoja autonomne vožnje [2]. Zbog toga je CARLA izgrađena na posebno prilagođenoj grani softvera za razvoj računalnih igara Unreal Engine 4 koji je otvorenog koda i dostupan cijeloj zajednici. Unreal Engine prikazuje realističnu scenu i fizikalnu simulaciju, a OpenDRIVE standard opisuje ceste i postavke gradske vožnje, dok se s pomoću programskog jezika Python može upravljati simulacijom, dodavati vozila i pješake te mijenjati vremenske uvjete. CARLA nudi postojeće scene, točnije sedam gradova, i razne modele, kao na primjer modeli vozila, pješaka i prometnih znakova. Svaki element je ručno izradio tim digitalnih umjetnika za potrebe simulatora [1]. Modeli autonomnih vozila treniraju se na cestama ponuđenih gradova, no postavlja se pitanje može li se proširiti ponuda cestovnih mreža i scenarija kako bi se pružili što raznovrsniji i bogatiji podaci? Tu u pomoć uskače proceduralno generiranje. Proceduralno generiranje je postupak u kojem algoritam stvara novi sadržaj uz minimalno uplitanje korisnika (korisnik zadaje samo početne uvjete) i uz predefinirani skup pravila [4]. Može se primijeniti na razna područja, kao na primjer stvaranje terena, cestovne mreže, zgrada ili cijelih gradova. Također, postoje razne metode koje se mogu koristiti za sâm algoritam i pravila, ovisno o primjeni i svrsi projekta, pa je tako šum čest za modeliranje terena, L-sustavi za gradnju mreža ulica te udruživanje primitiva za generiranje zgrada [4].

U ovom radu proučit će se proceduralno generiranje, L-sustavi i tehnologije potrebne za prikazivanje dobivene scene, što uključuje simulator CARLA, standard ASAM OpenDRIVE i programski jezik Python.

1. Teorijska pozadina

Kako bi se bolje razumjela implementacija, prvo je potrebno proučiti i dati pregled o svom korištenom znanju. U ovom poglavlju iznijet će se koncepti o simulatoru CARLA, zapisima cestovnih mreža, L-sustavima i proceduralnom generiranju.

1.1. CARLA

CARLA (Car Learning to Act) je simulator gradske vožnje, izgrađen kako bi pružio potporu izradi, treniranju i detaljnoj analizi rada modela autonomne vožnje [1]. Svaki model u fizičkom inteligentnom vozilu mora imati četiri podsustava: opažanje i modeliranje okoline s pomoću skupa senzora (npr. kamere, LIDAR i radar), lokalizacija vozila i stvaranje mape, planiranje puta odlučivanjem te kontrola pokreta koja upravlja samim vozilom [6]. CARLA ima implementiranu potporu i za percepciju i za kontrolu vozila, nudeći prilagodljivu konfiguraciju seta senzora uz razne signale (npr. GPS koordinate, brzina, ubrzanje, itd.) potrebne za treniranje strategija upravljanja vozilom [1]. Uz to, simulator nudi vlastite modele gradskog okruženja, uključujući modele vozila, pješaka, zgrada, uličnih znakova i okoliša. Scene su podijeljene na sedam gradova od kojih su dva „nepoznata“ jer se koriste za testiranje prilikom natjecanja (engl. *Leaderboard Challenge*) [2]. Unutar svakog grada mogu se naći dva tipa objekata: statični i dinamički [1]. Statički objekti uključuju sve 3D modele koji se ne kreću. To su zgrade, vegetacija, ulični znakovi i gradska infrastruktura. Dinamički objekti su vozila i pješaci. Kako bi se održala realnost i smislenost razvoja autonomnih vozila, svi modeli imaju zajedničko skaliranje i odražavaju veličine stvarnih objekata. Uz navedene objekte, postupak izgradnje scene je sljedeći. Na početku se smjeste ceste i pločnici. Zatim se ručno postavljaju kuće, vegetacija, teren i gradska infrastruktura. Na kraju se određuju lokacije gdje će se pojaviti dinamički objekti prilikom pokretanja simulatora. Jednom izrađena, scena se više ne mijenja, osim u slučaju promjene vremenskih uvjeta. CARLA nudi razne atmosferske uvjete i osvjetljenja, uključujući drukčije doba dana ovisno o položaju i jačini sunca, naoblaku, padaline i skliskost ceste, vjetar i maglu. Kako bi sve to utjecalo na treniranje i testiranje autonomnih vozila, potrebni su senzori koji će prikupiti podatke iz okoline. Trenutno postoje tri vrste senzora: kamere, detektori i ostali senzori [7]. Kamere se dalje mogu podijeliti na dubinske, RGB (engl. *Red Green Blue*, obične svakodnevne kamere), optički tok, semantička segmentacija, segmentacija instanci i

DVS (engl. *Dynamic Vision Sensor*). Kada je vozilo u blizini nekog događaja, aktiviraju se detektori koji mogu prepoznati sudar, prelazak u drugu traku i prepreku. Pod ostale senzore spadaju GNSS (globalni navigacijski satelitski sustav), IMU (engl. *Inertial Measurement Unit*), LIDAR (engl. *Light Detection and Ranging*), radar, RSS (engl. *Responsibility Sensitive Safety*) i semantički LIDAR.

CARLA je zamišljena kao simulacija dinamičnoga svijeta u kojem korisnik, tzv. agent (engl. *agent*), djeluje na svijet unutar simulacije i dobiva zauzvrat određene rezultate. Zbog toga CARLA ima klijent – poslužitelj arhitekturu. Poslužitelj vrti simulaciju i prikazuje scenu u Unreal Engineu, dok klijent može upravljati simulacijom šaljući naredbe i metanaredbe s pomoću programskog jezika Python i mrežnih utičnica (engl. *socket*). Klijent s naredbama može kontrolirati upravljanje vozilom (npr. rukovođenje volanom, ubrzanje i kočenje), s metanaredbama kontrolira ponašanje servera i simulacije (npr. ponovno pokretanje, mijenjanje uvjeta okoline i postavljanje seta senzora), a zauzvrat prima očitanja senzora autonomnog vozila.

1.2. Zapisi cestovne mreže

Precizne informacije o cestovnoj mapi su važni podaci za autonomna vozila. Na temelju njih izračunava se optimalna putanja kretanja, poboljšava se detekcija i klasifikacija objekata, pomaže se zadržavanje vozila u jednoj traci ili upozoravanje o prelasku iz jedne trake u drugu, efikasnije se upravlja potrošnjom goriva i odlučuje se o sigurnijem prolazu kroz raskrižje [8]. Kako bi se postigle ove koristi, mape cestovne mreže moraju ispunjavati dva zahtjeva: geometrijski i implementacijski zahtjev [8]. Geometrijski zahtjev podrazumijeva preciznu i točnu reprezentaciju geometrije ceste, uz dodatno ograničenje zabrane diskontinuiteta ili pogreške u topologiji. Ako se ne poštuje taj zahtjev, može doći do kvara autonomnog vozila i pogrešnog zaključivanja, što je iznimno opasno. Implementacijski zahtjev, s druge strane, traži učinkovit i kompaktan zapis podataka mape cestovne mreže. S tim zahtjevom smanjuje se zauzeće memorije, olakšava se upotreba mapa u vozilu (veći zapisi trebaju veće baze podataka i bolje upravljanje podacima, što je nepraktično za autonomna vozila), poboljšava se djelotvornost *online* prijenosa cestovnih informacija te se smanjuje računsko opterećenje pretprocesiranja geometrije ceste.

Zapisi cestovne mreže razlikuju se po namjeni za koju su dizajnirani. Tako postoji „Google Maps“ čije je korištenje namijenjeno prvenstveno ljudima [9]. S druge strane, formati kao

Autoware Vector Map, OpenDRIVE, Lanelet2, OSM (engl. *OpenStreetMap*) i NDS (engl. *Navigation Dana Standard*) namijenjeni su za autonomna vozila [9]. Autoware Vector Map je format koji je razvila tvrtka Aisan Technology, a sastoji se od tematski raspodijeljenih *csv* datoteka (npr. točka, čvor, traka, itd.) koje su međusobno povezane jedinstvenim identifikatorima [10]. OpenDRIVE format je 2018. godine preuzeo ASAM (*Association for Standardization of Automation and Measuring Systems*) od VIRES Simulationstechnologie GmbH kako bi standardizirao format za simulacije autonomne vožnje [10]. OpenDRIVE ima hijerarhijsku strukturu i implementiran je u XML (engl. *eXtensive Markup Language*) formatu, a sastoji se od referentne linije i traka koje su građene oko nje [3]. Lanelet2 je C++ biblioteka namijenjena pružanju prilagodljivog i opsežnog stvaranja mapa za autonomnu vožnju [10]. Temelji se na *lanelets* elementima (međusobno povezani atomski dijelovi ceste po kojima vozilo može voziti, a geometrijski su prikazani kao lijeva i desna granica, engl. *bound* [11]) i OSM formatu napisanom s pomoću XML-a [10]. OSM format je podatkovna mreža otvorenog koda koju mogu uređivati i unaprjeđivati volonteri, tvrtke i državne institucije u cijelom svijetu [12]. Pisan je u XML formatu, a sastoji se od čvorova, puteva i odnosa [13]. NDS format koristi format datoteka SQLite baze podataka, no potpune funkcionalnosti dostupne su samo članovima NDS društva [10].

U [9] i [10] pregledani su navedeni formati i uspoređeni u tablicama. Na temelju njihovih zaključaka, vidi se da je OpenDRIVE najbolja opcija formata za prikazivanje mapa koje koriste autonomna vozila i simulacije. ASAM je razvio standard koji je detaljan, pruža puno mogućnosti za oblikovanje cestovne mreže i može ga se koristiti u raznim simulatorima. NDS također daje dobre rezultate, no budući da nije otvorenog koda i da sve opcije nisu dostupne bez registracije, ovdje su napuštena daljnja istraživanja. Lanelet i OSM sljedeći su po bodovanju te će se OSM detaljnije opisati zato što je jednostavniji za korištenje od OpenDRIVE-a, besplatan i temelji se na stvarnim podacima [12].

1.2.1. ASAM OpenDRIVE

Glavni je cilj ASAM OpenDRIVE formata pružiti standardiziranu osnovu za prikazivanje cestovnih mreža. Zato koristi XML sintaksu, a zapis se sprema u datoteku ekstenzije *xodr* ili *xodrz* [3]. Prednost takvog zapisa je da se datoteke mogu primjenjivati u različitim simulatorima, pružajući industriji mogućnost smanjenja troška stvaranja mapa i pretvaranja u formate koji su potrebni za razvoj i testiranje modela autonomne vožnje. Informacije o cesti mogu biti temeljene na stvarnim podacima ili proizvoljno stvorene.

Budući da je napisan u XML shemi, OpenDRIVE se sastoji od hijerarhijskih čvorova (engl. *nodes*) koji se dodatno mogu proširiti proizvoljno definiranim podacima [3]. Time se korisniku pruža visoka personalizacija i prilagodba zapisa za određenu aplikaciju (najčešće simulaciju). Čvorovi su sintaksna gradivna jedinica, dok je referentna linija semantička. Referentna linija uvijek se nalazi u sredini ceste u *sth* koordinatnom sustavu (linija se pruža niz s-koordinatu, a udaljenost od linije niz t-koordinatu) te se ostali elementi, kao na primjer trake i signali, postavljaju uzduž referentne linije [3]. Cestovni elementi mogu se povezati međusobno ili s raskrižjem s pomoću atributa prethodnik (engl. *predecessor*) i sljedbenika (engl. *successor*). Cestovne trake se također mogu međusobno povezati. Sveukupno je definirano stotinjak čvorova unutar standarda. Svaki od njih ima određenu ulogu, kao npr. *<line>* koji predstavlja najosnovniji geometrijski elemenat: pravac, te atribute koji ga detaljnije opisuju. Neki čvorovi su obavezni, kao npr. *<planView>* unutar svakog *<road>* elementa kako bi se definirala referentna linija, a neki opcionalni, npr. *<parkingSpace>* element koji definira pravila parkiranja uz cestu. Svaka uloga i korištenje čvora detaljno je opisana u [3].

1.2.2. OpenStreetMap

OpenStreetMap je projekt kojeg je izradila zajednica kartografa, inženjera i humanitaraca, a koji raste svakodnevno [14]. Zadaća OpenStreetMap zajednice je doprinošenje i održavanje podataka o cestama, putevima, ugostiteljskim objektima, željeznicama, i mnogim drugim objektima, sve s pomoću lokalnog znanja. Kako bi se potvrdila točnost i ažuriranost podataka, koriste se slike iz zraka, GPS (engl. *Global Positioning System*) uređaji i jednostavne karte. Glavna prednost OSM formata je što je otvorenog koda i dostupan je svakome.

OpenStreetMap podaci zapisani su u topološki strukturiranom XML formatu [13]. Osnovni elementi koji se koriste za zapis podataka su: čvor (engl. *node*), put (engl. *way*) i odnos (engl. *relation*) [13]. Čvor predstavlja točku u prostoru, a sastoji se od geografske širine i dužine, koji su realni brojevi. Put predstavlja linearnu značajku, npr. cestu, koja se sastoji od najmanje dva čvora, a može biti zatvoren ili otvoren. Svaki element mora imati svoj jedinstveni identifikator (engl. *id*) kako bi se unutar elementa put i odnos mogla zadati referenca na željeni čvor ili put. Odnos je grupa od nula ili više primitiva (čvorova ili puteva) koji imaju zajedničku ulogu. Uz navedene osnovne elemente, postoje i oznake (engl. *tags*).

Svaki element može imati neograničeni broj slobodno definiranih oznaka koje se sastoje od ključa (engl. *key*) i vrijednosti (engl. *value*).

OpenStreetMap format je odličan kandidat za zapis cestovne mreže ako se žele prikazati topološke informacije i pozicije, no nije toliko dobar u predstavljanju geometrije [12]. Ipak, velika količina stvarnih podataka dostupnih cijelom svijetu daje mu znatnu prednost nad ostalim formatima.

1.3. Proceduralno generiranje

Kako je prije navedeno, proceduralno generiranje je postupak generiranja sadržaja kroz korake algoritma uz pomoć niza pravila i ograničenog uplitanja korisnika [4]. Glavne prednosti takvog postupka su malo memorijsko zauzeće, obilje generiranog sadržaja, raznolikost sadržaja i jednostavna izmjena parametara ili pravila algoritma [4]. Sadržaj kojeg proizvede algoritam često zauzima manje prostora u memoriji nego ručno izrađen objekt, no budući da je u ovom slučaju rezultat datoteka *.xodr* formata s osnovnom reprezentacijom cestovne mreže čija veličina ne utječe previše na zauzeće memorije, taj faktor nije presudan. S druge strane, proceduralno generiranje daje različite nasumične rezultate te se tako može dobiti skoro beskonačan broj objekata koji imaju slična svojstva, no različit sadržaj. U ovom radu, to će pružiti mnogobrojne staze i puteve po kojima se autonomna vozila mogu trenirati i testirati. Raznolikost je bitna, pogotovo u igrama, kada treba izraditi više istih objekata, ali s puno varijacija. Algoritam će izraditi raznolikiji sadržaj nego čovjek te će u ovom slučaju to značiti raznovrsniju cestovnu mrežu. Na rezultate se jednostavno utječe promjenom globalnih parametara ili pravila algoritma što poboljšava brzinu ponavljanja generiranja. Međutim, iako je mukotrpan ručni rad maknut iz postupka izrade sadržaja, rezultatima može nedostajati ljudska kreativnost i domišljatost.

Postoje razne podijele postupaka proceduralnog generiranja. Prema [15] proceduralno generiranje sadržaja za igre dijeli se na tradicionalni pristup, metode temeljene na pretraživanju (engl. *search-based methods*) i metode strojnog učenja. U tradicionalni pristup spadaju generator pseudoslučajnih brojeva, generativne gramatike, fraktali i šum te su to prvi postupci proceduralnog generiranja u igrama. Metode temeljene na pretraživanju funkcioniraju tako da prvo stvore neki sadržaj, onda ga ocjene određenom funkcijom te nastavljaju stvarati sadržaj dok ocjena ne dosegne neku granicu. Najčešće implementacije tih metoda su stohastički evolucijski algoritmi. Metode strojnog učenja uključuju rekurzivne

neuronske mreže (engl. *recurrent neural networks*), autoenkodere (engl. *autoencoders*), GAN-ove (engl. *Generative Adversarial Networks*) i Markovljeve modele (engl. *Markov model*). S njima se može autonomno generirati sadržaj, dizajnirati sadržaj u suradnji s autorom i sažimati podatke. S druge strane, [16] dijeli postupke proceduralnog generiranja na nasumično generiranje, generiranje na temelju gramatika i generiranje s pomoću evolucijskih algoritama.

U obje podijele, generativna gramatika zauzima važnu poziciju u postupku proceduralnog generiranja. Formalna gramatika sastoji se od skupa pravila (produkcija) koji gradi nizove znakova u nazivlju formalne gramatike iz završnih znakova [17]. Postupak počinje od početnog niza koji se naziva aksiom te se nad njim primjenjuju određena pravila. Ovisno o poretku uporabe pravila i broju koraka algoritma, može se generirati beskonačno mnogo različitih nizova znakova [5]. Stvaranje nizova formalnom gramatikom može se prikazati stablom [17], odnosno acikličkim neusmjerenim grafom. Jedna od popularnih gramatika za proceduralno generiranje je L-sustav [16].

1.3.1. L-sustavi

L-sustave je razvio biolog Aristid Lindenmayer kao matematičku teoriju razvoja biljaka [18]. U početku se prof. Lindenmayer posvetio topologiji biljaka, želeći opisati odnose između stanica ili većih biljnih modula, a geometrijski aspekti su razvijeni kasnije. Jedna od najpoznatijih interpretacija L-sustava je geometrija kornjače (engl. *turtle geometry*).

Glavna ideja L-sustava je prepisivanje (engl. *rewriting*) [18]. Prepisivanje je tehnika koja definira složene objekte tako da uzastopno zamjenjuje dijelove jednostavnog početnog objekta uz skup pravila, odnosno produkcija. Zamjena se primjenjuje paralelno i u isto vrijeme, simulirajući tako diobe stanica u višestaničnim organizmima [18]. L-sustav se obično opisuje s pomoću uređene trojke:

$$G = (V, w, P),$$

gdje G označava sustav, V rječnik, odnosno skup varijabli koje su dostupne sustavu, w je aksiom koji opisuje početni niz prije nego što ga L-sustav promijeni, a P predstavlja skup pravila koja određuju kako se sustav ponaša [5]. Ako unutar P postoji samo jedno pravilo koje se može primijeniti točno jednoj varijabli, takav sustav naziva se deterministički L-sustav [5]. Riječ „deterministički“ označava da će sustav uvijek proizvesti isti rezultat. Deterministički L-sustav ima nekoliko nedostataka, uključujući ograničavajući izbor (jedno

pravilo za svaku varijablu) i nepostojeću raznolikost. Budući da je cilj L-sustava oponašati prirodu, determinizam nije dobar odgovor jer daje identične rezultate koliko god puta se pokrene dok u prirodi nijedna biljka nije sasvim identična. Dodavanjem više pravila za pojedinu varijablu, i posljedično nasumičnost, predstavlja se stohastički L-sustav [5]. Slučajnost se dobiva tako da se svakom produkcijskom pravilu pridodaje faktor vjerojatnosti. Time se postiže prilagodba međusobnog utjecaja pravila, dok se i dalje zadržava mogućnost odabira produkcijskog pravila za svaku varijablu. Taj odabir se najčešće odlučuje s generatorom pseudonasumičnih brojeva, uz napomenu da je potrebno inicijalizirati generator s pravilnim sjemenom (engl. *seed*) [5].

Kako bi se prešlo iz nizova u geometriju, koristi se metoda kornjače. Stanje kornjače pamti se s pomoću trojke (x, y, α), gdje x i y predstavljaju položaj kornjače unutar Kartezijevog koordinatnog sustava, a α kut po kojem se kornjača giba [18]. Ovisno o veličini koraka d i faktor povećanja kuta δ, kornjača se giba po koordinatnom sustavu i ostavlja trag. Njeno gibanje predstavlja implementaciju L-sustava [5].

1.4. Pregled postojećih rješenja

U ovom poglavlju dat će se pregled nekoliko implementacija proceduralnog generiranja cestovnih mreža.

U radu [19] autori su odlučili organizirati proces hijerarhijski u slojeve, što odražava semantički opis cestovnog okruženja. Najniži sloj je sloj čvorova (engl. *Nodes*) koji stvara popis čvorova na temelju danog terena. Svaki čvor ima svoju poziciju i važnost (engl. *importance factor*) te na temelju tih faktora sloj topologije cesta (engl. *Topological roads*) grupira čvorove kao velegradove, gradove i sela. Na kraju, sloj središnjih linija (engl. *Centerlines*) stvara definiciju cestovnog puta za svaku cestu, a sloj definicije cestovnog okruženja (engl. *Road environment definition*) kreira 3D modele cestovne mreže i okolnog terena.

Autori Jo i Sunwoo predložili su u [20] algoritam za stvaranje precizne cestovne mape za autonomnu vožnju koji se sastoji od tri koraka: prikupljanje podataka, nadopunjavanje podataka i modeliranje ceste. Na početku su opremili vozilo s RTK-GPS senzorom koje ima centimetarsku točnost te su s pomoću njega sakupili sirove podatke. Budući da RTK-GPS senzor ima nekoliko ranjivosti, u drugom koraku kombiniraju njegove podatke s

informacijama sa senzora na vozilu. Konačno, prema obrađenim podacima grade matematički model ceste koji opisuje geometriju cestovne mreže.

Jedan od glavnih dijelova proceduralnog generiranja virtualnih gradova, kojeg opisuju Brol i Anroniuk u [21] je proceduralno generiranje cestovne mreže. Njihovo rješenje je planarni graf koji opisuje cestovnu mrežu grada. U planarnom grafu čvorovi predstavljaju raskrižja i zavoje, a rubovi ceste koje ih spajaju. Takav zapis ne ograničava vrste cestovnih elemenata te se čak mogu naći i nadvožnjaci u cestovnoj mreži. Nakon generiranja planarnog grafa, stvaraju se 3D modeli ceste na temelju visinske mape ili modela terena.

Slično prethodnom radu, [22] također opisuje proceduralno generiranje gradova, ali s malo drukčijim pristupom. Sharma temelji uzorak cestovne mreže na gustoći populacije. Prema tome, ceste su podijeljene u dvije klase: primarne (autoceste) i sekundarne (gradske ceste). Primarne ceste povezuju gusto naseljena područja, dok su sekundarne ceste definirane kao ceste koje su okružene autocestama. Na ulaz generatora postavlja se slika u sivim tonovima (engl. *gray-scaled image*) na kojoj vrijednost 1 (bijelo) odgovara najgušće naseljenom području, a vrijednost 0 (crno) nenaseljenom području. U početku sustav gleda područja s najvećom vrijednosti na mapi te ih pretvara u populacijske čvorove (engl. *Population-nodes*). Oni predstavljaju dijelove mape koji će biti spojeni autocestom. Sljedeće, autocesta se gradi tako da se odabere jedan populacijski čvor, iz njega se pruže zrake prema drugim središtima te se uspoređuje suma udaljenosti čvorova svake zrake. Najveća suma određuje smjer gradnje autoceste te se tako postupak ponavlja dok se ne izgrade sve autoceste. Za izgradnju gradskih cesta korišteni su prilagođeni L-sustavi, osim za kružne ceste koje se generiraju oko određenog centra. Sve ceste su kreirane s pomoću Catmull-Rom krivulje.

Rad [23] također proučava proceduralno generiranje grada, no Parish i Müller su drukčije organizirali proces proceduralnog generiranja cestovne mreže. Uzeli su L-sustav i s pomoću njega implementirali osnovna pravila cestovne mreže. Takav L-sustav vraća šablonu cestovne mreže, bez vanjskih utjecaja. Zatim se na nju primjenjuju globalni ciljevi koji spadaju u dvije kategorije: uzorak ulica (engl. *street pattern*) i gustoća naseljenosti (engl. *population density*). Na kraju se provjeravaju lokalna ograničenja koja mogu biti granice kopna, vode ili parka, nadmorska visina i sjecišta cesta.

2. Implementacija

Implementacija će pratiti rad [5] uz par preinaka kako bi se rezultat prilagodio ASAM OpenDRIVE standardu te se neće koristiti mape gustoće naseljenosti.

Postupak generiranja cestovne mreže može se podijeliti na četiri dijela: definiranje pravila, generiranje stabla s pomoću L-sustava, pretvaranje stabla u ciklički neusmjereni graf te na kraju stvaranje cestovne mreže u *xord* formatu. Svako poglavlje obradit će zaseban dio postupka pa će tako u poglavlju 2.1 biti opisana klasa s pomoću koje se definiraju pravila i kako ona utječu na konačan rezultat. U poglavlju 2.2 pokazat će se primjena tih pravila, proći će se kroz rekurziju postupka te objasniti kako se dobije nasumičnost cestovnih mreža. Poglavlje 2.3 obradit će prebacivanje strukture podataka iz stabla u graf i detaljnije objasniti korištenu biblioteku *networkx*, a u poglavlju 2.4 bit će opisano od čega se sastoji OpenDRIVE zapis, kako su definirane klase koje predstavljaju određeni segment ceste i na koji način se te klase iscrtavaju.

2.1. Pravila L-sustava

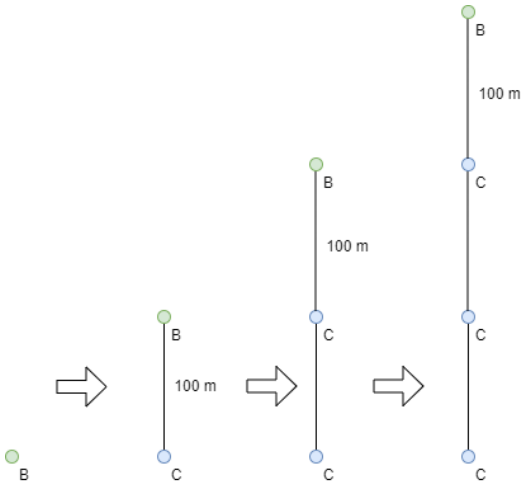
Pravila po kojima će se generirati L-sustav definirana su u klasi `Rule` koja se sastoji od šest atributa: `matchnodetype`, `changetotype`, `spawntype`, `spawncount`, `length` i `angle`. Prva tri atributa određuju tip čvora, s time da `matchnodetype` postavlja tip čvora kojeg algoritam traži kada želi primijeniti neko pravilo. Kada ga nađe pretvara ga u `changetotype` i od njega nadalje stvara novi čvor tipa `spawntype`. Ovisno o atributu `spawncount`, određuje se broj novih čvorova koji se postavljaju na udaljenost spremljenoj u atributu `length` i u smjeru atributa `angle` (u stupnjevima).

Primjer odsječka kôda u kojem se definiraju pravila izgledao bi ovako:

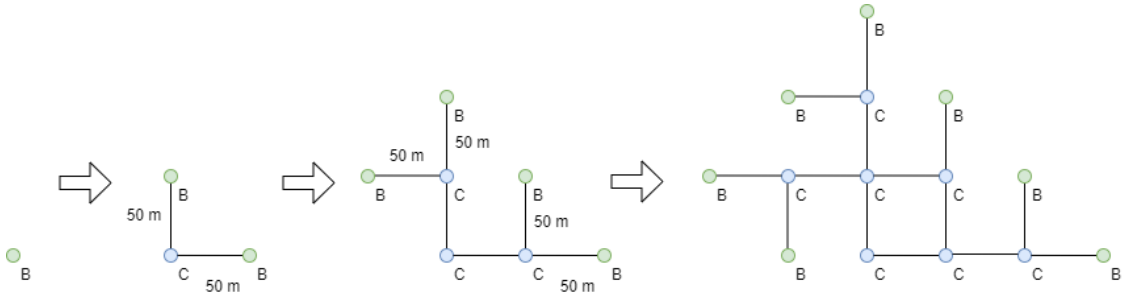
```
rule1 = Rules.Rule(length=100)
rule2 = Rules.Rule("B", "C", "B", 2, 50, 90)
rule3 = Rules.Rule("B", "C", "B", 1, 20, 30)
```

U prvom redu definira se pravilo koje uzima zadane vrijednosti te postavlja jedino atribut `length` na vrijednost 100. To pravilo stvara ravnu cestu duljine sto metara (slika 1). Drugo i treće pravilo definiraju vlastite tipove čvorova, a razlikuju se po tome što drugo pravilo stvara dva čvora na udaljenosti pedeset metara od pronađenog čvora pod kutom od devedeset

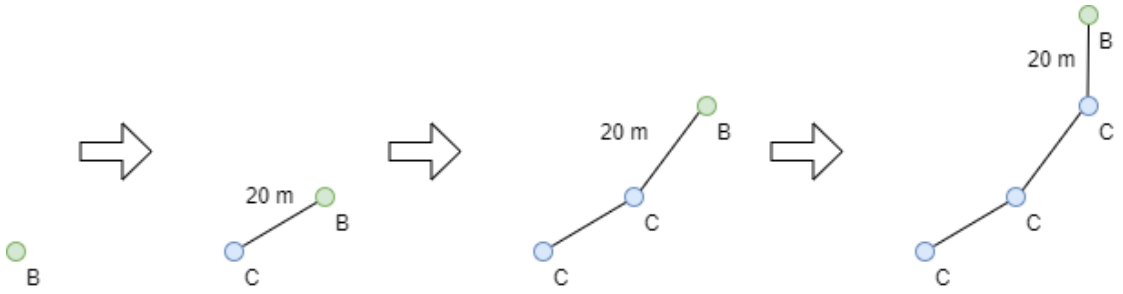
stupnjeva (slika 2), dok treće pravilo stvara samo jedan čvor na udaljenosti dvadeset metara pod kutom od trideset stupnjeva (slika 3).



Slika 1. Primjer prvog pravila kroz tri iteracije, nacrtano u Draw IO [30]



Slika 2. Primjer drugog pravila kroz tri iteracije, nacrtano u Draw IO [30]



Slika 3. Primjer trećeg pravila kroz tri iteracije, nacrtano u Draw IO [30]

Ovakav način definiranja pravila daje korisniku veliku slobodu korištenja i eksperimentiranja s raznim izgledima mreža. S druge strane, mogu se definirati različite vrste čvorova za koje se primjenjuje određeno pravilo i time imati veliku kontrolu nad samim procesom generiranja. Različiti tipovi čvorova nude mogućnost proširenja postupka generiranja L-sustava uvođenjem hijerarhije čvorova, a time i cesta, čime se može dobiti realističniji prikaz cestovne mreže.

2.2. Stabla i nasumičnost

Proces generiranja L-sustava, odnosno stabla, započinje početnim čvorom koji je spremljen unutar varijable `axiom`. Nad njime se tada primjenjuje funkcija `processNode` koja se rekurzivno poziva za svako dijete-čvor početnog čvora. Na kraju vraća početni čvor, ali koji se pretvorio u korijen stabla i koji sadrži popis svoje djece, koja onda imaju popis svoje, a ona svoje, itd. Stvaranje novih čvorova (primjena pravila) odvija se u funkciji `applyRules`. Pravilo se odabire nasumično kako bi cestovna mreža nakon svakog pokretanja bila različita te se to ostvaruje slučajnim odabirom iz liste pravila:

```
randomrule = self.rules[random.randrange(0, len(self.rules))]
```

Ako tip čvora odgovara odabranom pravilu, sljedeći korak je definiranje svih potrebnih parametara novog čvora. Klasa `Node` predstavlja čvor, a sadrži atribute `id`, `s`, `x`, `y`, `nodetype`, `angle`, `length`, `children` i `junction`. Kada se stvara novi čvor, atribut `id` mora biti jedinstven te zbog toga postoji interni brojač koji se poveća nakon stvaranja novog čvora. Budući da su za kreiranje čvora definirani samo duljina i kut, a u standardu OpenDRIVE očekuju se koordinate u Kartezijevom koordinatnom sustavu, potrebno je pretvoriti te vrijednosti u x i y koordinate (radi jednostavnosti, ovdje se zanemaruje z koordinata). Za pretvaranje iz polarnih koordinata (r, θ) u Kartezijeve koordinate (x, y) korištene su formule:

$$x = r\cos\theta \tag{1}$$

$$y = r\sin\theta \tag{2}$$

Kut θ dobio se tako da se polukrug podijelio s brojem čvorova koji će se generirati i taj broj nadodao na kut koji je bio spremljen u početnom čvoru, čime svi čvorovi završavaju ispred početnog čvora i time se izbjegavaju oštri kutovi. Tako dobiveni kut θ mora se pretvoriti u radijane.

Atribut `s` postavlja se na nulu, budući da je svaki čvor početak ceste, a time i referentne linije, dok se `junction` postavlja na `False` jer se još ne zna koliko će čvorova biti povezano s tim čvorom.

Kada su definirani svi potrebni podaci, postavlja se varijabla `newnode` na ovakav način:

```
newnode = Tree.Node(self.counter, 0, node.x + new_x, node.y +
new_y, randomrule.spawntype, new_angle)
```

te se dodjeljuje kao dijete polaznom čvoru.

2.3. Ciklusi i raskrižja

Glavna motivacija pretvaranja dobivenog stabla L-sustava u graf upravo su ciklusi. Kako je ilustrirano u [5], ceste L-sustava mogu se početi približavati jedna drugoj ili čak presijecati. Takvo ponašanje je poželjno budući da su ciklusi i raskrižja često prisutni u stvarnim cestovnim mrežama, no stablo nije najpovoljnija struktura za prikaz takve ceste. Jedan od bitnijih problema je što čvorovi u stablu imaju znanje samo o svojoj djeci-čvorovima, ne i o roditeljima, osim ako se ne implementira povratna veza. U ovom slučaju, odabir strukture grafa činio se kao bolji izbor, zbog čega je korištena biblioteka *networkx*. *Networkx* je pisan u Python jeziku, a stvoren je kako bi omogućio istraživanje i analizu grafova i pripadajućih algoritama [24]. Od struktura podataka mreža, biblioteka pruža potporu stvaranja jednostavnih grafova, usmjerenih grafova te grafova s dvostrukim bridovima i petljama. Osnovne gradivne jedinice grafa u *networkx*-u su vrhovi i bridovi. Vrhovi mogu biti bilo koji Python objekt, kao na primjer broj, niz znakova, datoteke ili funkcije, pa su se u ovom slučaju koristile instance klase `Node`. Bridovi su implementirani kao par vrhova, npr. brid između vrha A i B zapisuje se kao (A, B), a svaki brid može sadržavati i dodatna proizvoljna svojstva. Kako bi se mogli vizualizirati rezultati, *networkx* može surađivati s *matplotlib* bibliotekom, uz jednostavne algoritme za pozicioniranje čvorova temeljene na metodama zasnovanim na sili te spektralnim i geometrijskim metodama, što dodatno olakšava stvaranje i analiziranje grafova. Uz nabrojene funkcionalnosti, *networkx* ima i mogućnost izračuna svojstava i metrika mreže, kao na primjer najkraći put, centralnost međupoloženosti (engl. *betweenness centrality*), grupiranje (engl. *clustering*) i raspodjela stupnjeva vrha.

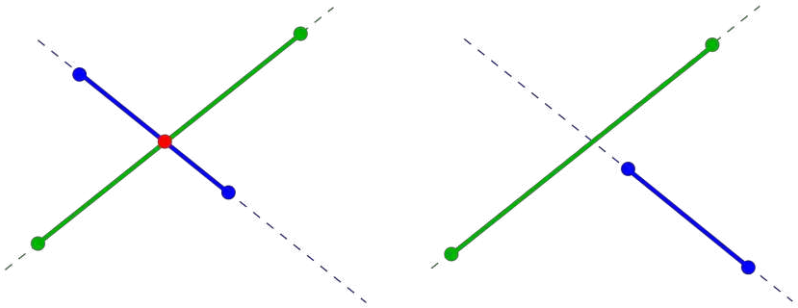
Unutar klase `Graph` definirane su funkcije potrebne za stvaranje grafa, provjere raskrižja te dodatne funkcije za brisanje vrhova koji su međusobno geometrijski blizu i stvaranja novih vrhova kod presjeka bridova. Prebacivanje strukture stabla u graf provodi se rekurzivno tako da se prolazi kroz sve čvorove stabla, dodaje ih se kao vrhove grafa te se između odnosa roditelj-dijete stvara novi brid. Raskrižje se provjerava jednim prolaskom kroz sve vrhove grafa te provjerom je li stupanj vrha veći ili jednak tri. Ako jest, varijabla `junction` se ažurira na vrijednost `True`. Prema [3], u OpenDRIVE standardu postoje tri vrste raskrižja: obično (prometne trake se preklapaju), izravno (povezane trake se ne preklapaju) i virtualno raskrižje (glavna cesta nije prekinuta). Zbog jednostavnosti, u ovom radu koristit će se

običnо raskrižje. Kasnije, kada će se zapisivati cestovna mreža u *xodr* format, bit će potrebno znati je li neka cesta povezana s raskrižjem ili cestom te je zato uvedena ova funkcija.

Brisanje vrhova koji su u neposrednoj blizini dodatna je funkcionalnost koja se može i ne mora koristiti, ovisno o rezultatima koje daje i namjeravanom izgledu mreže. Implementirana je zbog slučaja u kojem L-sustav ima puno iteracija te se vrhovi počinju nakupljati na malom području. Kada se to dogodi, pronalaze se čvorovi koji su na nekoj željenoj udaljenosti te se brišu iz cestovne mreže, uz preusmjeravanje bridova na najbliži čvor kako se ne bi izgubile informacije. Tu može doći do previše kompliciranih križanja (desetak cesta na jednom raskrižju) što ne daje realističan učinak. Za izračun udaljenosti vrhova koristi se Euklidska udaljenost u dvije dimenzije:

$$d(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{3}$$

Prepoznavanje sjecišta također je prirodna reakcija na ponašanje L-sustava zato što se može dogoditi da se bridovi grafa sijeku. Problemu se početno pristupilo određivanjem sjecišta dvaju pravaca, no ubrzo se ustanovilo da se pravci, koji su produžetci bridova, mogu sjeći izvan grafa (i bridova) kako je ilustrirano na slici 4.



Slika 4. Prikaz presjeka dva linijska segmenta, preuzeto s [25].

Kako bi se dobio presjek bridova, odnosno linijskih segmenata, koristila se parametarska reprezentacija pravaca definiranih točkama (x_1, y_1) i (x_2, y_2) te (x_3, y_3) i (x_4, y_4):

$$(x(s), y(s)) = (x_1 + s(x_2 - x_1), y_1 + s(y_2 - y_1)) \tag{4}$$

$$(x(t), y(t)) = (x_3 + t(x_4 - x_3), y_3 + t(y_4 - y_3)) \tag{5}$$

Segmenti se sijeku u točci (x_0, y_0) samo ako vrijedi $0 \leq s_0, t_0 \leq 1$. Točka (s_0, t_0) je rješenje sustava jednadžbi:

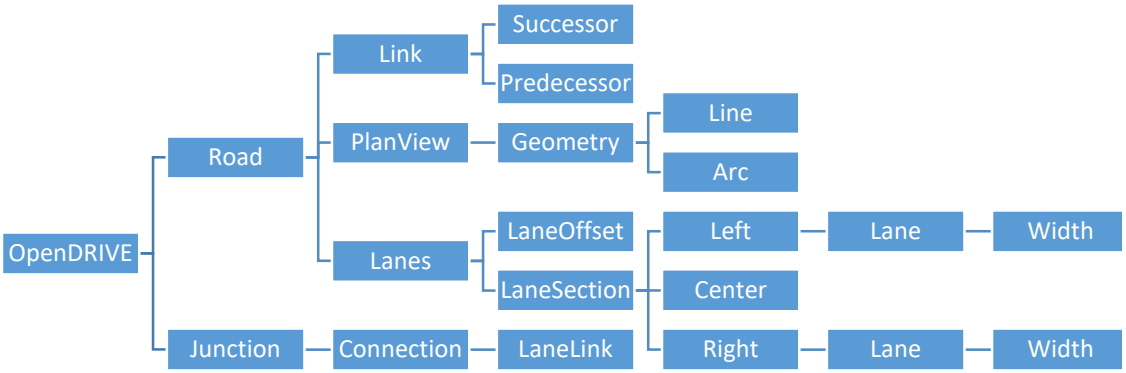
$$s(x_2 - x_1) - t(x_4 - x_3) = x_3 - x_1 \quad (6)$$

$$s(y_2 - y_1) - t(y_4 - y_3) = y_3 - y_1 \quad (7)$$

Sustav jednadžbi može se riješiti s pomoću Cramerovog pravila. Tada vrijedi $s_0 = \frac{D_s}{D}$ i $t_0 = \frac{D_t}{D}$, gdje su D_s i D_t definirani kao determinante matrica u kojima je stupac uz varijablu s, odnosno t, zamijenjen s vrijednostima s druge strane znaka jednakosti, a D determinanta matrice vrijednosti uz nepoznanice [26]. Potrebno je pripaziti na slučaj kada su segmenti paralelni zato što onda vrijedi $D = 0$ i može doći do dijeljena s nulom. U ovom slučaju to se jednostavno rješava tako da se ne izračunavaju vrijednosti s_0 i t_0 jer nema presjeka. Jednom kada se izračunaju (s_0, t_0) uvrštavaju se u (4) ili (5) te se tako dobije točka (x_0, y_0). Prije stvaranja novog vrha, provjerava se postoji li točka presjeka u popisu vrhova grafa budući da se svi bridovi sijeku u vrhu. To se također moglo provjeravati dijele li vrh bridovi koje se razmatra. Ako je sjecište bridova novo, stvara se novi vrh te se brišu stari bridovi i dodaju novi koji spajaju prijašnje vrhove s novim vrhom.

2.4. Segmentacija cestovne mreže

OpenDRIVE standard ima detaljno opisana pravila i strogo propisan izgled svakog XML elementa. Zbog toga svaka oznaka koja u OpenDRIVE formatu ima oblik *<naziv atribut="vrijednost">* implementirana je kao zasebna klasa. Hijerarhijski, sve klase korištene za generiranje cestovne mreže mogu se prikazati s pomoću grafa 1.



Grafikon 1. Hijerarhijski prikaz klasa u standardu OpenDRIVE, ideja preuzeta iz [9]

Svaka klasa sastoji se od konstruktora u kojem se definiraju potrebni atributi, kao na primjer `id`, `x` i `y` koordinate, `roads` i sl., te potrebne funkcije za ispis tog cestovnog elementa. Većina klasa ima početni i završni element kojim se označuje njegovo djelovanje i hijerarhijska pozicija. Primjerice, *geometry* element koji opisuje geometriju svake ceste izgleda ovako:

```
<geometry s="0" x="0" y="0" hdg="1.5707963267948966" length="50.0">
      <line/>
</geometry>
```

Za takve strukture, klase imaju dvije funkcije ispisa: `printBegin` i `printEnd`. Prva ispisuje početnu liniju s potrebnim atributima, a druga oznaku kraja tog elementa. Kako bi se obje funkcije mogle pozvati, kreirana je još jedna funkcija koja poziva `printBegin`, prolazi kroz sve elemente koji su hijerarhijski ispod trenutnog elementa, nad njima poziva funkciju ispisa i na kraju pokreće `printEnd`. Cestovni elementi koji imaju jednu liniju ispisa, kao na primjer `Line`, definiraju samo jednu funkciju za ispis. Jednom kada se pokrene funkcija `printOpenDrive` unutar klase `OpenDRIVE`, ona sâma ispisuje cijelu mrežu rekurzivnim pozivom svih ostalih elemenata.

Funkcija `generateMesh` uzima kao argument graf te njegove vrhove i bridove pretvara u cestovne elemente OpenDRIVE standarda. Funkcija iterira po svim bridovima grafa, određuje duljinu, normalizirani vektor smjera i usmjerenje brida u radijanima te početne (x, y) koordinate. Nakon toga određuje se širina ceste s pomoću klase `Width`. Ona se koristi u klasi `Lane` kako bi se definirale prometna traka (`drivingLane`) i rubnik (`shoulderLane`). Uz njih, definira se i referentna linija (`referenceLine`). Ovisno s koje strane se nalaze, prometna traka i rubnik dodjeljuju se lijevoj (`leftLane`) i desnoj traki (`rightLane`), a referentna linija središnjoj traki (`centerLane`). Sve tri trake spremaju se u klasu `LaneSection` koja se, uz `LaneOffset`, pridružuje klasi `Lanes`. Nakon njih, definira se geometrija, na sljedeći način:

```
geometry = RoadView.Geometry(firstNode.s, x, y, heading,
length, line=line).
```

Tako definirana geometrija dodaje se klasi `PlanView`. Uz navedene elemente, trebaju se ustanoviti susjedni cestovni elementi kako bi auti cestom mogli upravljati. U OpenDRIVE standardu povezivanje cestovnih elemenata dopušteno je samo kada su referentne linije direktno povezane. Preklapanja i preskoke se preporuča izbjegavati, kako je detaljno

ilustrirano u [3]. Povezanost cesta reprezentira se klasom `Link` koja može sadržavati nula ili više klasa `Predecessor` i `Successor`. Klasa `Predecessor` predstavlja prethodnika povezanog na početak referentne linije trenutnog cestovnog elementa, dok klasa `Successor` označava sljedbenika cestovnog elementa koji je spojen na kraj referentne linije. Prethodnici i sljedbenici mogu biti ceste ili raskrižja te se za svaki definiraju posebni atributi. Na primjer, za raskrižje prethodnika postavlja se tip prethodnika na „junction“, identifikacijski broj na *id* raskrižja, a točka povezivanja na „start“ ili „end“ ovisno o položaju raskrižja. S druge strane, za cestu sljedbenika tip se postavlja na „road“, identifikacijski broj na broj brida, a točka povezivanja na „start“ ili „end“ ovisno o položaju referentne linije te ceste (započinje li ili završava cesta u toj točki). Na kraju se sve trake, geometrija, poveznice i dodatni atributi kao naziv, duljina, identifikacijski broj i informacija pripada li cesta raskrižju spremaju u instancu klase `Road`:

```
road = Road.Road("Road " + str(graph.edges[edge]['id']),
length, graph.edges[edge]['id'], -1, [lanes], planView,
[link])
```

Svaka cesta, odnosno brid, sprema se u klasu `OpenDRIVE`. Kada petlja završi svoje izvršavanje, otvara se datoteka formata *xodr* i u nju se zapisuju svi nizovi znakova (engl. *string*) koje vraća klasa `OpenDRIVE`. Također, na početku je potrebno deklarirati da je datoteka u XML formatu. Cjelokupan kôd izgledao bi ovako:

```
f = open("cesta.xodr", "w")
f.write('<?xml version="1.0" encoding="UTF-8"?>\n')
f.write(opendrive.printOpenDrive())
f.close()
```

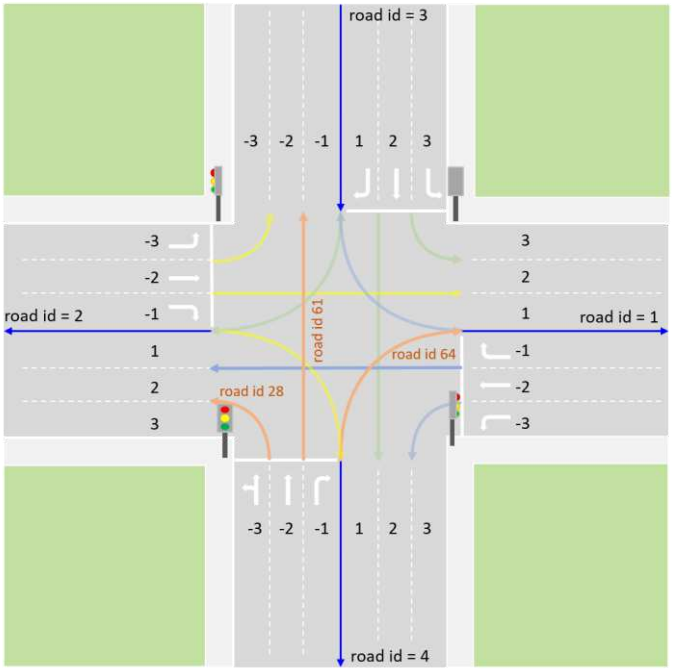
Navedena funkcija dobra je za osnovno stvaranje ravnih cesta, no nedostaje joj baratanje raskrižjima. Funkcija `generateMesh` samo stvara nove ceste na kraju prethodne bez zavoja i glatkih prijelaza. Zbog toga je dodana funkcija `generateJunctions`. Poziva se nakon izvršene petlje po bridovima, a prije spremanja u *xodr* datoteku. Za razliku od `generateMesh`, `generateJunction` iterira po vrhovima grafa i provjerava je li vrh raskrižje. Ako jest, odrađuje se potrebna računica i postupci, a ako nije, zanemaruje ga se i nastavlja se na sljedeći vrh. Kada se pronađe raskrižje, stvara se instanca klase `Junction`, prvotno s nedefiniranim vezama jer će se one kasnije nadodati. U raskrižju se želi povezati svaku cestu sa svakom cestom te se zato iterira po svim kombinacijama bridova koji sadržavaju trenutni čvor. Kako ne bi došlo do preklapanja cesta, svaka cesta se pomiče za

pet metara unazad. To pomicanje ostvareno je tako da se izračuna duljina i normalizirani smjer pružanja ceste te se na originalne koordinate dodaje pet puta usmjerenje ceste:

```
start = [node.x + 5 * direction1[0], node.y + 5 * direction1[1]]

end = [node.x + 5 * direction2[0], node.y + 5 * direction2[1]]
```

Kada se u raskrižju spaja više cesta, može doći do dva slučaja: s jedne ceste se skreće u zavoj prema drugoj ili se nastavlja voziti ravno (slika 5.)



Slika 5. Prikaz raskrižja iz standarda OpenDRIVE [3]

Odluka o ravnoj cesti ili zavoju donosi se unutar funkcije `checkCollinear` koja prima tri točke i vraća vrijednost `True` ili `False`. U suštini funkcija provjerava jesu li tri točke kolinearne tako da provjerava jesu li nagibi pravca između prve dvije točke i pravca između druge i treće točke jednaki. Ako su definirane dvije točke (x_1, y_1) i (x_2, y_2), jednadžba pravca može se zapisati na ovaj način:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \tag{8}$$

Razlomak $\frac{y_2 - y_1}{x_2 - x_1}$ predstavlja nagib pravca. Iznimka su točke koje čine vertikalne pravce pa dolazi do dijeljenja s nulom. U tom slučaju, nagib se postavlja na beskonačnost. Zbog numeričke nepreciznosti zapisa, nagibi se uspoređuju tako da se provjerava je li njihova apsolutna razlika manja od deset na minus desetu.

Ako je donesena odluka da je cesta ravna, izračunava se smjer kretanja ceste i njena duljina, formulama:

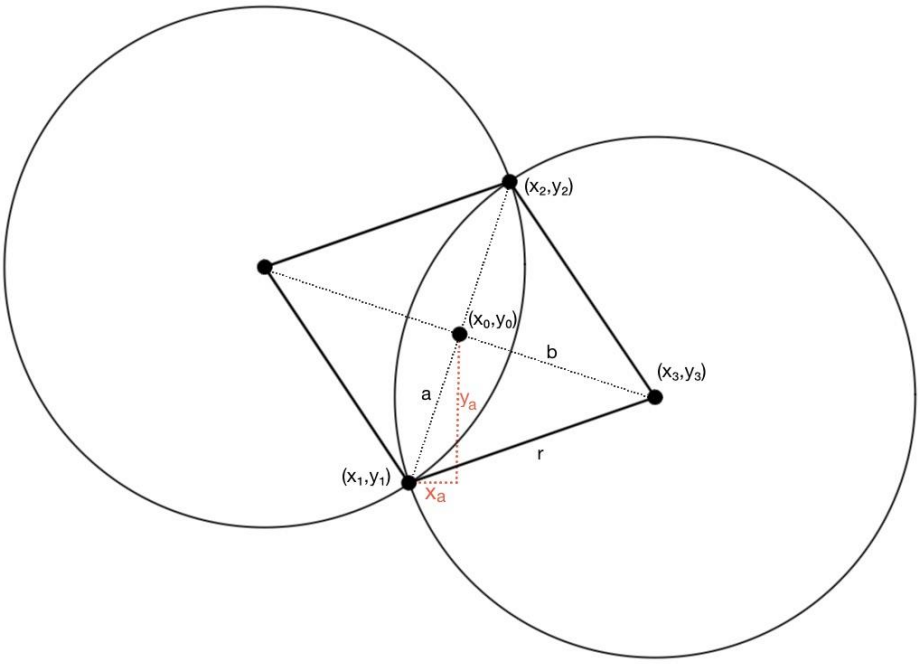
```
heading = math.atan2((end[1] - start[1]), (end[0] - start[0]))
```

```
distance = math.sqrt(math.pow(end[0] - start[0], 2) + math.pow(end[1] - start[1], 2))
```

te se crta geometrija na isti način kao i u prethodnoj funkciji `generateMesh`. Ako je odlučeno da je potreban zavoj, slijedi više računanja. U OpenDRIVE standardu zavoj se može prikazati sa spiralom, lukom, kubnim polinomima i parametarskim kubnim polinomima. Prema [3], spirale se koriste za prijelaz između ravne linije i luka. Spirala linearno mijenja zakrivljenost referentne linije, dok luk zadržava konstantnu zakrivljenost. Kubni polinomi se ne preporučaju za korištenje te se umjesto njih mogu definirati parametarski kubni polinomi koji su fleksibilniji, a primjenjuju se na složene krivulje dobivene iz izmjerenih podataka. Zbog jednostavnosti, ovdje je definiran samo luk koji stalnom vrijednošću zakrivljuje referentnu liniju. Sâma zakrivljenost računa se s pomoću formule (9):

$$curvature = \frac{1}{radius} \tag{9}$$

gdje je $radius$ polumjer kruga čiji kružni isječak odgovara zavoju. Ako je vrijednost zakrivljenosti pozitivna, krivina se kreće u smjeru suprotnom od kazaljke na satu (lijevi zavoj), a ako je negativna, u smjeru kazaljke na satu (desni zavoj). Da bi se izračunala zakrivljenost zavoja, prvo je potrebno definirati kružnicu i kružni isječak koji će zavoj slijediti. Poznate su dvije točke (x_1, y_1) i (x_2, y_2) između kojih će biti iscrtan zavoj. Postoje dva moguća centra kružnice (x_3, y_3) i (x_4, y_4), kako je prikazano na slici 6.



Slika 6. Računanje centra kružnice s pomoću dvije točke, preuzeto iz [27].

Prema [27], poznate točke i dva centra tvore romb takav da je svaka stranica duljine polumjera kružnice. Pola udaljenosti između dvije poznate točke može se dobiti s pomoću sljedeće formule:

$$a = \sqrt{(\frac{x_2 - x_1}{2})^2 + (\frac{y_2 - y_1}{2})^2} \tag{10}$$

Budući da dijagonale romba tvore pravokutne trokute, udaljenost između dva moguća središta, a koja je okomita na segment između dvije početne točke, može se izračunati s pomoću formule:

$$b = \sqrt{r^2 - a^2} \tag{11}$$

Dijagonala $2b$ dijeli romb na dva jednakokračna trokuta te se krakovi, odnosno polumjer, mogu izračunati prema sljedećoj formuli:

$$r = \frac{a}{sin\frac{\theta}{2}} \tag{12}$$

gdje je θ kut kod središta kružnice.

Da bi se dobilo središte romba, označeno s (x_0, y_0) na slici 3, potrebno je definirati varijable x_a i y_a. Ako su $x_a = \frac{x_2 - x_1}{2}$ i $y_a = \frac{y_2 - y_1}{2}$, onda se središte romba formulira s:

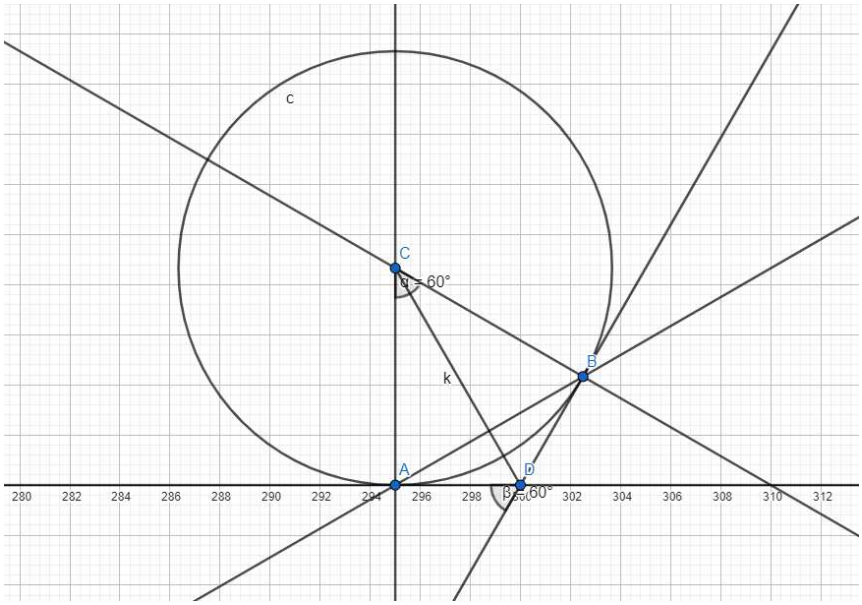
$$x_0 = x_1 + x_a,\ y_0 = y_1 + x_a \tag{13}$$

Sada se lako odredi centar kružnice:

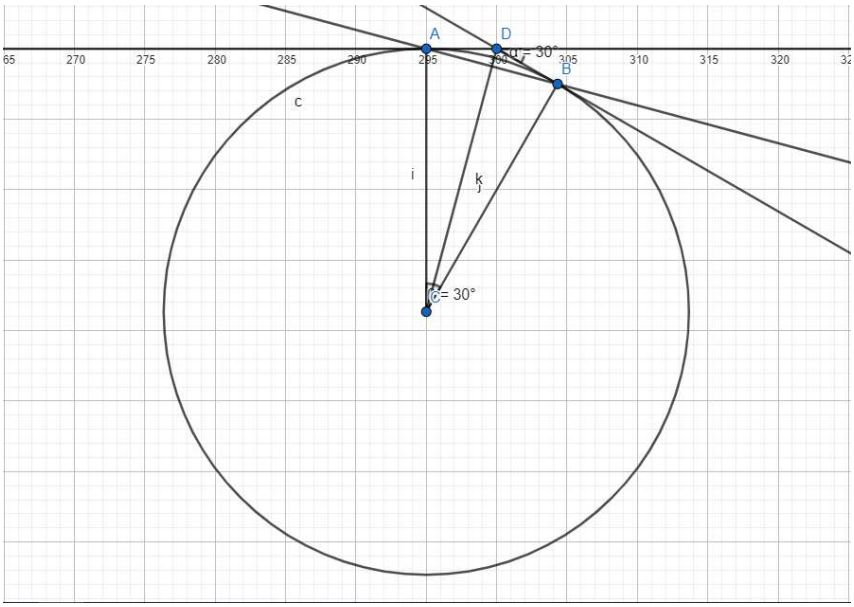
$$x_{3,4} = x_0 \pm \frac{b\, y_a}{a} \tag{14}$$

$$y_{3,4} = y_0 \mp \frac{b\, x_a}{a} \tag{15}$$

Kružnica koja je poželjnija je ona čije je središta udaljenije od vrha raskrižja. To se može vidjeti na primjerima lijevog i desnog zavoja koji su ilustrirani na temelju podataka iz generiranih cestovnih mreža (slika 7 i slika 8). Točka D označava vrh u kojem se dva brida sijeku, a točke A i B su početne i završne točke zavoja, odnosno krajevi cesta koje je potrebno spojiti. Točke presjeka tih dviju cesta uvijek će biti dalje od kružnice jer će zavoj biti unutar trokuta koje čine točke A, B i D.



Slika 7. Primjer lijevog zavoja, nacrtan u GeoGebri [29]



Slika 8. Primjer desnog zavoja, nacrtan u GeoGebri [29]

Točke A, B i D su poznate, a središte C se može izračunati prema (14) i (15) te je jedina preostala nepoznanica kut uz centar kružnice. Pravac koji prolazi točkama A i D, kao i pravac koji prolazi točkama D i B, je tangenta na kružnicu *c*. Svaka tangenta točke na kružnici okomita je na normalu koja spaja tu točku i središte kružnice. Tako su okomiti pravci koji prolaze točkama D i B i točkama B i C, a isto vrijedi i za tangentu u točki A i njenu normalu. Uz to vrijedi da su kutovi s okomitim krakovima sukladni ako su oba tupa ili oba šiljasta, odnosno zbroj im je 180° ako je jedan tupi, a drugi šiljasti. Iz toga se može zaključiti da će kut $\angle ACB$ biti jednak kutu $\angle D$ i jednak 180° umanjeno za vrijednost kuta $\angle BDA$. Nema dovoljno podataka da se izračuna šiljasti kut kod vrha D, no s pomoću kosinusnog poučka [31] može se izračunati tupi kut kod vrha D:

$$\theta = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \tag{16}$$

Stranice a, b, c su redom duljine između vrhova B i D, A i D te A i B. Kut uz središte kružnice sada je $180° - \theta$.

Za izračun svih geometrijskih elementi koristi se standardna Python biblioteka *math*. Kada se dobe sve potrebne vrijednosti, a prije nego se stvore OpenDRIVE klase, potrebno je provjeriti smjer zavoja. Polumjer će uvijek imati pozitivnu vrijednost te će time i $curvature$ iz (9) uvijek biti pozitivan broj (dobit će se lijevi zavoj). Zbog toga se na temelju početne

točke $start = (x_s, y_s)$, srednje točke $middle = (x_m, y_m)$ i završne točke $end = (x_e, y_e)$ određuje skreće li zavoj u lijevo i desno:

$$\sigma = (x_m - x_s)(y_e - y_s) - (y_m - y_s)(x_e - x_s) \tag{17}$$

Ako je vrijednost σ veća od nule, kretanje je suprotno smjeru kazaljke na satu (lijevi zavoj), a ako je manja od nule, luk se kreće u smjeru kazaljke na satu (desni zavoj) [28]. U primjeru nacrtanom u GeoGebri (slika 7 i slika 8) početak bi bio točka A, sredina točka D, a završetak točka B.

Postupak stvaranja ceste isti je kao i prije. Potrebno je definirati element luka koji sadrži zakrivljenost u sebi te geometriju u kojoj se spremaju podaci o točki početka, usmjerenju, duljini i luku. Usmjerenje se računa s obzirom na početnu točku i točku raskrižja (presijecanja dviju cesta) te zapravo odgovara pravcu tangente na kružnicu, a duljina ceste dobije se formulom: $d = r\theta$. Ukupan kôd bi izgledao ovako:

```
arc = RoadView.Arc(curvature)
geometry1 = RoadView.Geometry(0.0, start[0], start[1],
heading, distancecurve, arc=arc)
```

Trake se definiraju na već navedeni način, a susjedi se definiraju tako da je cesta kod početne točke prethodnik, a cesta kod završne točke sljedbenik. Na kraju se postavljaju povezanosti između traka unutar križanja koja objašnjavaju OpenDRIVE standardu preko kojih cesta se spajaju dolazeće ceste u raskrižju.

3. Rezultati

3.1. Pokretanje rješenja

Sav kôd pisan je u integriranom razvojnom okruženju (engl. *IDE*) PyCharm te se s pomoću njega pokreće. PyCharm je alat koji pruža mogućnost uređivanja kôda, automatsko nadopunjavanje, provjeru sintakse i otklanjanje pogrešaka, u području znanosti o podacima i razvoju web aplikacija [34]. U projektu su korištene sljedeće biblioteke u Pythonu: *math* za osnovne matematičke funkcije, *itertools* za stvaranje iteratora kroz podatke, *networkx* za rad s grafovima, *matplotlib* za crtanje grafova i *random* za generiranje nasumičnih brojeva.

Uz sve potrebne datoteke za generiranje cestovne mreže, dodana je još i skripta za pokretanje rješenja u simulatoru CARLA. U njoj se koriste funkcionalnosti biblioteke *carla*, kao što su povezivanje s klijentom i učitavanje svijeta, otvaranje proizvoljne *xodr* datoteke (u ovom slučaju datoteke u kojoj je zapisana cestovna mreža) te učitavanje *xodr* mape u klijentski svijet. Kako je prije objašnjeno, CARLA ima klijent-poslužitelj arhitekturu te su klijenti jedni od najvećih i najbitnijih elemenata tog sustava. Klijenti se mogu povezati s poslužiteljem, dohvaćati podatke i mijenjati svijet simulacije, sve s pomoću skripti [35]. Svaki klijent se definira s pomoću funkcije `carla.Client` koja prima IP adresu i dva TCP priključka koji komuniciraju sa serverom. Nakon toga se može koristiti funkcija `get_world()` kako bi se učitao svijet simulacije. Zajedno bi to izgledalo ovako:

```
client = carla.Client('localhost', 2000)
world = client.get_world()
```

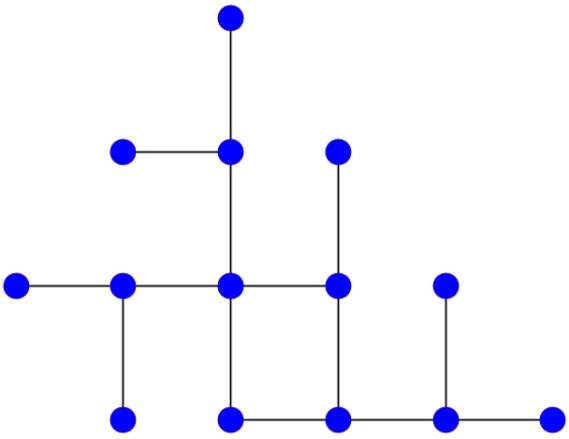
Sljedeći korak je učitavanje *xodr* datoteke. Za to se koristi tipična Python funkcija `open()` koja vraća datotečni objekt nad kojim se onda može pozvati funkcija `read()` i spremiti cijelu datoteku u jednu varijablu. Ta varijabla se prosljeđuje funkciji `generate_opendrive_world()`. Ona stvara novu mapu koja će se prikazivati u simulaciji, uz opcionalne dodatne parametre:

```
world = client.generate_opendrive_world(xodr_xml)
```

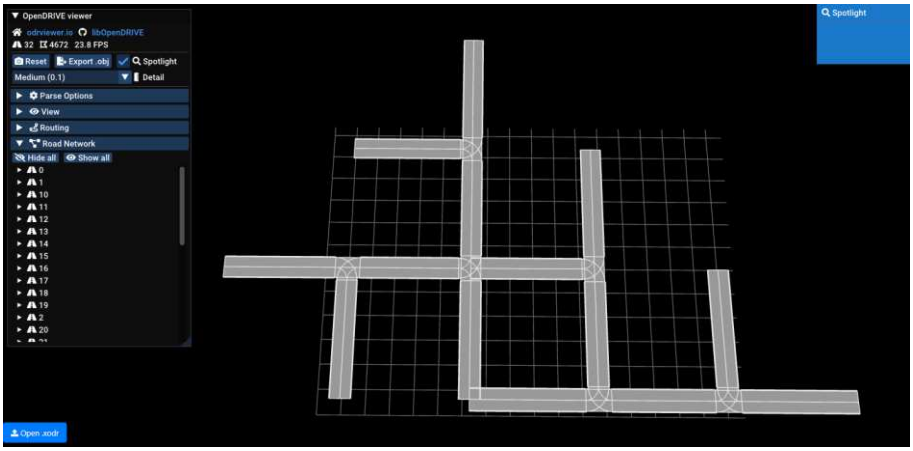
Nakon što je napisana, skripta se pokreće putem naredbenog retka.

3.2. Vizualizacija rezultata

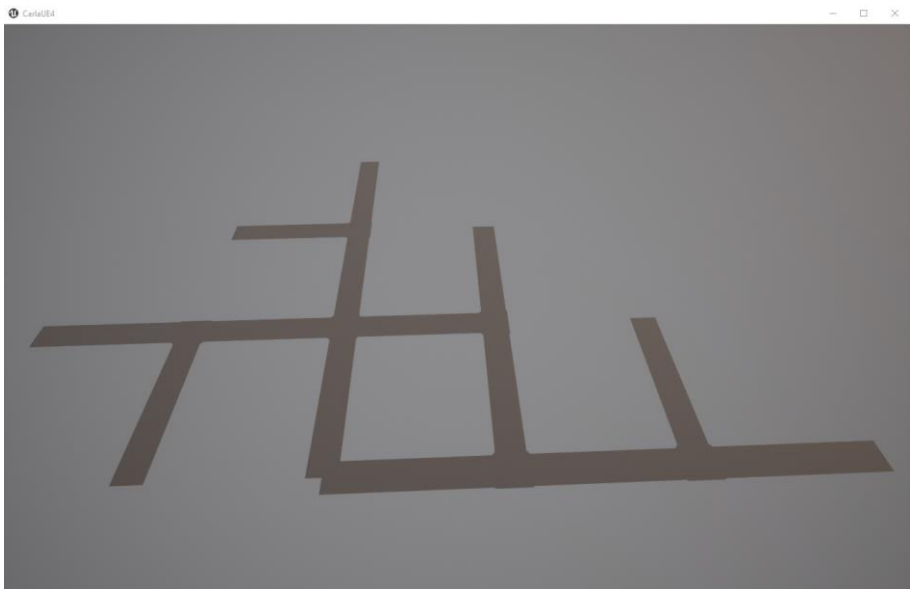
Rezultat implementiranog rješenja je *xodr* datoteka sukladna s OpenDRIVE standardom, no takav format ljudima nije pogodan za interpretaciju. Zbog toga su korištena tri načina vizualizacije rezultata: biblioteka *matplotlib* [32], *OpenDRIVE viewer* web aplikacija [33] i simulator CARLA [2]. *Matplotlib* je Python biblioteka koja omogućuje crtanje nalik MATLAB-u. Ima mnogih primjena u raznim područjima, no ovdje se koristio paket *pyplot* za vizualizaciju dobivenog grafa iz stabla L-sustava. *OpenDRIVE viewer* je aplikacija koja se otvara u pregledniku, a pruža mogućnost učitavanja *xodr* datoteka i prikazivanja cesta, njihovog zapisa, povezanosti i dr. Simulator CARLA je primarno osmišljen za treniranje i testiranje autonomnih vozila, no ceste po kojima automobili voze slijede OpenDRIVE standard te se mogu uvesti proizvoljne *xodr* datoteke. Na slikama 9, 10 i 11 može se vidjeti primjer iste cestovne mreže prikazane u ta tri alata.



Slika 9. Vizualizacija primjera cestovne mreže s pomoću *matplotlib* biblioteke

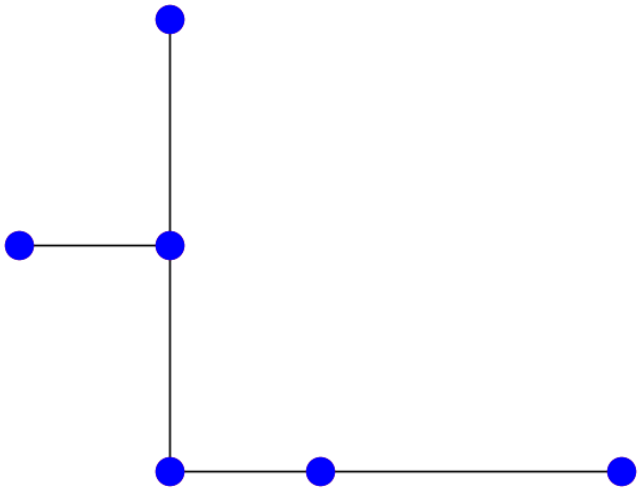


Slika 10. Vizualizacija primjera cestovne mreže s pomoću *OpenDRIVE viewer*-a

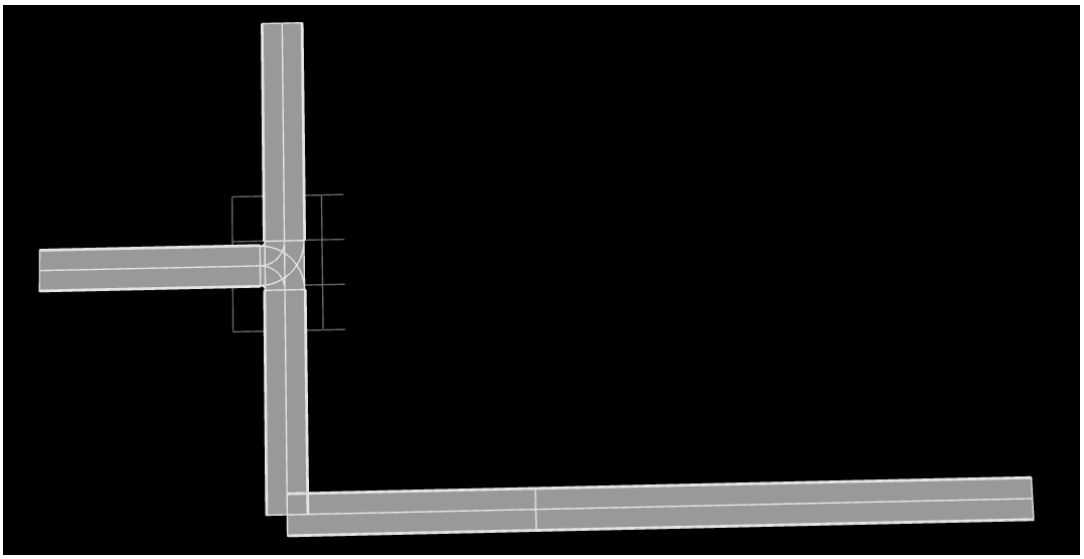


Slika 11. Vizualizacija primjera cestovne mreže s pomoću simulatora CARLA

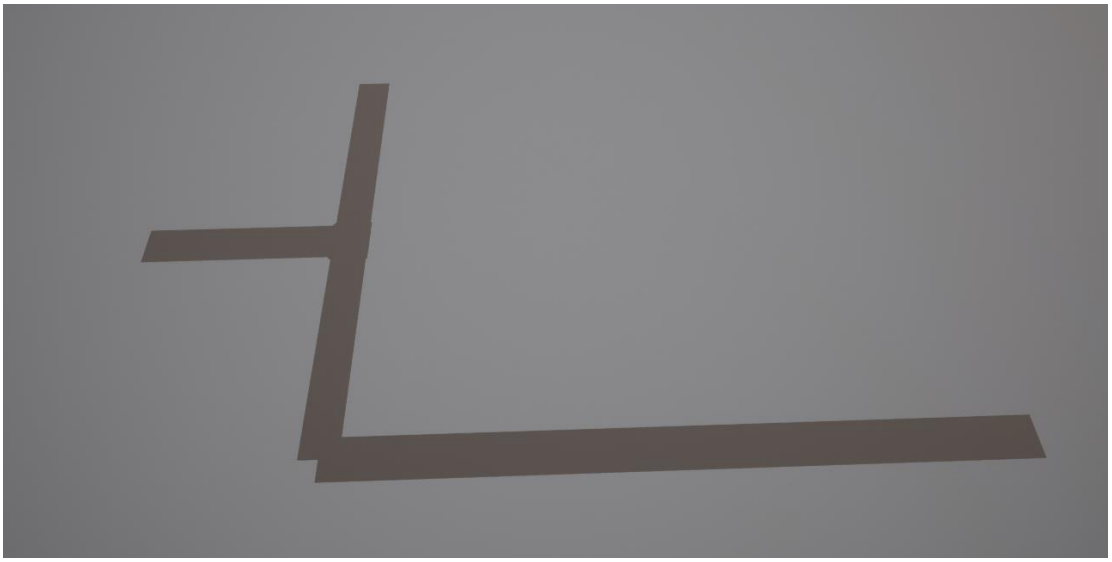
Generiranje cestovne mreže teče kroz određeni broj iteracija. Ako ima malo iteracija, mreža će biti jednostavnija, dok kod puno iteracija može doći do prevelikog preklapanja i eksplodiranja broja vrhova i bridova. Na primjer, ovako izgledaju ulice kada se postupak pokrene s pravilima definiranim u poglavlju 2.1, a broj iteracija se postavi na jedan:



Slika 12. Jednostavni primjer cestovne mreže (broj iteracija = 1) s pomoću *matplotlib*-a

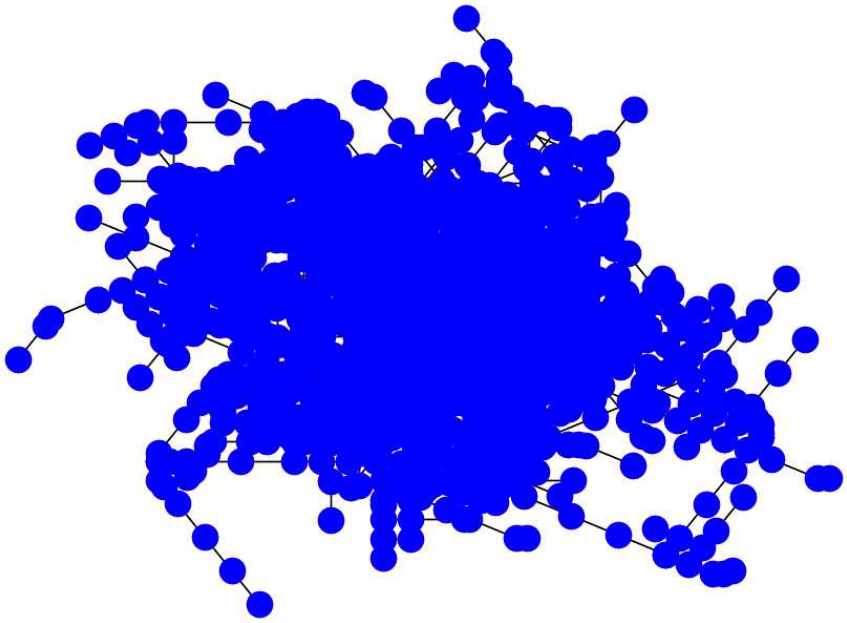


Slika 13. Jednostavni prikaz cestovne mreže (broj iteracija = 1) u *OpenDRIVE viewer*-u

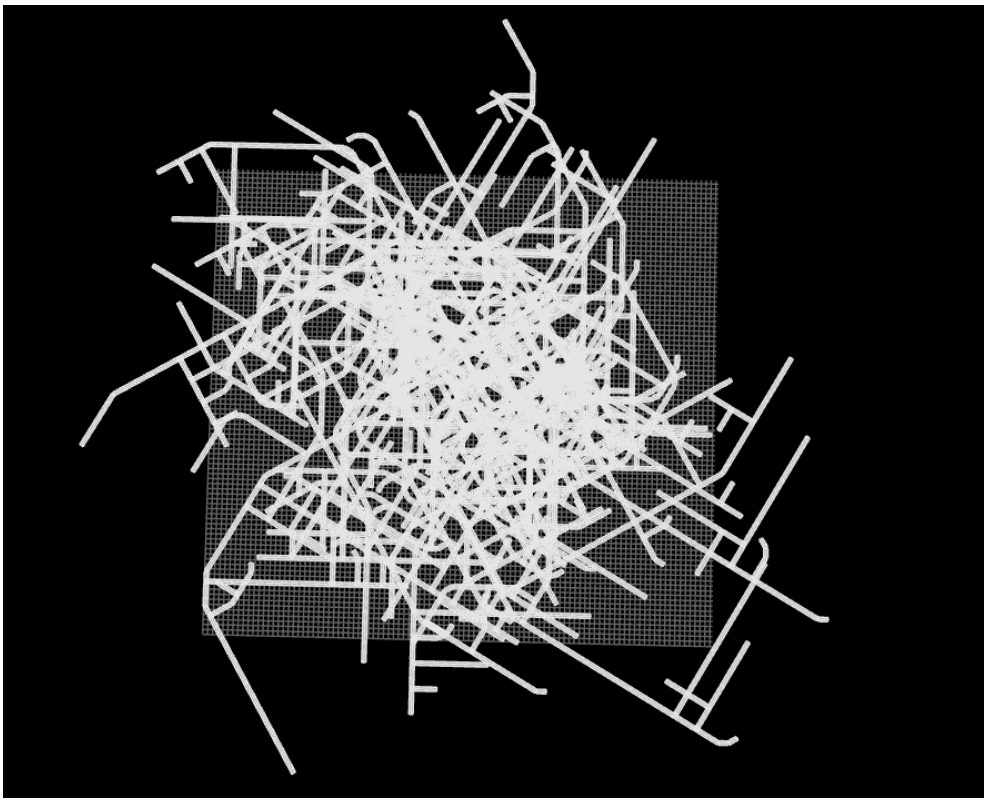


Slika 14. Jednostavni prikaz cestovne mreže (broj iteracija = 1) u simulatoru CARLA

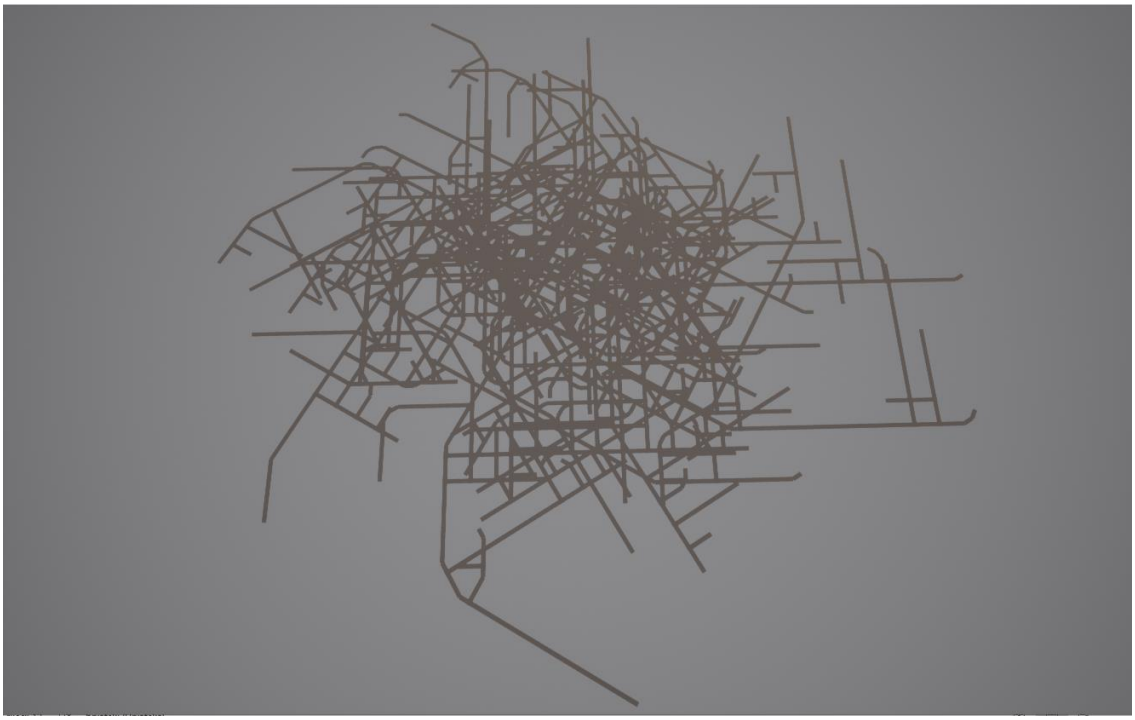
Ako se iteracija postavi na veći broj, recimo dvadeset, dobije se veća i kompliciranija mreža, ali s puno preklapanja i gomilanja:



Slika 15. Prikaz cestovne mreže (broj iteracija = 20) s pomoću *matplotlib*-a

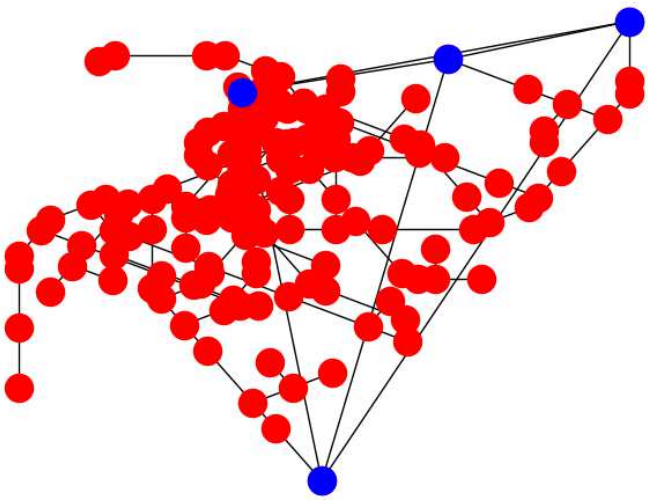


Slika 16. Prikaz cestovne mreže (broj iteracija = 20) s pomoću *OpenDRIVE viewer*-a

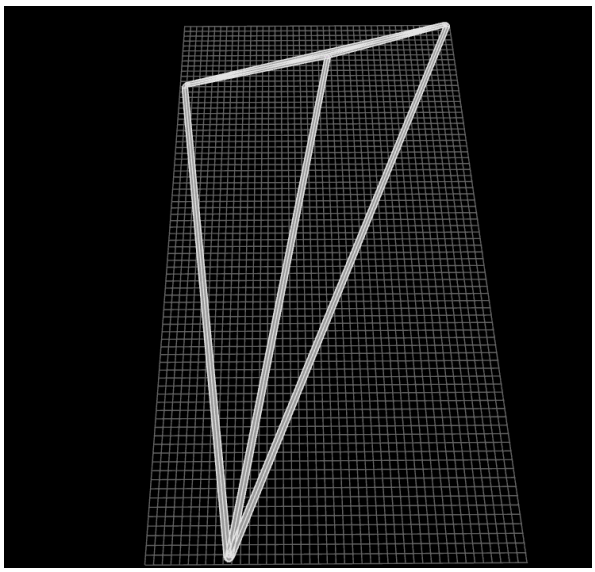


Slika 17. Prikaz cestovne mreže (broj iteracija = 20) s pomoću simulatora CARLA

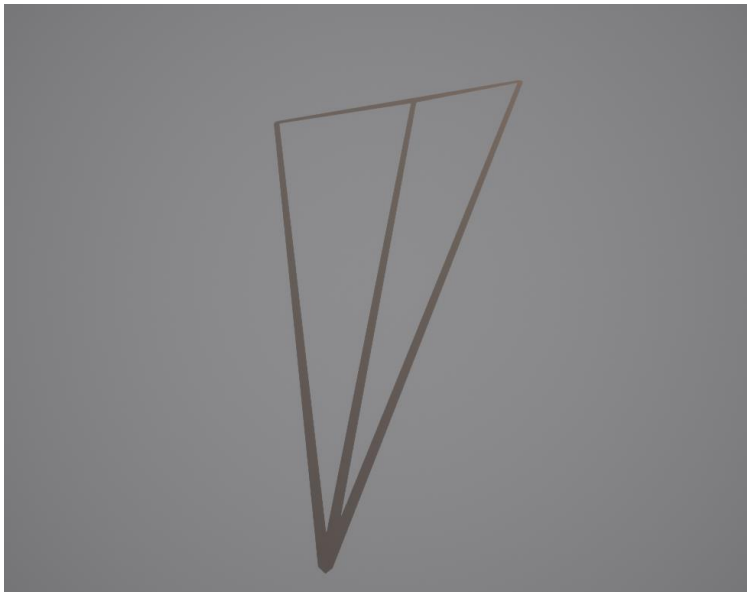
Takav sustav cestovne mreže nije efikasan ni vizualno primamljiv. Zbog toga su dodane dvije funkcije, kako je opisano u poglavlju 2.3: sažimanje grupiranih vrhova i stvaranje raskrižja na sjecištima bridova. Kada se te funkcije ukomponiraju u generiranje mreže, dobe se ovakvi rezultati (prag udaljenosti nakon kojeg se brišu vrhovi postavljen je na dvjesto metara):



Slika 18. Prikaz cestovne mreže (broj iteracija = 10) s pomoću *matplotlib*-a i dodatnih funkcija, gdje crvene točke predstavljaju vrhove prije brisanja, a plave poslije



Slika 19. Prikaz cestovne mreže (broj iteracija = 10) s pomoću *OpenDRIVE viewer*-a



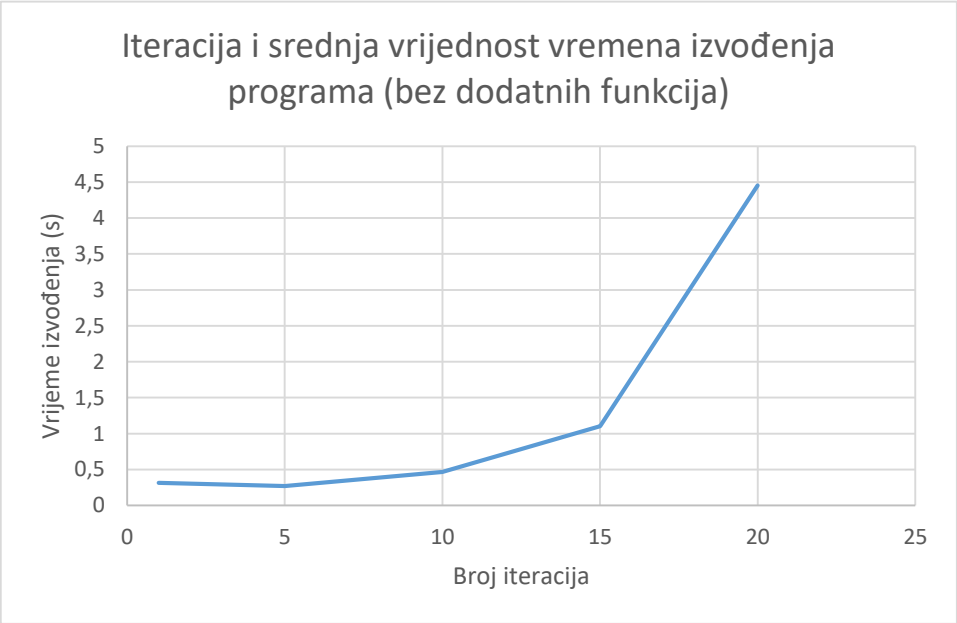
Slika 20. Prikaz cestovne mreže (broj iteracija = 10) s pomoću simulatora CARLA

Broj iteracija utječe na vrijeme izvođenja programa. Što je broj iteracija veći, programu je potrebno više vremena za izvođenje (tablica 1). Budući da se nakon svakog pokretanja dobije nasumičan rezultat, testiranje je provedeno kroz deset pokretanja te je uzeta njihova prosječna vrijednost, uz napomenu da se pokretanje programa izvodilo na računalu s procesorom Intel(R) Core(TM) i5-8250U i 16 GB RAM memorije.

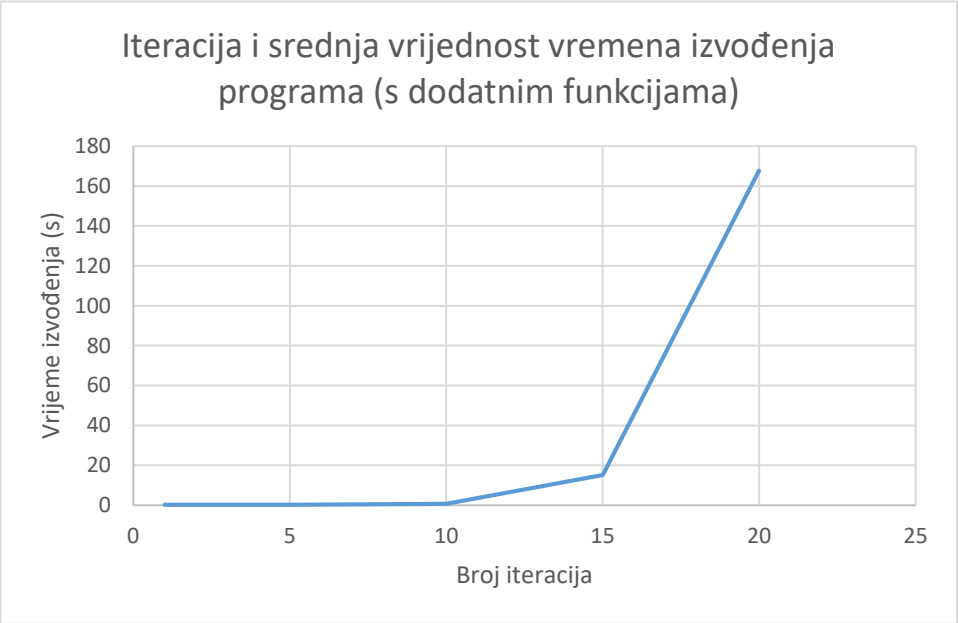
Tablica 1. Odnos broja iteracija i srednje vrijednosti vremena izvođenja

	Iteracija	1	5	10	15	20
Srednja vrijednost vremena izvođenja (s)	Bez funkcija	0.3139	0.2698	0.4662	1.1047	4.4546
	S funkcijama	0.2578	0.2989	0.6785	15.2412	167.7573

Iz tablice 1 te iz grafikona 2 i 3 može se vidjeti da se vrijeme izvođenja programa eksponencijalno povećava s povećanjem broja iteracija. Naravno, kod brisanja čvorova, parametar blizine može se podesiti tako da se produži ili skrati vrijeme izvođenja, ovisno koliko se veliko područje sumiranja odabere (u ovom slučaju 150 metara). Kada se uključe dodatne funkcije, vrijeme trajanja se povećava zato što se izvode dodatne petlje po vrhovima i bridovima grafa te, iako se miču vrhovi koji su višak, kada se obrađuju raskrižja, ceste se udvostručuju, što dodatno opterećuje generiranje.



Grafikon 2. Prikaz broja iteracija naprema srednjoj vrijednosti vremena izvođenja programa bez dodatnih funkcija

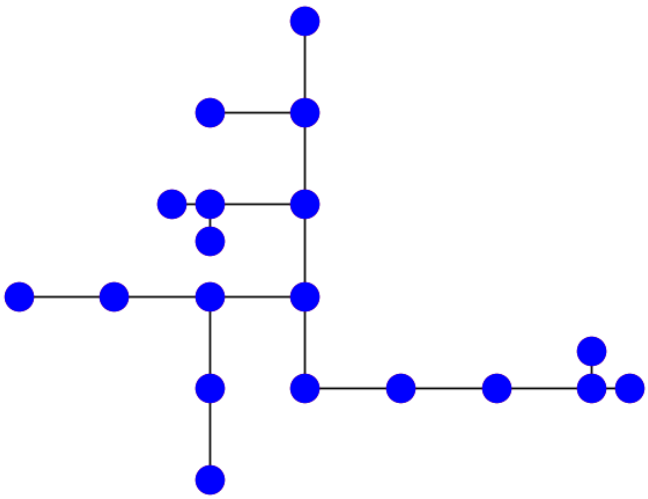


Grafikon 3. Prikaz broj iteracija naprema srednjoj vrijednosti vremena izvođenja programa s dodatnim funkcijama

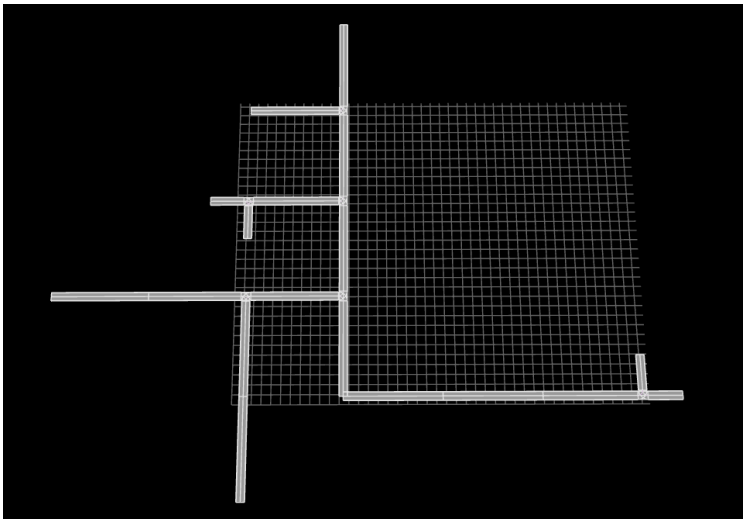
Dodatna varijabilnost je odabir pravila. Ako se napišu pravila koja stvaraju nove ceste pod pravim kutom, mreža će izgledati planski uređena (slika 21, 22 i 23). Suprotno, ako se odaberu pravila koja stvaraju ceste pod nepravilnim kutovima, sustav će izgledati kao da je organski narastao (slika 24, 25 i 26). Na primjer, pomoću pravila

```
rule1 = Rules.Rule(length=100)
rule2 = Rules.Rule("B", "C", "B", 2, 100, 90)
rule5 = Rules.Rule("B", "C", "B", 2, 40, 90)
```

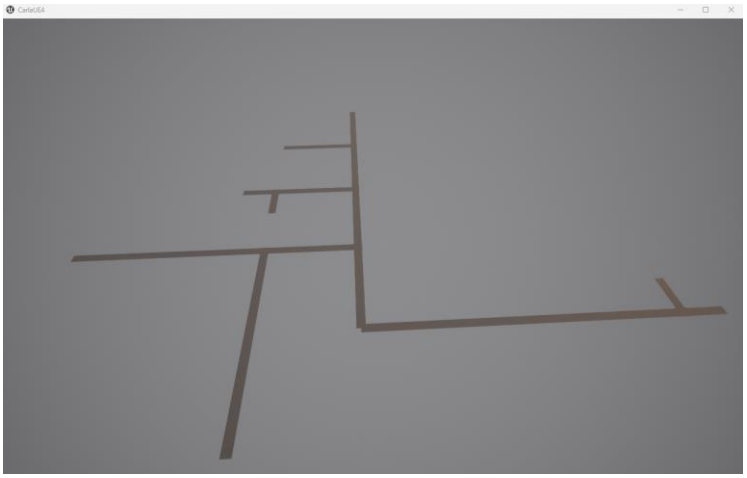
dobit će se cestovna mreža nalik ovoj:



Slika 21. Prikaz planskog izgleda cestovne mreže u *matplotlib*-u



Slika 22. Prikaz planskog izgleda cestovne mreže u *OpenDRIVE viewer*-u

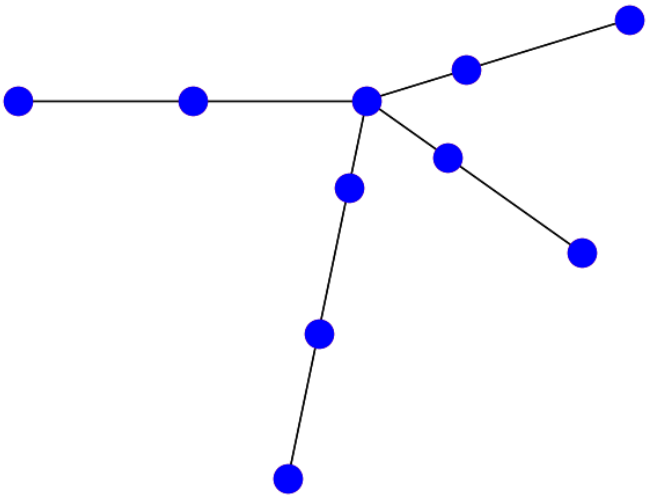


Slika 23. Prikaz planskog izgleda cestovne mreže u simulatoru CARLA

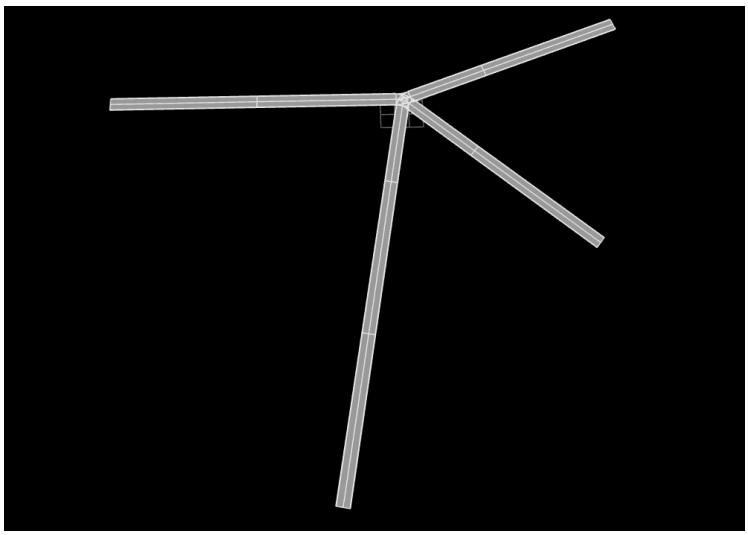
S druge strane, pomoću pravila

```
rule1 = Rules.Rule(length=100)
rule3 = Rules.Rule("B", "C", "B", 3, 60, 30)
rule4 = Rules.Rule("B", "C", "B", 2, 70, 60)
```

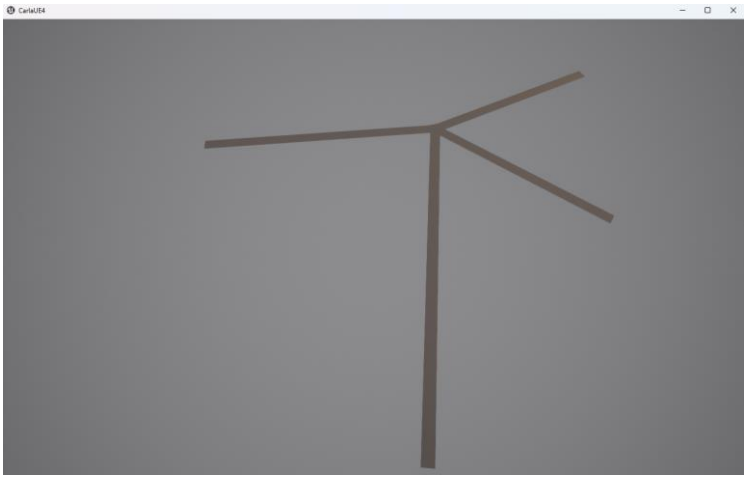
iscrtava se ovakva mreža:



Slika 24. Prikaz neplanskog izgleda cestovne mreže u *matplotlib*-u



Slika 25. Prikaz neplanskog izgleda cestovne mreže u *OpenDRIVE viewer*-u



Slika 26. Prikaz neplanskog izgleda cestovne mreže u simulatoru CARLA

3.3. Proširenja

Proceduralno generiranje je velika tema koja pruža mnogo mogućnosti, no zahtjeva i puno truda i ulaganja. Ovo rješenje ponudilo je tek djelić tog potencijala. ASAM OpenDRIVE također je opsežan standard te bi trebalo puno više proučavanja kako bi se u cijelosti pokrio. Zato će se u ovom poglavlju dati pregled mogućih smjerova razvoja i poboljšanja trenutnih rezultata.

Prvo područje na kojem se mogu primijeniti poboljšanja je sustav pravila. Za sada, pravila zadaju vrstu čvora koje se traži, tip čvora s kojim se zamjenjuje, koliko i koji čvorovi se dodaju te na koju udaljenost i pod kojim kutom. Sljedeći korak je uvesti nasumičnost gdje se na početku zadaje interval duljine ceste i kuta pod kojim će se generirati. Tako neće sve ceste kreirane jednim pravilom biti iste, a samo generiranje moglo bi izgledati prirodnije. Ako se želi postići dojam planski uređenog grada i ulica, mogu se dodati drukčije vrste cesta. Na primjer, gradom bi se prostirala glavna cesta na koju se spajaju manje ulice koje imaju kratke uličice kao završetke. Takve oznake cesta moguće su jer se unutar grafa biblioteke *networkx* mogu bridovima pridjeljivati razni atributi. To bi otvorilo mogućnost manipuliranja broja traka unutar svake ceste gdje bi se moglo, na primjer, glavnoj cesti postaviti četiri trake za svaki smjer prometa, ulici dvije, a uličici jednu. Uz navedene prijedloge, mogao bi se primijeniti utjecaj na nasumičnost. U trenutnom rješenju, pravilo se bira iz uniformne razdiobe Python pseudonasumičnog generatora *random*. Jedino kako se može postaviti favoriziranje je da se u listu pravila više puta postavi isto pravilo pa će imati veću vjerojatnost da će biti izabrano. U budućim poboljšanjima, mogla bi se dodati

funkcionalnost u kojoj pravilo ima dodatan atribut koji govori kolika je šansa da će ga pseudonasumičan generator odabrati te time pridodati važnost svakom pravilu.

Drugo područje koje bi profitiralo optimizacijom je stvaranje i pretraživanje stabla. Trenutno, tijekom stvaranja stabla, kroz čvorove se prolazi rekurzivno dubinskim pretraživanjem stabla (engl. *Depth-first Search*), što znači da se prvo prođe kroz svu djecu-čvorove trenutnog čvora i tek se onda na njega primjenjuje pravilo. Drugi izbor mogao je biti algoritam pretraživanja u širinu (engl. *Breadth-first Search*) u kojem bi se prvo obradili svi čvorovi na istoj razini prije nego bi se krenulo na njihovu djecu. Zbog rekurzije i prirode L-sustava, ipak je odabran DFS algoritam. Tijekom pretraživanja, struktura podataka više nije stablo nego je graf, no svejedno se koristi jednostavna petlja koja iterira po vrhovima ili bridova. Kod generiranja ceste i njenih dijelova, mora se ići po svakom bridu te se tu ne može izbjeći petlja, ali kod micanja nakupljenih čvorova, mogao bi se koristiti neki drugi algoritam pretraživanja grafa koji bi sortirao vrhove po njihovoj udaljenosti i tako brže brisao susjede. Uz efikasnije algoritme, dodatno ubrzanje procesa generiranja omogućila bi paralelizacija. Traženje najbližeg susjeda moglo bi se podijeliti u dretve u kojoj svaka uzima jedan čvor i uspoređuje ga s drugima. Osim toga, brže rješenje bi bilo da se svaka cesta obrađuje u svojoj dretvi te se na kraju samo spoje u glavni element `OpenDRIVE`, budući da poredak zapisa cesta nije bitan. U trenutku kada se uvede paralelizam, može se uključiti i grafičke procesore (engl. *Graphics Processing Unit*, GPU) koji simultano odrađuju zadatke.

Treće područje na kojem se može proširiti trenutno rješenje je standard OpenDRIVE. OpenDRIVE definira puno više elemenata nego što je korišteno u ovom projektu te bi se tako u sâmu cestovnu mrežu mogle dodati ostale vrste raskrižja (izravno i virtualno), povezati raskrižja tako da tvore kružni tok, uvesti razne objekte kao što su tuneli, mostovi, mjesta za parkiranje i pješački prijelazi, postaviti signale na cesti (semafori, prometni znakovi i oznake na cesti) te modelirati cestovne tvorevine kojima se služe vozila na tračnicama (vlakovi i tramvaji). Trenutno rješenje implementirano je u dvodimenzionalnom prostoru, no OpenDRIVE nudi mogućnost uzdizanja ceste, odnosno omogućuje trodimenzionalnosti. S pomoću elementa `elevation` cestu se može povisiti uzduž referentne linije. Također, površina ceste može se detaljno opisati u OpenCRG formatu koji se može ukomponirati u OpenDRIVE u obliku `CRG` elementa unutar `surface` elementa. OpenCRG ne sadrži vizualni izgled cestovne površine, već informacije o atributima ceste, kao na primjer je li od asfalta ili je kaldrma i ima li rupe [3].

Konačno, uz predložena poboljšanja, rješenje bi se moglo proširiti na cijelu scenu. Proširenje bi uključivalo dodavanje zgrada, kuća, parkova i trgova, čime se stvara realističnija i složenija okolina, koja je više nalik gradu i koja daje više povratnih informacija senzorima modela autonomnih vozila.

Zaključak

Simulacije su iznimno korisne i bitne prilikom treniranja i testiranja modela autonomnih vozila. Općenito, za složene modele strojnog učenja potrebno je puno podataka na kojima bi mogli kvalitetno učiti, a podaci za validaciju moraju biti neviđeni kako ne bi došlo do prenaučenosti. Iako je jednom gustom, ručno iscrtanom cestom moguće pokriti najčešće scenarije vožnje, bilo bi korisno moći generirati svaki puta drukčiji neviđen grad, odnosno cestovnu mrežu. Zbog toga se u ovom radu istražilo proceduralno generiranje i njegova primjena u stvaranju cestovne mreže u OpenDRIVE formatu.

Nakon što se proučio ASAM OpenDRIVE format, teorijska pozadina L-sustava i proceduralnog generiranja te se u Python jeziku napisao odgovarajući kôd, mogu se analizirati prednosti i nedostaci implementiranog rješenja. U cijelosti, sustav je lak za nadograđivanje zbog modularnosti i funkcionalnosti koje su odijeljene u zasebne datoteke. Želi li se modificirati stvaranje stabla s pomoću L-sustava, dovoljno je promijeniti klasu `LSystemGenerator`. S druge strane, ako se želi promijeniti ili dodati način ispisa u *xodr* formatu, može se manipulirati datotekama u kojima su definirani dijelovi OpenDRIVE standarda. Također je moguće dodati nove funkcije koje mogu djelovati na *networkx* graf unutar klase `Graph`. Uz dodavanje i mijenjanje sâmog procesa, sustav je personaliziran i pruža veliki utjecaj na generaciju cestovne mreže s pomoću pravila. Korisnik zadaje pravila proizvoljno i prema njima dobiva određene rezultate. Kada se primijene odgovarajuća pravila, mogu se postići kreativne ili uvjerljive mreže cesta. Nasumičnost nudi bogatu količinu mreža u kojoj nijedna nije ista kao druga. U teoriji, ovaj sustav bi mogao davati beskonačno mnogo rješenja, no zbog pseudonasumičnog generatora brojeva i ograničenog broja pravila i generiranih simbola to nije moguće. Iako proceduralno generiranje cestovne mreže daje obećavajuće rezultate i nudi veliko područje primjene, treba se dotaknuti i njegovih nedostataka. Glavni manjak u primjeni L-sustava za proceduralnog generiranje cesta, kako tvrdi [19], je ponavljanje cestovnih segmenata koje vozač, odnosno model, može lako naučiti napamet, dok je za procese istraživanja ponašanja vozača bolja nepoznata putanja ceste. Osim toga, L-sustavi su zamišljeni kao matematička teorija razvoja biljaka i dijeljenja stanica te ne stvaraju sami po sebi cikluse, koji su česti u cestovnim mrežama. U ovom rješenju to se moralo implementirati tako da se prepoznaju sjecišta stranica grafa i tako stvaraju ciklusi, što je dodatno računski opteretilo algoritam. Nadalje, odnos između vremena izvođenja generiranja ceste i iteracija može negativno utjecati na performanse

sustava jer se trajanje programa povećava s porastom broja iteracija. Jasno je da je broj iteracija povezan i sa složenošću cestovne mreže te vrijeme generiranja ceste može biti toliko veliko da uspori ili onemogući cijeli proces.

Kako bi se uklonili navedeni nedostaci i poboljšala trenutna svojstva rješenja, u budućem radu može se nadograditi sustav pravila, optimizirati izvedba algoritama, uključiti dodatne elemente ASAM OpenDRIVE standarda i proširiti proceduralno generiranje na cijelu scenu, kako je opisano u poglavlju 3.3. Područje proceduralnog generiranja pokazuje se obećavajuće i vrijedno istraživanja te otvara puno mogućnosti nadogradnje u simulacijama, računalnim igrama, virtualnoj stvarnosti, urbanističkom planiranju i obrazovanju.

Literatura

[1] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V. *CARLA: An Open Urban Driving Simulator*. 1st Annual Conference on Robot Learning, (2017.), str. 1-16.

[2] CARLA, *CARLA Simulator*. Poveznica: https://carla.readthedocs.io/en/latest/start_introduction/; pristupljeno 7. ožujka 2024.

[3] ASAM, *ASAM OpenDRIVE*, 3. kolovoza 2021.

[4] Brol, A., Antoniuk, I. *Procedural Generation of Virtual Cities*. 24th International Conference on Computational Problems of Electrical Engineering (CPEE), Grybów, Polska (2023.), str. 1-4.

[5] Jormedal, M. *Procedural Generation of Road Networks Using L-Systems*. Diplomski rad. Linköpings universitet Department of Electrical Engineering, 2013.

[6] Cheng, H. *Autonomous Intelligent Vehicles: Theory, Algorithms, and Implementation*. Serijal: *Advances in Computer Vision and Pattern Recognition*. 1. izdanje. London: Springer London, 2011.

[7] CARLA, *Sensors and Data*. Poveznica: https://carla.readthedocs.io/en/latest/core_sensors/; pristupljeno 9. ožujka 2024.

[8] Jo, K., Sunwoo, M. *Generation of a Precise Roadway Map for Autonomous Cars*. IEEE Transactions on Intelligent Transportation Systems, 15,3 (2014.), str. 925-937.

[9] Chiang, K.-W., Pai, H.-Y., Zeng, J.-C., Tsai, M.-L., El-Sheimy, N. *Automated Modeling of Road Networks for High-Definition Maps in OpenDRIVE Format Using Mobile Mapping Measurements*. Geomatics, 2, 2 (2022.) str. 221–235. https://doi.org/10.3390/geomatics2020013

[10] Sepp, E. *Creating High-Definition Vector Maps for Autonomous Driving*. Diplomski rad. University of Tartu Institute of Computer Science Conversion Master in IT Curriculum, 2021.

[11] Bender, P., Ziegler, J., Stiller, C. *Lanelets: Efficient Map Representation for Autonomous Driving*. IEEE Intelligent Vehicles Symposium (IV), Dearborn, Michigan, USA (2014.), str. 420–425.

[12] Godoy, J., Artuñedo, A., Villagra, J. *Self-Generated OSM-Based Driving Corridors*. IEEE Access. 7, (2019.), str. 20113-20125.

[13] OpenStreetMap contributors *OpenStreetMap (OSM) XML Reader/Writer*, FME. Poveznica: https://docs.safe.com/fme/html/FME-Form-Documentation/FME-ReadersWriters/osm/osm.htm; pristupljeno 14. ožujka 2024.

[14] OpenStreetMap contributors *OpenStreetMap*. Poveznica: https://www.openstreetmap.org/about; pristupljeno 16. Ožujka 2024.

[15] Zhang, Y., Zhang, G., Huang, X. A *Survey of Procedural Content Generation for Games*. International Conference on Culture-Oriented Science and Technology (CoST), Lanzhou, China (2022.), str. 186-190.

[16] Chelladurai, J. *Exploring Complex Toroidal Mazes with L-systems*. 4th International Conference on Computing and Communication Systems (I3CS), Shillong, Meghalaya, India (2023.), str. 1-6.

[17] Srbljić, S. *Jezični procesori 1: Uvod u teoriju formalnih jezika, automata i gramatika*. 1. izdanje. Zagreb: Element, 2000.

[18] Prusinkiewicz, P., Lindenmayer, A., Hanan et al. *The Algorithmic Beauty of Plants*. 1. izdanje. New York: Springer-Verlag, 1990.

[19] Campos, C., Leitão, J. M., Pereira, J. P., Ribas, A., Coelho, A. F. *Procedural generation of topologic road networks for driving simulation*. 10th Iberian Conference on Information Systems and Technologies (CISTI), Águeda, Aveiro, Portugal, (2015.), str. 1-6.

[20] Jo, K., Sunwoo, M. *Generation of a Precise Roadway Map for Autonomous Cars*. IEEE Transactions on Intelligent Transportation Systems. 15, 3 (2014.), str. 925-937.

[21] Brol, A., Antoniuk, I. *Procedural Generation of Virtual Cities*. 24th International Conference on Computational Problems of Electrical Engineering (CPEE), Grybów, Poland, (2023.), str. 1-4.

[22] Sharma, R. *Procedural City Generator*. International Conference System Modeling & Advancement in Research Trends (SMART), Moradabad, India, (2016.), str. 213-217.

[23] Parish, Y. I. H., Müller, P. *Procedural modeling of cities*. 28th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, Kalifornija, SAD, (2001.), str. 301–308.

[24] Hagberg, A. A., Schult, D. A., Swart, P. J. *Exploring network structure, dynamics, and function using NetworkX*. 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), Pasadena, Kalifornija, SAD, (2008), str. 11–15

[25] By Ag2gaeh - Own work, CC BY-SA 4.0, Poveznica: https://commons.wikimedia.org/w/index.php?curid=44999745; pristupljeno 3. lipnja 2024.

[26] Klein, R.E. *Teaching linear systems theory using Cramer's rule*. IEEE Transactions on Education. 33, 3 (1990), str. 258-267

[27] David K. *Finding the Center of a circle given two points and a radius (algebraically)*, Mathematics Stack Exchange, (2017, travanj). Poveznica: https://math.stackexchange.com/q/1781546; pristupljeno 12. svibnja 2024.

[28] Blatter, C. *How can I determine whether an arc is clockwise or anti-clockwise?* Mathematics Stack Exchange, (2015, veljača). Poveznica: https://math.stackexchange.com/q/1166492; pristupljeno 14. svibnja 2024.

[29] GeoGebra. Poveznica: https://www.geogebra.org/; pristupljeno 4. travnja 2024.

[30] Draw IO. Poveznica: https://www.drawio.com/; pristupljeno 10. lipnja 2024.

[31] kosinusni poučak. *Hrvatska enciklopedija, mrežno izdanje*. Leksikografski zavod Miroslav Krleža, (2013 – 2024). Poveznica: https://www.enciklopedija.hr/clanak/kosinusni-poucak; pristupljeno 10. lipnja 2024.

[32] Hunter, J., Dale, D., Firing E., Droettboom, M., Matplotlib razvojni tim. *matplotlib*. Poveznica: https://matplotlib.org/stable/; pristupljeno 11. lipnja 2024.

[33] OpenDRIVE viewer. Poveznica: https://odrviewer.io/; pristupljeno 8. ožujka 2024.

[34] PyCharm. Poveznica: https://www.jetbrains.com/pycharm/; pristupljeno 12 lipnja 2024.

[35] CARLA, *1st. World and client*. Poveznica: https://carla.readthedocs.io/en/latest/core_world/; pristupljeno 12. lipnja 2024.

[36] Pandžić, I.S., Pejša, T., Matković, K., Benko, H., Čereković, A., Matijašević, M. *Virtualna okruženja: Interaktivna 3D grafika i njene primjene*. 1. izdanje. Zagreb: Element, 2011.

Sažetak

Proceduralno generiranje cestovne mreže

Rad opisuje problematiku treniranja modela autonomnih vozila i simulacija gradske vožnje, točnije simulatora CARLA. Kako bi se pružio veći broj raznovrsnijih cestovnih mreža uveden je pojam proceduralnog generiranja. Proceduralno generiranje omogućuje stvaranje rezultata s pomoću algoritma i uz određena pravila. Pravila i algoritam mogu se definirati na puno načina, a u ovom radu proučava se potencijal i utjecaj Lindenmayerovih sustava (L-sustava). Dobivena cestovna mreža prikazuje se u obliku stabla, koji je pogodniji za sâmo stvaranje, i grafa, koji prikazuje cikluse i jednostavnije pretražuje vrhove i bridove, a rezultat se na kraju sprema u *xodr* datoteku sukladnu ASAM OpenDRIVE standardu. Implementacija rješenja izvedena je korištenjem programskog jezika Python. U završnom dijelu rada opisani su i prikazani rezultati zajedno s mogućim poboljšanjima te prednosti i nedostaci implementiranog postupka.

Ključne riječi: proceduralno generiranje; simulator CARLA; ASAM OpenDRIVE; L-sustavi; cestovna mreža

Summary

Procedural Generation of Road Network

The thesis describes the challenges of training autonomous driving models and urban driving simulations like the CARLA simulator. To provide a greater variety of road networks, the concept of procedural generation was introduced. Procedural generation allows creation of results using an algorithm and specific set of rules. The rules and algorithm can be defined in many ways, and this paper explores the potential and impact of Lindenmayer systems (L-systems). The resulting road network is represented as a tree, which is more suitable for creation process, and graph, which shows cycles and allows for simpler traversal of vertices and edges. The result is then saved in the *xodr* file compliant with the ASAM OpenDRIVE standard. Programming language Python is used for implementation. In the final part of the thesis, the results are described and presented, along with possible improvements and the advantages and disadvantages of the implemented procedure.

Keywords: procedural generation; CARLA simulator; ASAM OpenDRIVE; L-systems; road networks

Privitak

Cijelom kôdu koji je korišten za implementaciju može se pristupiti preko Github repozitorija: https://github.com/antonijaengler/masters_thesis