SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5883

SIMULACIJA AUTONOMNE VOŽNJE AUTOMOBILA

Simon Grgurina

Zagreb, siječanj 2019

Sadržaj

1. Uvod

Autonomna vozila rastuća su industrija koja u zadnje vrijeme dobiva puno pozornosti. S projektima poput Teslinog autopilota i Google-ovog samovozećeg automobila popularnost autonomnih vozila se drastično povećala. Postoji više koristi autonomnih vozila, od boljeg upravljanja prometom i ekonomičnije vožnje do smanjivanja broja nesreća eliminiranjem ljudske greške. Osim povećanja ekonomičnosti koja dolazi od "glađe" vožnje, bolje upravljanje prometom smanjivalo bi broj prometnih gužva i također povećalo ekonomičnost. Ove prednosti uštedjeli bi ogromne količine novca u cijenama goriva i materijalnim štetama prometnih nesreća. Osim primjene u običnoj vožnji, postoje inicijative za stvaranje autonomnih vozila namijenjena za utrke. Primjeri takvih inicijativa su Roborace[1] i Self Driving Cars[2].

Ovaj završni rad, bavit će se uporabom autonomnih vozila i ostvarivanjem najboljih vremena na trkaćim stazama. Rad će biti ostvaren u Unreal Engine 4 grafičkom pogonu.

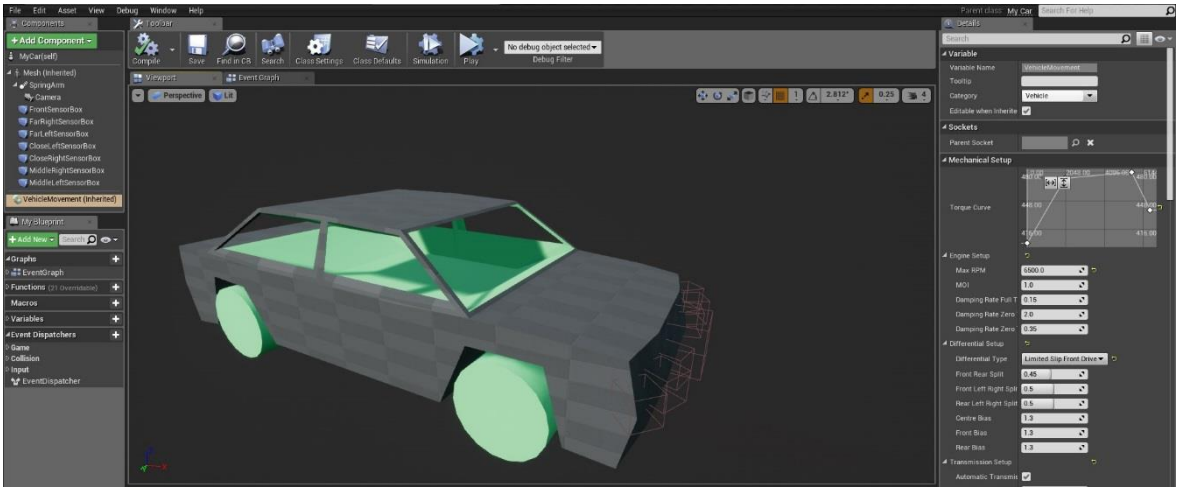
2. Unreal Engine

Unreal Engine je grafički pogon koji je napravila tvrtka Epic Games. To je grafički pogon najviše razine kvalitete koji je dostupan za besplatno, odnosno za pet posto profita ako izrađeni proizvod zaradi više od tri tisuće dolara. Baš zbog toga je često korišten, kako od strane studenata i amatera, tako i od kompanija koje proizvode veoma uspješne videoigre.

U ovom završnom radu korištena je četvrta inačica grafičkog pogona, koja je objavljena u svibnju 2012. godine. U izradi završnog rada korištena je kombinacija vizualnog skriptiranja nacrta (engl. Blueprints) koje nudi Unreal Engine, te C++ koda. Neuronska mreža i genetski algoritam ostvareni su C++ kodom, dok su automobil, staza te sustav treniranja ostvareni korištenjem nacrta.

3. Izrada automobila

Za izgled automobila korišten je model napravljen od strane Epic Games-a [3]. Model se uvozi u Unreal Engine iz fbx formata, koji se koristi za prijenos datoteka između različitih programa za rad s digitalnim trodimenzionalnim objektima. Prilikom uvoza generira se kosturska mreža (engl. Skeletal Mesh), objekt kostura (engl. Skeleton), te fizički objekt (engl. Physics Asset). Nakon podešavanja fizičkog objekta stvara se nacrt pomoću klase “Wheeled Vehicle“.

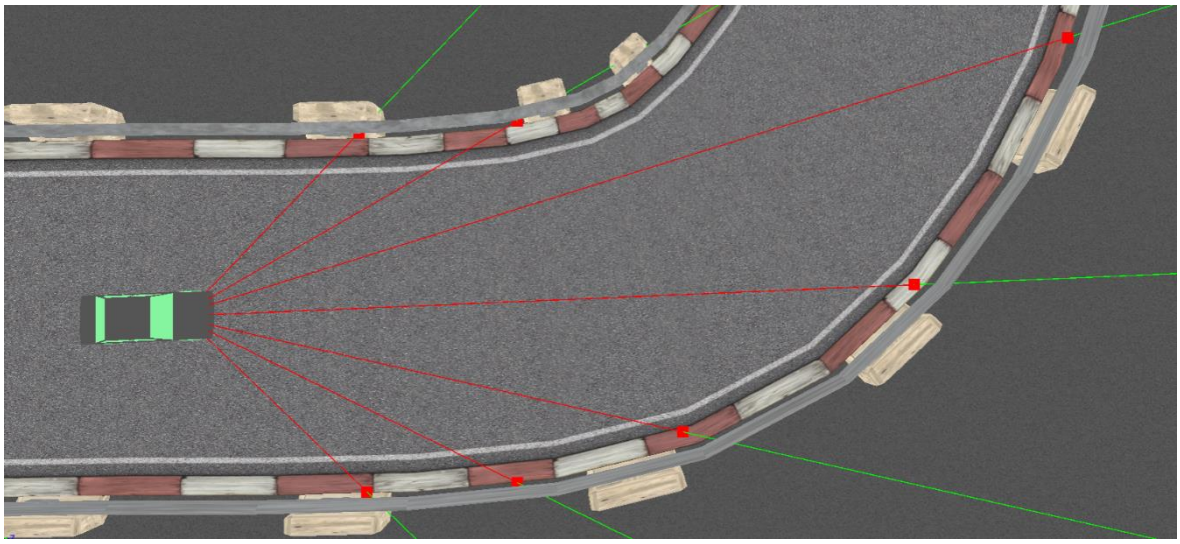


Slika 1. Nacrt automobila

Na slici 1 vidimo nacrt koji će predstavljati naš automobil i sadržavati komponente mreže i kretanja vozila (engl. VehicleMovement). Dodajemo kameru, te pomoću klase “VehicleWheel“ kotače, te kontrole vozila. Nakon dodavanja animacije kotača dobivamo potpuno funkcionalan automobil koji se može upravljati od strane korisnika. Naknadno ćemo trebati i omogućiti neuronskoj mreži da preuzme kontrolu nad vozilom.

3.1 Senzori udaljenosti

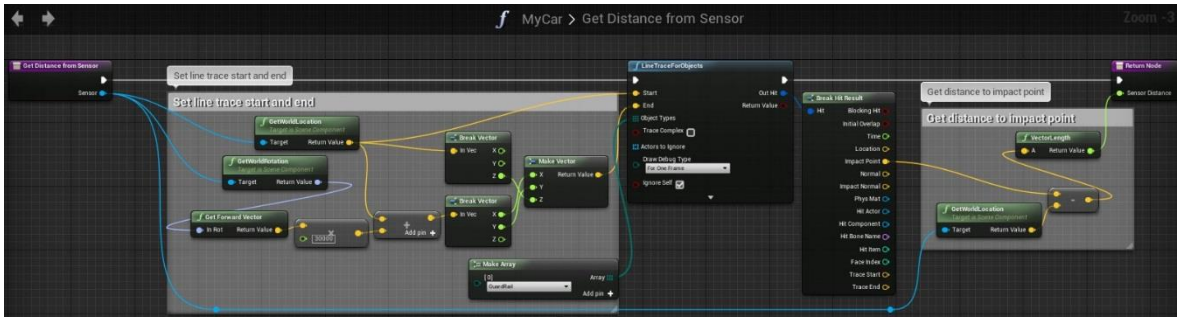
Kako bi neuronska mreža mogla upravljati automobilom ona će morati primati nekakve informacije o okolišu u kojem se nalazi. U tu svrhu će automobil imati senzore koji će mu javljati udaljenost od ograda staze. Njihov prikaz vidimo na slici 2.



Slika 2. Prikaz rada senzora

Senzori projiciraju zraku u prostor ispred sebe te prilikom sudara (engl. Collision) s objektima vraćaju informacije o sudaru. Automobil ima sedam senzora pod različitim kutovima u intervalu od -45 do +45 stupnjeva u odnosu na smjer automobila. Kako će neuronska mreža biti korištena za trkaću stazu, automobilu neće biti važno ono što je već prošao. Broj senzora odabran je kako bi automobil imao dovoljno informacija o duljini ravnih dijelova staze te o nadolazećim zavojima i njihovoj oštrini.

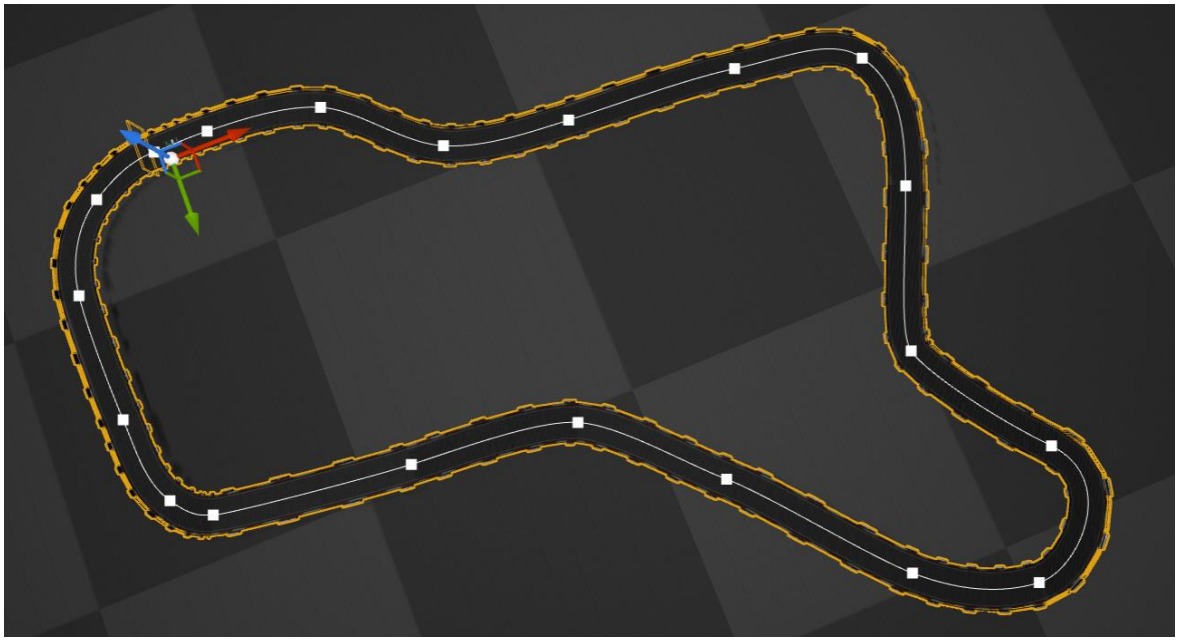
Implementacija koristi čvor “LineTraceForObjects“ koji prepoznaje samo sudare s njemu zadanim vrstama objekata. Važnost vrsta objekata biti će detaljnije objašnjena u poglavlju o sudarima. Od informacija koje čvorovi vraćaju o sudarima tu se koriste lokacije sudara kako bi se izračunale tražene udaljenosti. Implementacija u vizualnom skriptnom sustavu prikazana je na slici 3.



Slika 3. Implementacija funkcije za dohvaćanje vrijednosti određenog senzora

4. Izrada staze

Izrada staze vrši se pomoću generatora staza koji dinamički stvara kružnu stazu na osnovi proizvoljnog broja točaka koje mu postavimo. Između svake dvije zadane točke stvara se segment staze koji se sastoji od tri statičke mreže: cesta, lijeva ograda, i desna ograda. Statičke mreže te teksture i materijali potrebni za ostvarivanje segmenata ceste javno su dostupni od strane Epic Games-a zajedno sa uputama [4]. Slika 4 prikazuje stazu s dvadeset točaka.



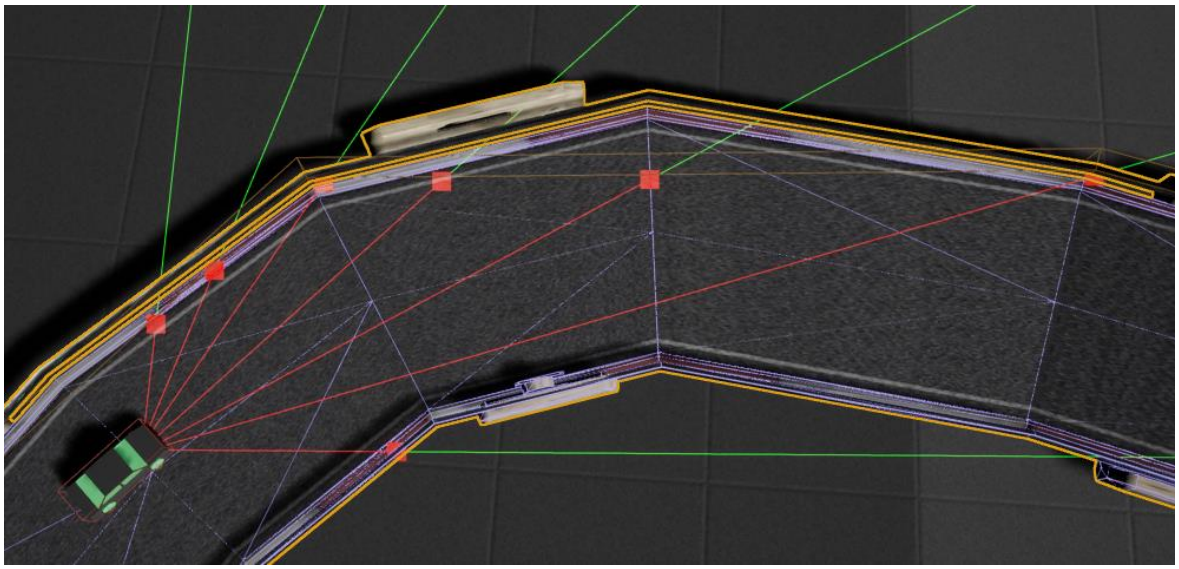
Slika 4. Primjer korištenja generatora staze

Generator staze sadrži dvije dodatne komponente, jedna predstavlja startno ciljnu liniju a druga predstavlja poziciju na kojoj automobil kreće, i na koju se vraća nakon završenog kruga. Startno ciljna linija ostvarena je komponentom “Box Collision” postavljena na ”OverlapAllDynamic”, zbog toga automobil neće imati fizičku reakciju na sudar sa startno ciljnom linijom, ali možemo koristiti događaje (engl. Events) prilikom sudara.

5. Sudari

Unreal Engine pri određivanju rezultata sudara ili preklapanja objekata koristi “kanale” (Object channels i trace channels). Oni omogućavaju klasificiranje objekata koje koristimo. Svakome objektu možemo pojedinačno postaviti reakciju na svaki od tih kanala. Kako bi naš automobil mogao prepoznati kada je udario u ogradu staze i kada se samo vozi cestom, on mora razlikovati stazu od ograda. Zbog toga napravimo novi “object channel” te osiguramo da pri konstrukciji staze postavimo ogradu na novostvoreni kanal.

Za osiguravanje prepoznavanja sudara s ogradama staze važno je i postaviti složenost sudara (engl. Collision complexity) na objektima od kojih se konstruira ograda u generatoru staze. Ako se složenost ne postavi na “Use complex collisions as simple” dolazi do velikih odstupanja između ograda staze i lokacija gdje dolazi do sudara.



Slika 5. Prikaz razlike između postavki složenosti

Slika 5 prikazuje dio staze, gornja ograda koristi postavku “Simple and Complex” dok donja ograda koristi postavku ”Use complex collisions as simple”. Crvene točke pokazuju gdje senzor detektira rub staze, i gdje bi se sudar a rubom staze dogodio. Očito je kako dolazi do odstupanja, od manjih odstupanja na tri lijeva senzora, do velikog odstupanja na sljedeća dva senzora. U kontrastu s time, lokacija ruba koju krajnji desni senzor detektira se savršeno poklapa s ogradom staze.

6. Neuronske Mreže

6.1 O neuronskim mrežama

Ljudski mozak sastoji se od mreže ogromnog broja povezanih neurona. Svaki neuron sastoji se od tijela, velikog broja dendrita preko kojih prima signale i jednog aksona preko kojih ih šalje drugim neuronima. Signali u mozgu se prenose između neurona preko sinapsa koje spajaju neurone.

Umjetne neuronske mreže programska su struktura inspirirana strukturom mozga. Tijelo neurona je predstavljeno zbrajalom ulaznih signala, jakosti sinapsa s težinskim funkcijama te aksoni aktivacijskom funkcijom. Neuroni u mreži su organizirani u tri vrste slojeva, pri čemu je veličina svih slojeva proizvoljna te se prilagođava problemu koji mreža rješava. Ulazni sloj prima ulazne vrijednosti i prosljeđuje ih dalje u mrežu. Između ulaznog i izlaznog sloja nalazi se proizvoljan broj skrivenih slojeva koji obrađuju ulazne vrijednosti dok izlazni sloj vraća izlazne vrijednosti koje nam neuronska mreža izračunava poput realnih brojeva ili klase u klasifikacijskom problemu.

Jedan neuron prima ulazne vrijednosti i množi ih pripadnim težinskim vrijednostima za tu vezu nakon čega nad njima izračunava zbroj. Vrijednosti izlaza iz neurona dobiva se izračunavanjem aktivacijske funkcije za vrijednost zbroja.

6.2 Primjena za problem vožnje automobila

Za naš problem, ulazne vrijednosti mreže bit će vrijednosti udaljenosti koje dobivamo od senzora dok će izlazne vrijednosti biti jačina zakrenutosti volana te vrijednost koja predstavlja jačinu gasa i kočnice. Zakrenutost volana te jačina gasa i kočnice su predstavljeni realnim brojevima u intervalu od -1 do +1. Zbog toga će ulazni sloj biti jednake veličine kao broj senzora dok će izlazni sloj biti veličine dva.

Za aktivacijsku funkciju izlaznog sloja se koristi tangens hiperbolni zbog toga što vraća vrijednosti u intervalu od -1 do +1 koji nam je potreban za izlazne vrijednosti. Tangens hiperbolni u početku je korišten i za skriveni sloj, no njegovo korištenje imalo je posljedicu da većina neuronskih mreža vraća izlazne vrijednosti koje automobil pokreću unatrag i udesno. To bi se događalo neovisno o promjenama strukture neuronske mreže ili parametara genetskog algoritma. Zbog toga se u skrivenim slojevima kao aktivacijska funkcija koristi sigmoidna. Uzrok takve posljedice korištenja tangensa hiperbolnog nije identificiran ali promjenom aktivacijske funkcije se to prestane događati.

6.3 Implementacija

Neuronska mreža implementirana je C++ klasom koja nasljeđuje UObject klasu Unreal Engine-a. Nasljeđivanje UObject klase znači da se klasa može koristiti kao varijabla nacrtima u UE4, i ne treba svoj zasebni fizički objekt. Kako bi UE4 prepoznao klasu te njezine atribute ili funkcije potrebno ih je označiti odgovarajućom

anotacijom, “UCLASS”, “UPROPERTY” te “UFUNCTION”, Također je imenima klasa potrebno dodati odgovarajući prefiks ovisno o klasi koju nasljeđuju. Anotacije mogu također sadržavati ključne riječi poput “BlueprintType” za klase, “BlueprintReadWrite“ za atribute i “BlueprintCallable” za funkcije. Ključne riječi određuju razinu pristupa koju će nacrti (engl. Blueprint) imati. Primjer korištenja anotacija u klasi NeuralNet može se vidjeti na Slici 6.

```
UCLASS(BlueprintType)
class AUTONOMOUSCAR_API UNeuralNet : public UObject
{
    GENERATED_BODY()
public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
        TArray<UNeuralLayer*> layers;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
        bool hasBiasNeurons = false;

    UFUNCTION(BlueprintCallable)
        TArray<float> output(UPARAM(ref) TArray<float> input);
    UFUNCTION(BlueprintCallable)
        void addLayer(UPARAM(ref) UNeuralLayer* layer);
    UFUNCTION(BlueprintCallable)
        void initWeights();
```

Slika 6. Dio koda datoteke NeuralNet.h

7. Genetski Algoritam

Genetski algoritam je optimizacijski algoritam inspiriran evolucijom u prirodi. Algoritam ima broj kromosoma koje predstavljaju populaciju. U svakom prolazu algoritma najbolji kromosomi se zadržavaju, dok se ostali odbacuju. Tada se iz prošle populacije izabiru roditeljski kromosomi, te se njihovim križanjem dobiva novi. Novi kromosom se mutira i dodaje u novu populaciju, i postupak se ponavlja dok se populacija ne popuni. Nova populacija se tada evaluira i ponovno predaje genetskom algoritmu na optimiziranje, sve dok ne dobijemo željene rezultate. U našem slučaju imamo populaciju neuronskih mreža, čije težine predstavljaju kromosome. Populacija će se predavati genetskom algoritmu sve dok ne dobijemo neuronsku mrežu koja će ostvarivati zadovoljavajuće prolaze stazom.

Za optimiziranje potrebna je vrijednost koja predstavlja uspješnost neuronske mreže. Tu vrijednost dobiti ćemo od količine staze koju je automobil uspješno prošao te brzine kojom je prošao. Što je više staze prošao i što je brže prošao, to je mreža koja je upravljala njime bolja.

7.1 Implementacija

Genetski algoritam je izabran zbog toga što ima mogućnost nezavisnog učenja bez unaprijed poznatih izlaznih vrijednosti. To je važno zbog toga što bi idealna neuronska mreža trebala pronalaziti bolje rezultate nego što bi ljudi mogli ostvariti i predati kao njezino željeno ponašanje.

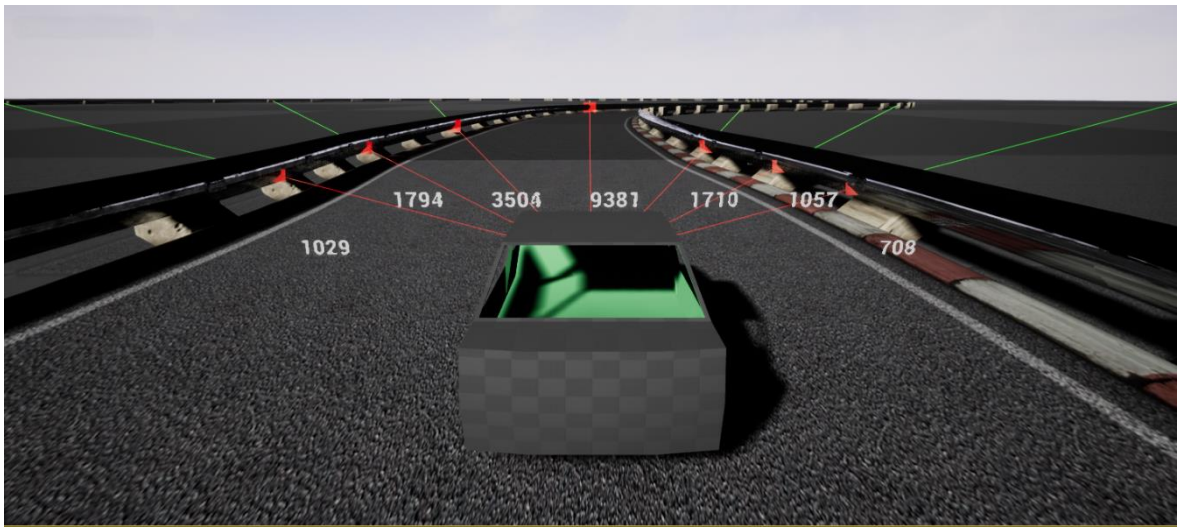
Genetski algoritam implementiran je kao C++ klasa s funkcijom koraka koja se poziva iz nacrta. Funkcija prima neuronske mreže i njihove pripadne vrijednosti greške i stvara novu populaciju. Težine najboljih mreža se zadržavaju, nakon čega se ostatak populacije popunjava novima. Od težina postojećih mreža se biraju dva roditelja, od kojih se pomoću križanja stvaraju nove težine koje se tada mutiraju i dodaju u novu populaciju. Prikaz algoritma je na slici 7.

```
void UGeneticAlgorithm::step(TArray<UNeuralNet*> nets, TArray<float> errorValues) {
    TArray<FWeightsArray> newWeights;

    //pair up neural nets and error values
    netErrorPairs.Empty();
    for (int32 i = 0; i < nets.Num(); i++) {
        FNetErrorPair newPair;
        newPair.Key.weights = nets[i]->getFlatWeights();
        newPair.Value = errorValues[i];
        netErrorPairs.Emplace(newPair);
    }

    //sort by error values
    netErrorPairs.Sort([](FNetErrorPair A, FNetErrorPair B) {
        return A.Value < B.Value;
    });

    //keep the best nets
    for (int32 i = 0; i < elitism; i++) {
        FWeightsArray weightsToKeep;
        weightsToKeep.weights.Append(netErrorPairs[i].Key.weights);
        newWeights.Emplace(weightsToKeep);
    }

    //create new units to fill population
    while(newWeights.Num()<populationSize) {
        TArray<FWeightsArray> parents = getParents();
        FWeightsArray parent1 = parents[0];
        FWeightsArray parent2 = parents[1];

        FWeightsArray newUnitsWeights;
        newUnitsWeights.weights = crossover(parent1.weights, parent2.weights);
        mutate(newUnitsWeights.weights);
        newWeights.Emplace(newUnitsWeights);
    }

    for (int32 i = 0; i < nets.Num(); i++) {
        nets[i]->setWeights(newWeights[i].weights);
    }
}
```

Slika 7. Funkcija koraka genetskog algoritma

7.2 Sustav treniranja

Sustav treniranja koristeći klasu “Game Instance” u koju se pohranjuje populacija, vrijednosti greške za svaku od njih te genetski algoritam. Kada korisnik započne treniranje, u automobil se serijski učitavaju mreže iz populacije i sa svakom mrežom automobil vozi jedan krug. Pri kraju kruga, ovisno o uspjehu ili neuspjehu se izračunava vrijednost greške, koja se sprema u “Game instance”. Kada završi prolaz populacije, mreže i njihove nove vrijednosti greške se predaju genetskom algoritmu.



Slika 8. Prikaz scene iz perspektive automobila

Na slici 8 može se vidjeti prikaz završnog izgleda scene iz perspektive automobila. Iznad automobila su prikazane trenutne vrijednosti senzora.

Treniranje je rađeno na računalu s četiri jezgre frekvencije 3.10 GHz i 8 GB radne memorije. Vrijeme jedne iteracije treniranja mijenja se puno ovisno o performansi pojedinačnih mreža, od nekoliko sekundi u slučaju brzog pogađanja ograde, do nekoliko minuta u slučaju spore vožnje stazom. Nije ostvarena željena performansa neuronskih mreža, prolaz cijelog kruga staze. Genetski algoritam ima tendenciju pronaći lokalni minimum vrijednosti greške koji se inače manifestira udarom ograde na određenoj lokaciji. Neuronske mreže tada počnu težiti pogađanju te lokacije.

8. Zaključak

U ovom završnom radu pokazano je kako u Unreal Engine 4 grafičkom pogonu ostvariti vožnju automobila. Implementirane su neuronske mreže koje su povezane s automobilom u svrhu autonomne vožnje. Neuronske mreže su trenirane pomoću genetskog algoritma. Izradom ovog završnog rada stekao sam nova znanja o Unreal Engine 4 grafičkom pogonu i strojnom učenju.

Za poboljšanje dobivenih rezultata potrebno bi bilo pronaći najbolje dimenzije neuronske mreže i parametre genetskog algoritma. Za njihovo lakše pronalaženje od najveće pomoći bila bi izrada novog sustava treniranja koja omogućava stvaranje više automobila koji istovremeno voze stazom, bez utjecaja jedan na drugog. To bi ubrzalo treniranje tako da se omogući istovremeno testiranje svih neuronskih mreža jedne populacije.

9. Literatura

[1] *Roborace,* http://roborace.com/, datum pristupa 20.1.2019.

[2] *Self Driving Cars*, http://selfracingcars.com/, datum pristupa 20.1.2019.

[3] Ori Cohen (Epic Games), datum nastanka 4. 6. 2014., *Vehicles: Overview & Car Setup | 01 | v4.2 Tutorial Series | Unreal Engine*, https://www.youtube.com/watch?v=9ariPx6M33o, datum pristupa 29. 4. 2017.

[4] Zak Parrish(Epic Games), datum nastanka 10. 12. 2014., *Using Splines & Spline Components | Live Training | Unreal Engine,* https://www.youtube.com/watch?v=wR0fH6O9jD8, datum pristupa 24. 5. 2017.

[5] Epic Games, https://www.unrealengine.com/faq, *Frequently Asked Questions (FAQ),* datum pristupa 8. 6. 2017.

[6] Russel S. J., Norvig P., Artificial Intelligence: A Modern Approach, Third Edition. Prentice Hall, 2010.

SAŽETAK

Simulacija Autonomne Vožnje Automobila

Ovaj završni rad obrađuje korištenje neuronskih mreža za upravljanje virtualnim vozilom. Opisan je način ostvarivanja virtualnog vozila te stvaranja staze u Unreal Engine 4 grafičkom pogonu. Istražen je način povezivanja neuronske mreže s automobilom. Neuronska mreža je trenirana korištenjem genetskog algoritma te iskorištena za upravljanje virtualnim vozilom. Pri izradi rada korišten je vizualni skriptni jezik dostupan u grafičkom pogonu kao i C++ programski jezik.

Ključne riječi: simulacija vožnje vozila; neuronska mreža; genetski algoritam; Unreal Engine 4; C++; virtualno okruženje

Abstract

Simulator for Autonomous Car Driving

This final work deals with using neural networks to control a virtual vehicle. A way of creating a virtual vehicle and race track in the Unreal Engine 4 graphics engine was described. A way of connecting a neural network to a vehicle was explored. A neural network was trained using genetic algorithm and used to control a virtual vehicle. The visual scripting system that is offered in Unreal Engine 4 was used alongside the C++ programming language in the implementation.

Keywords: driving simulation; neural network; genetic algorithm; Unreal Engine 4; C++; virtual environment