SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6404

Autonomno kretanje virtualnih objekata

Marin Hrkec

Zagreb, lipanj 2019.

SADRŽAJ

1. Uvod

Računalni vid, raspoznavanje uzoraka, obrada prirodnog jezika, samovozeći automobili su samo neka od velikih postignuća umjetne inteligencije. Neki zadaci su jednostavni za čovjeka, ali vrlo teški za računalo. Umjetna inteligencija se koristi za rješavanje vrlo složenih problema koje je teško riješiti na klasičan algoritamski način jer se ne zna ni kako ih ljudi rješavaju. Jedan od takvih problema je i autonomno kretanje objekata. Testiranje autonomnog kretanja pravih objekata bilo bi neekonomično pa se često rade grafičke simulacije autonomnog kretanja.

Središte ovog rada je proučiti metode umjetne inteligencije pogodne za autonomno upravljanje kretanjem virtualnih objekata, napraviti modele virtualnih objekata i model scene te implementirati simulaciju kretanja objekata u sceni u grafičkom programskom okruženju *Unity*.

2. Korišteni alati i tehnologije

2.1. Programsko okruženje *Unity*

Unity je multiplatformsko grafičko programsko okruženje (engl. *game engine*) koje omogućuje jednostavnu izradu 2D i 3D igara. *Unity* je razvila tvrtka *Unity Technologies SF* 2005. godine. Podržava pisanje skripti u programskom jeziku C#, dok su u prijašnjim verzijama bili podržani i programski jezici Javascript i Boo. Skripta je jedan od osnovnih dijelova igre, to je tekstualna datoteka s nizom instrukcija koje definiraju program koji *Unity* izvodi.
Velika prednost programskog okruženja *Unity* je velika javna baza skripti, modela, animacija i scena za izradu igre. Za izradu ovog rada, koristio sam neke dijelove iz kolekcije *Standard Assets*.[1]

Unity podržava izradu igara za 25 različitih platformi. Prema podatcima iz 2018. godine, *Unity* se koristio za izradu oko polovice novih mobilnih igara na tržištu i oko 60 posto sadržaja s proširenom ili virtualnom stvarnosti.[4]

2.2. Programski jezik C#

Programski jezik C# je objektno orijentirani programski jezik. Razvila ga je tvrtka *Microsoft* oko 2000. godine zajedno s .NET platformom. Nastao je s namjerom da bude jednostavan, moderan jezik visokih performansi za .NET platformu.[2]

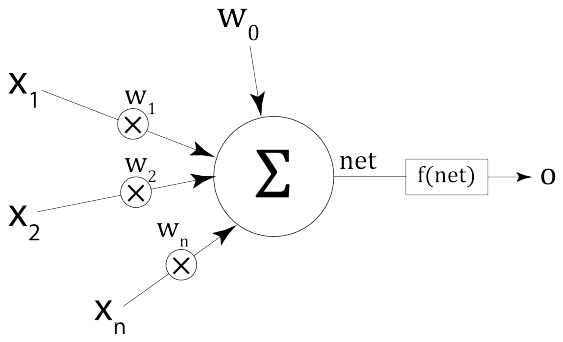
3. Umjetna inteligencija

Umjetna inteligencija (engl. *Artificial Intelligence*) dio je računarske znanosti. Bavi se izgradnjom računalnih sustava koji obavljaju zadaće za koje je potreban neki oblik inteligencije.[3] Jedno od područja umjetne inteligencije je strojno učenje. Strojno učenje je programiranje računala na način da optimiziraju neki kriterij uspješnosti temeljem prethodnog iskustva ili podatkovnih primjera. Učenje se svodi na izvođenje algoritma koji optimizira parametre modela.[5]

3.1. Neuronske mreže

3.1.1. Umjetni neuron

Jednostavan model umjetnog neurona naziva se TLU-perceptor (engl. *Threshold Logic Unit*). To je matematički izraz inspiriran radom i strukturom biološkog neurona. Kao i biološki neuron, perceptron prima ulaze, procesira ih i donosi odluku o izlazu. Model TLU-perceptrona je prikazan na slici 3.1.



Slika 3.1: TLU-perceptron

Svaki ulaz neurona x_i ima odgovarajuću težinu w_i. Vrijednost svakog ulaza x_i množi se s težinom tog ulaza w_i i sumira u tijelu neurona. Toj sumi se još pridodaje konstantan pomak w_0 koji se često označava i b (engl. *bias*). Time je definirana ukupna

vrijednost net.

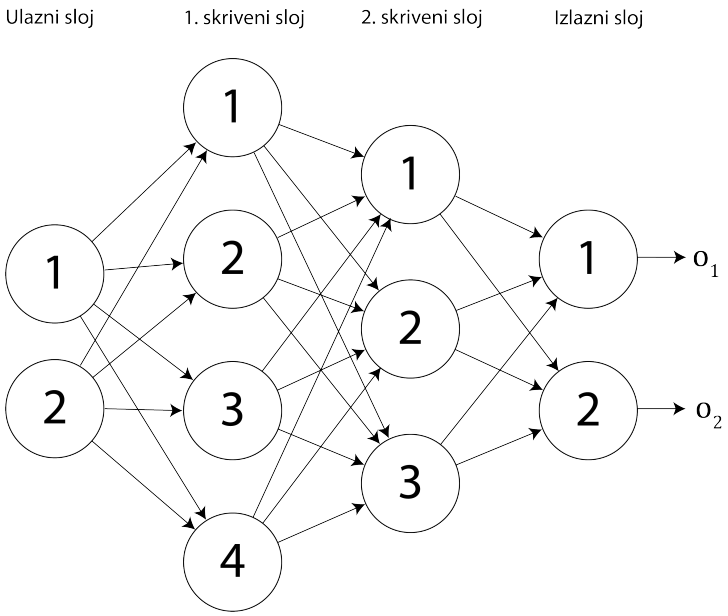
$$net = \left(\sum_{i=1}^{n} x_i \cdot w_i \right) + w_0 \tag{3.1}$$

Ta se vrijednost propušta kroz prijenosnu funkciju (engl. *transfer function*) koja predstavlja konačan izlaz neurona.

3.1.2. Umjetna neuronska mreža

Umjetna neuronska mreža je skup međusobno povezanih jednostavnih procesnih elemenata (neurona) čija se funkcionalnost temelji na biološkom neuronu. U radu s umjetnim neuronskim mrežama postoje dvije faze rada:

1. faza učenja (treniranja) i
2. faza obrade podataka (iskorištavanja)



Slika 3.2: Unaprijedna slojevita umjetna neuronska mreža

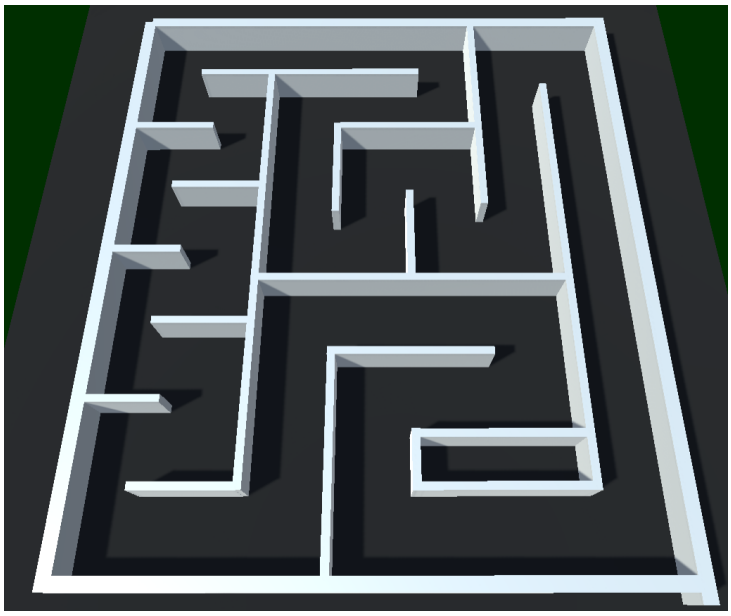
Učenje (treniranje) umjetne neuronske mreže je iterativan postupak predočavanja ulaznih primjera i očekivanog izlaza pri čemu dolazi do postupnog prilagođavanja težina veza između neurona. Jedno predočavanje svih uzoraka naziva se epohom.[6]
Umjetna neuronska mreža se sastoji od tri sloja: ulaznog, skrivenog i izlaznog sloja. Skriveni sloj može sadržavati više podslojeva i svaki podsloj može sadržavati različiti broj neurona. Neuroni ulaznog sloja prosljeđuju ulazne podatke za učenje ostatku

mreže. Neuron nekog sloja na ulazu prima izlaze svih neurona prethodnog sloja pomnožene težinom, a izlaz tog neurona primaju svi neuroni sljedećeg sloja na ulazu. Izlazi neurona izlaznog sloja predstavljaju izlaze neuronske mreže. Model unaprijedne slojevite umjetne neuronske mreže s dva skrivena sloja prikazan je na slici 3.2. U unaprijednoj neuronskoj mreži nema povratnih veza (ciklusa) među neuronima.
Učinkovit postupak učenja višeslojnih neuronskih mreža naziva se postupak propagacije pogreške unatrag (engl. *Error backpropagation*). Postupak osvježava težine neurona nakon propagiranja ulaznih podataka ovisno o pogrešci na izlazu.

4. Struktura scene

4.1. Izrada terena

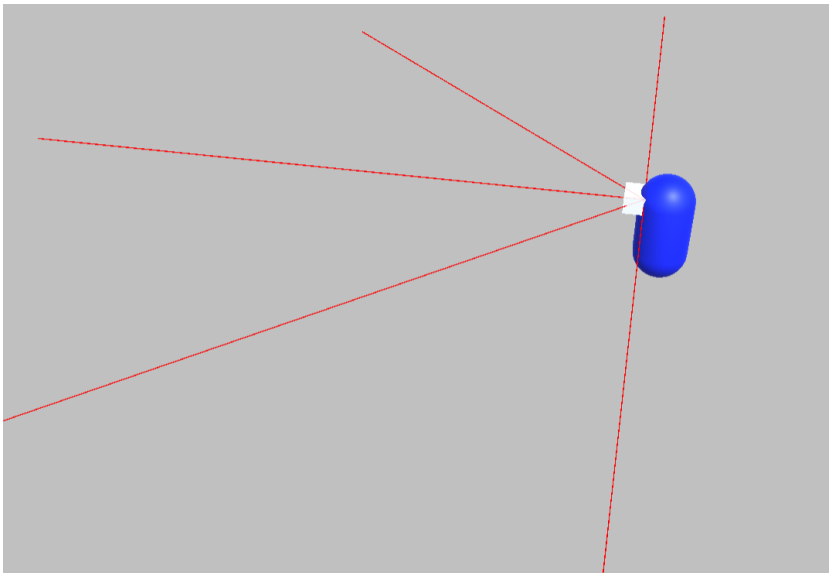
Teren je kvadratnog oblika ograđen zidovima koji definiraju željenu putanju objekta. Izrađen teren je prikazan na slici 4.1. Izrađen teren nema zadani cilj, već je cilj da model lika koji se kreće po sceni prati putanju terena ne sudarajući se sa zidovima.



Slika 4.1: Teren labirinta

4.2. Model pokretnog objekta (lik)

Za kretanje po sceni i prikupljanje podataka za treniranje neuronske mreže se koristi jednostavan objekt — valjak koji na sebi ima kvadar koji predstavlja prednju stranu, tj. oči objekta. Model objekta je prikazan na slici 4.2. Na taj objekt je postavljeno 5 senzora koji se koriste za određivanje udaljenosti objekta od prepreka (zidova). Senzori su označeni crvenom bojom i svi su jednake udaljenosti. Prilikom gibanja kojim upravlja



Slika 4.2: Model za kretanje po sceni

neuronska mreža, giba se lik preuzet iz kolekcije *Standard Assets — AIThirdPerson-Controller*. Model tog lika je prikazan na slici 4.3. Taj lik ima komponentu *AICharac-terControl* pomoću koje može pratiti druge objekte u sceni. Neuronska mreža upravlja valjkom koji se ne prikaže u sceni kojeg prati *AIThirdPersonController* i na taj način daje dojam da zapravo upravlja njime.

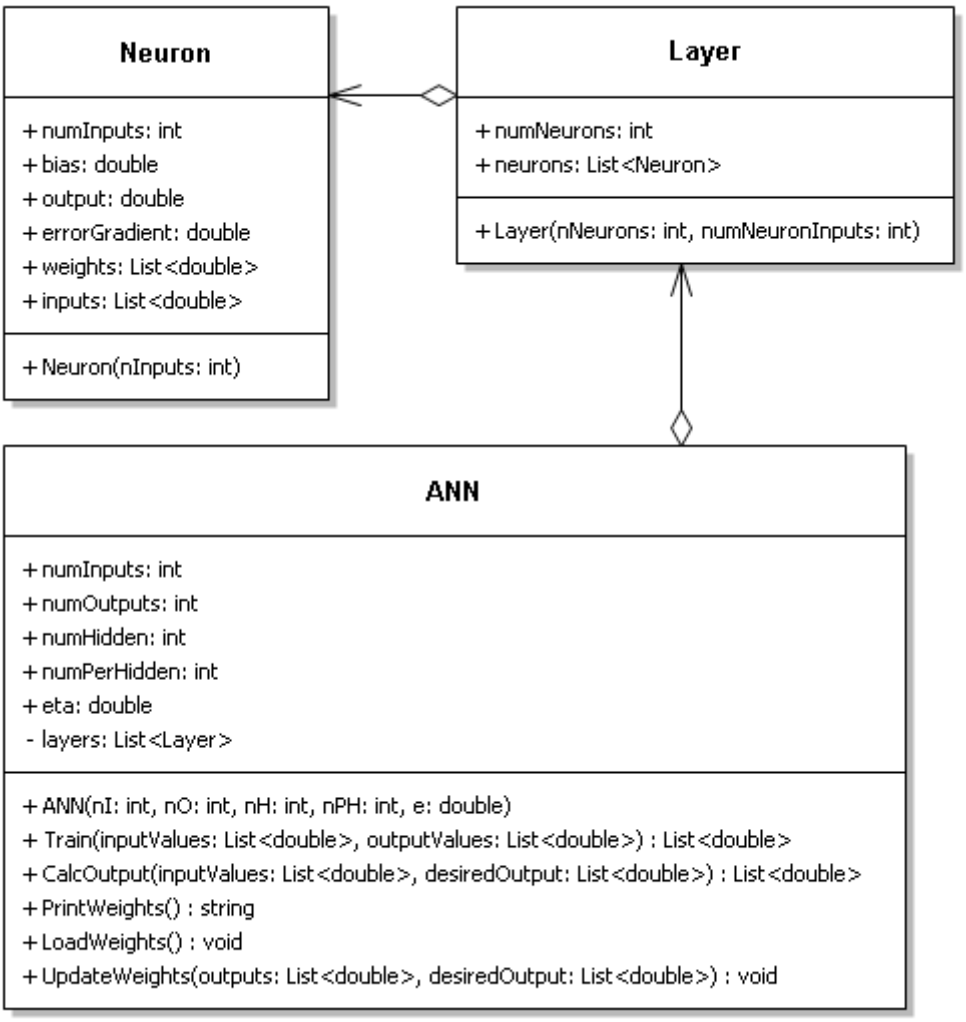
Slika 4.3: Model lika koji se kreće po sceni

5. Implementacija

5.1. Implementacija neuronske mreže

Prva faza rada umjetne neuronske mreže je faza učenja. Za učenje neuronske mreže potrebni su podatci za učenje. Za pokretanje objekta, potrebno je implementirati skriptu za pokretanje. Na slici 5.3 je isječak koda za određivanje udaljenosti pomoću senzora. Prikupljanje podataka je provedeno tako da sam upravljao objektom kroz teren dva puta i u datoteku zapisao udaljenosti senzora od zidova, brzinu i rotaciju objekta. Takvim postupkom je stvorena datoteka veličine 583KB s više od 6000 redaka i ona predstavlja skup za učenje neuronske mreže. Nakon prikupljanja podataka, potrebno je provesti fazu učenja.

Iako postoje razne gotove implementacije umjetnih neuronskih mreža, poput *Tensorflowa*, odlučio sam sâm izraditi jednostavnu neuronsku mrežu. Dijagram razreda za implementaciju neuronske mreže je prikazan na slici 5.1. Razred *Neuron* predstavlja umjetni neuron. *Neuron* ima konstruktor kojem se kao argument predaje broj ulaza u neuron i pri pozivu konstruktora se vrijednosti težina postavljaju na slučajnu vrijednost. Razred *Layer* predstavlja jedan sloj u neuronskoj mreži. *Layer* ima konstruktor kojem se kao argumenti predaju broj neurona u tom sloju i broj ulaza u svaki neuron. Razred *ANN* predstavlja umjetnu neuronsku mrežu (engl. *Artificial Neural Network*). Pri stvaranju mreže, kao argumenti konstruktora šalju se broj ulaza, broj izlaza, broj skrivenih slojeva, broj neurona u skrivenom sloju i stopa učenja η. Metoda *CalcOutput* koristi se računanje izlaza na temelju trenutnih težina neurona. Metoda *UpdateWeights* ažurira vrijednosti težina postupkom propagacije pogreške unatrag. Metoda *Train* se koristi za treniranje neuronske mreže. Kao argumenti, metodi *Train*, šalju se ulazne i očekivane izlazne vrijednosti. *Train* poziva metode *CalcOutput* za izračun izlaza i *UpdateWeights* za ažuriranje težina. Isječak koda metode *Train* prikazan je na slici 5.2. Metoda *PrintWeights* zapisuje vrijednosti težina u tekstualnu datoteku, a metoda *LoadWeights* učitava vrijednosti težina iz tekstualne datoteke.



Slika 5.1: Dijagram razreda neuronske mreže

5.2. Struktura neuronske mreže

Neuronska mreža korištena u radu se sastoji od tri sloja. Odabrana neuronska mreža na ulaz prima podatke o udaljenostima do zida sa senzora. Skriveni sloj sadrži pet neurona, a izlazni sloj sadrži dva neurona - prvi izlaz definira brzinu, a drugi rotaciju objekta. Budući da se objekt može pomicati unaprijed i unazad te rotirati ulijevo i udesno, korištena prijenosna funkcija je tangens hiperbolni.

$$f(x) = tanh(x)$$

$$f : \mathbb{R} \mapsto [-1, 1]$$

Funkcija tangens hiperbolni preslikava skup realnih brojeva u interval $[-1, 1]$ što odgovara pomicanju i rotiranju u dva smjera. Pozitivna vrijednost označava pomicanje

```
public List<double> Train(List<double> inputValues, List<double> desiredOutput)
{
    List<double> outputValues = new List<double>();
    outputValues = CalcOutput(inputValues, desiredOutput);
    UpdateWeights(outputValues, desiredOutput);
    return outputValues;
}
```

Slika 5.2: Metoda *Train*

unaprijed, odnosno rotiranje udesno, a negativna pomicanje unazad, odnosno rotiranje ulijevo. Pri pokretanju scene, mreža počinje s treniranjem i staje nakon određenog broja epoha. Po završetku izvođenja, težine neuronske mreže se zapišu u datoteku.

Takva neuronska mreža je davala najbolje rezultate nakon treniranja. Prvenstveno, širina terena nije bila konstantna pa se neuronska mreža nije snalazila na takvom terenu. Zato sam nastojao kreirati teren s konstantnom širinom. Neuronska mreža se tada puno bolje snalazila i nakon više treniranja sam najbolju neuronsku mrežu spremio u datoteku.

Učenje neuronske mreže je provođeno na računalu s dvojezgrenim procesorom frekvencije radnog takta 2.5 GHz, 8GB radne memorije i grafičkom karticom osnovne frekvencije 1021 MHz. Za zadovoljavajući rezultat, bilo je potrebno 35 minuta učenja u 10000 epoha.

```
RaycastHit hit;
float distance;
if (Physics.Raycast(origin, direction, out hit, visibleDistance))
    distance = hit.distance;
```

Slika 5.3: Očitavanje udaljenosti sa senzora

6. Grafičko korisničko sučelje

6.1. Glavni izbornik

Pri pokretanju apikacije, pojavi se glavni izbornik s četiri gumba. Glavni izbornik je prikazan na slici 6.1. Gumb *Igraj* pokreće scenu za prikupljanje podataka za učenje neuronske mreže. Gumb *Učenje* pokreće scenu za učenje u kojoj se neuronska mreža trenira. Gumb *Učitaj mrežu* pokreće scenu za testiranje naučene neuronske mreže, a gumb *Izađi* služi za izlazak iz aplikacije.

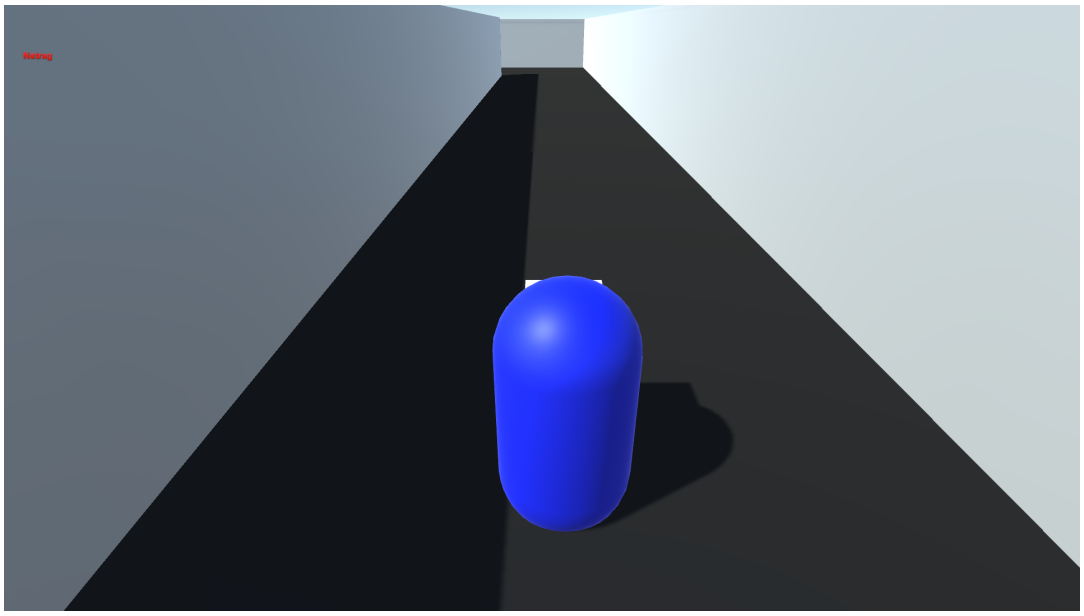
Slika 6.1: Glavni izbornik

6.2. Scena za prikupljanje podataka

U sceni za prikupljanje podataka, korisnik upravlja objektom kroz labirint. Scena je prikazana na slici 6.2. Aplikacija zabilježi udaljenosti sa senzora, brzinu i rotaciju lika

u svakoj sličici (engl. *frame*) i pri izlasku iz scene se ti podaci spremaju u tekstualnu datoteku *trainingData.txt*. Zapisi u toj datoteci su oblika:

```
senzor1,senzor2,senzor3,senzor4,senzor5,brzina,rotacija
```



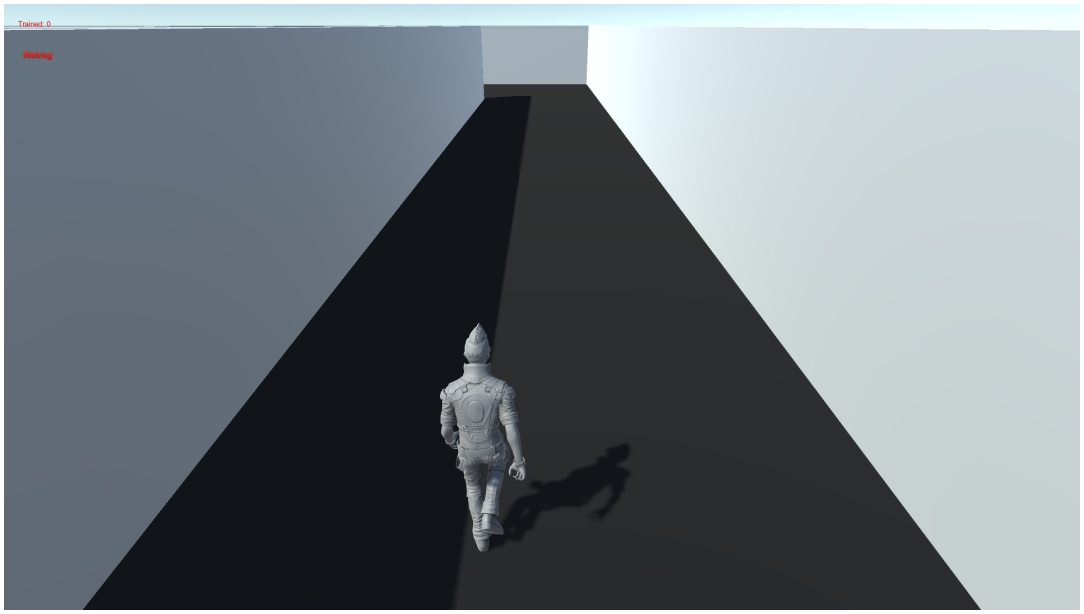
Slika 6.2: Scena za prikupljanje podataka

6.3. Scena za učenje

U sceni za učenje, neuronska mreža uči na temelju podataka zapisanih u datoteci *traiiningData.txt* u određen broj epoha. Scena je prikazana na slici 6.3. Nakon prolaska svih epoha, neuronska mreža počinje upravljati likom u sceni kroz labirint. Pri izlasku iz scene, spremaju se trenutne vrijednosti težina svih neurona u tekstualnu datoteku *weights.txt*.

6.4. Scena za testiranje

Scena za testiranje je istovjetna sceni za učenje, s razlikom da se u sceni za testiranje odmah učitaju težine svih neurona iz tekstualne datoteke *weights.txt*. Nakon učitavanja datoteke, neuronska mreža odmah počinje upravljati likom kroz scenu.



Slika 6.3: Scena za učenje

7. Zaključak

Ideja strojnog učenja je da računalo samo uči na temelju podataka ili prethodnog iskustva. To puno olakšava rješavanje problema koji se ne mogu riješiti algoritamski. U ovom završnom radu, prikazano je kako u grafičkom pogonu *Unity* ostvariti pokretanje objekata. Za simulaciju autonomnog kretanja, odabrana je implementacija umjetnom neuronskom mrežom.

U ovoj simulaciji, umjetna neuronska mreža pokazala se kao zadovoljavajući izbor za simulaciju autonomnog kretanja objekata. Implementirano je da se objektom upravlja samo na temelju podataka dobivenih sa senzora. Za poboljšanje dobivenih rezultata, potrebno bi bilo pronaći optimalne dimenzije neuronske mreže i povećati broj senzora za veću preciznost.

Izradom ovog završnog rada stekao sam nova znanja o grafičkom pogonu *Unity*, strojnom učenju i neuronskim mrežama i smatram da ću znanje stečeno izradom ovog rada moći primijeniti u mnogim drugim područjima.

LITERATURA

[1] Unity Asset Store : Standard Assets. https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351.

[2] Wikipedia: C# (programming language). https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

[3] Enciklopedija.hr: Umjetna inteligencija. http://www.enciklopedija.hr/natuknica.aspx?ID=63150.

[4] DeepMind partners with gaming company for AI research. https://www.dailydot.com/debug/unity-deempind-ai/.

[5] B. D. Bašić i J. Šnajder. *Predavanja: Strojno učenje*, 2016.

[6] B. D. Bašić, M. Čupić, i J. Šnajder. *Predavanja: Umjetne neuronske mreže*, 2018.

Autonomno kretanje virtualnih objekata

Sažetak

U ovom radu opisuje se kako upotrijebiti umjetnu neuronsku mrežu za autonomno kretanje virtualnog objekta u sceni. Obrađen je postupak izrade neuronske mreže. Opisan je postupak propagacije pogreške unatrag. Rezultat rada je simulacija autonomnog kretanja objekta u sceni. Korisnik može tipkovnicom upravljati objektom, generiranim podacima trenirati neuronsku mrežu te pokrenuti autonomno kretanje.

Ključne riječi: umjetna inteligencija, strojno učenje, neuronske mreže, autonomno kretanje, virtualni objekti, C#, Unity

Autonomous Movements of Virtual Objects

Abstract

This thesis describes how to use neural network for autonomous movements of virtual object in a scene. It is described how to make a neural network. Error back-propagation algorithm is described. Resulting work is an autonomous movement of an object simulation. User can navigate an object using a keyboard and train neural network with generated data and start the autonomous movement.

Keywords: artificial intelligence, machine learning, neural networks, autonomous movements, virtual objects, C#, Unity