Matematički fakultet

Univerzitet u Beogradu

Igranje 2D/3D igrica upotrebom neuronskih mreža

Seminarski rad u okviru kursa Računarska inteligencija

Mentor:

Prof. dr Aleksandar Kartelj

Studenti:

Nikola Veselinović

Br. indeksa 200/2015

Aleksandra Tešić

Br. Indeksa 316/2017

Sadržaj

Uvod

Predmet projekta je razvoj 2D/3D igrica primenom neuronskih mreža. Dizajnirana je programirana igrica Super Mario koja svojom poznatom funkcionalnošću omogućava primenu metoda veštačke inteligencije i neuronskih mreža.

Neuronske mreže se često koriste za simuliranje protivničkog igrača kod različitih igrica. Tehnike koje se koriste protežu se od korišćenja evolucijskih algoritama u kombinaciji s neuronskim mrežama, preko učenja s podrškom i dodeljivanja ocene, sve do korišćenja tehnika neuronskih mreža u kombinaciji s teorijom igara.

Neuronska mreža je jedan oblik implementacije sistema veštačke inteligencije, koji predstavlja sistem koji se sastoji od određenog broja međusobno povezanih procesora ili čvorova, ili procesnih elemenata koje nazivamo veštačkim neuronima.

Telo neurona naziva se čvor ili jedinica. Svaki od neurona ima lokalnu memoriju u kojoj pamti podatke koje obrađuje. Podaci koji se obrađuju su lokalni podaci kao i oni koji se primaju preko veze. Podaci koji se ovim kanalima razmenjuju su obično numerički.

Arhitektura neuronske mreže predstavlja specifično povezivanje neurona u jednu celinu. Struktura neuronske mreže se razlikuje po broju slojeva. Prvi sloj se naziva ulazni, a poslednji izlazni, dok se slojevi između nazivaju skriveni slojevi. Najčešće ih ima tri, ali to se uglavnom odnosi na manje projekte jer sto je veći broj neurona u srkivenom sloju to je više informacija potrebno za učenje, i to je više vremena potrebno, ali se zato kompleksnije stvari mogu naučiti. Prvi sloj, tj. ulazni je jedini sloj koji prima podatke iz spoljašnje sredine, sledeći (skriveni) prosleđuje relevantne podatke do trećeg (izlaznog) sloja. Na izlazu trećeg sloja dobijamo konačan rezultat. Složenije neuronske mreže imaju više skrivenih slojeva. Slojevi su međusobno potpuno povezani. U našem projektu na ulaznom sloju imamo 2 neurona.

Slojevi komuniciraju tako što se izlaz svakog neurona iz prethodnog sloja povezuje sa ulazima svih neurona narednog sloja. Znači, svaki čvor ima nekoliko ulaza i jedan izlaz. Jačina veza kojom su neuroni povezani naziva se težinski faktor (weight).

Veštačke neuronske mreže su jedna od najpopularnijih tehnika veštačke inteligencije. Poslednjih godina primenjuju se u mnogim oblastima i postale nezaobilazne u rešavanju sve složenijih problema koji se javljaju u svremenom svetu.

Učenje NM (Neuronska Mreža) se svodi na učenje iz primera kojih bi trebalo da bude što više da bi mreža mogla da se ponaša preciznije u kasnijoj eksploataciji. Proces učenja dovodi do korigovanja sinaptičkih težina. Kada uzorci koji se predstavljaju mreži ne dovode više do promene ovih koeficijenata, smatra se da je dostignut vrhunac učenja.
Postoji tri tipa obučavanja:

- nadgledano obučavanje - mreži se predstavljaju ulazni podaci i očekivani izlazni podaci
- obučavanje ocenjivanjem - mreži se ne predstavljaju očekivani izlazni podaci nego joj se posle izvesnog vremena predstavlja ocena prethodnog rada. Jedan od primera je mreža koja uči da balansira štap. Kad god štap padne, mreži se prosleđuje ocena prethodnog rada, na primer, u obliku ugaonog odstupanja štapa od ravnoteže.
- samoorganizacija - mreži se predstavljaju isključivo ulaz

Polazeći od toga da su veštačke neuronske mreže familija statističkih modela učenja bazirana na biološkim neuronskim mrežama, tačnije neuronima, osnovna ideja primene

veštačke neuronske mreže u ovom radu je da računarska simulacija omogućava učenje pojmova, prepoznavanje šablona i donošenje odluka na način koji je sličan čovekovom.

Kako bi neuronska mreža naučila da prepoznaje i klasifikuje pojmove, mora da postoji povratna informacija. Povezanost između biološkog i veštačkog neurona možemo uočiti na primeru ljudskog mozga kao što je opisano u sledećim rečenicama. Svi ljudi koriste povratne informacije, u svakom trenutku. U ovom slučaju, mozak igrača koji prvi put igra igricu posmatra način kretanja neprijatelja i stvara sliku o tome koji bi bio najjednostavniji način da se stigne do cilja. Evidentira dobre i pogrešne poteze. Sledeći put kad igra, mozak se seća šta je pogrešno uradio i to ispravlja, u nadi da će postići bolje rezultate. Povratna informacija se koristi kako bi se poredio željeni ishod sa ishodom koji se stvarno desio.

Očekuje se da će na tom principu funkcionisati i igrica Mario, tako što će unaprediti aplikaciju održavanjem kvaliteta igren anivou odluke svakog igrača, bez narušavanja balansa i osnovnih principa funkcionalnosti AI (Artificial Inteligence).

Razlog za unapređivanje je to što će i iskusnimij igračima koji su naučili mehaniku igre i dalje biti proporcionalno zanimljivo jer će se jednostavno mehaničko ponašanje promeniti i adaptirati, ubacujući element iznenađenja.

Opis rešenja

Programski jezik u kom je radjen projekat je C#, a okruženje za dizajn igrice je Unity, i korišćeni su Unity Engine paketi za određene funkcije, a okruženje za kucanje koda je Visual Studio. Program se sastoji iz više delova, za skoro svaku instancu je pisan poseban kod, gde se na kraju sve te instance spajaju i čine jedinstven program. U glavnoj instanci- **NeuralNetwork.cs** se nalazi opsti algoritam po kom funkcionisu svi posebni delovi kad se spoje u jednu celinu. Napomenućemo neke delove glavnog koda i njihovo funkcionisanje.

Glavni cilj ovog projekta je pronalaženje načina da igrica ostane zanimljiva bez obzira na to koliko se igrač izvestio u njoj, zard jednostavnosti ovo će biti primenjeno na igrici Super Mario. Svi neprijatelji u globalu imaju jednostavan patern ponašanja i na njima je lako da se ubaci iznenadni element koji se adaptira i menja sa nivoom veštine igrača, jer smo na njima primenili ANN (Artificial Neural Network). Ekvivalentno može se primeniti i na komplikovanije AI, ali, kao što je već rečeno, zbog lakše uočljivosti radimo to na primeru Super Maria. Dakle, kad imamo veliki broj akcija koji AI radi teže je da se primeti razlika u radu i reakcijama kroz vreme, a potrebno je i više vremena i resursa.

Glavna prepreka za Maria predstavlja pečurka, kojom se relativno lagano može upravljati. Pod upravljačem se misli na neuronske mreže koja je kontroliše. Tokom prvih generacija pokretanja programa pečurka se po default-u kreće samo u levo, sve dok ne naiđe do prve prepreke, odnosno zida ili ti „kraja ekrana“. Kako se ide iz generacije u generaciju pečurka se sve više kreće ka Mariu sa ciljem da odnese pobedu nad njim. Naravno, ona lako može biti ubijena tako što Mario skoči na nju. Dakle, iz generacije u generaciju pečurka „stvar svest“ (naravno preko NM) o tome da treba ipak da bude u maloj distanci u odnosu na njega kako bi mogla da izbegne potencijalan napad, a uspela da porazi Maria.

Prema podeli NM prema smeru prostiranja informacija, neuronska mreža koju koristimo je **FeedForward** - Viši slojevi ne vraćaju informaciju u niže slojeve. Vrše prostiranje signala samo u jednom smeru (od ulaza prema izlazu) odnosno propagaciju signala. Feed forward neuronska mreža jeste mreža koja je svrstana u tu grupu na osnovu smera prostiranja informacija. To znači da se informacije koje se prostiru mrežom kreću u određenom smeru i na osnovu njega imamo podelu. Drugi naziv za ovu vrstu neuronskih mreža jeste neuronske mreže sa prostiranjem signala unapred gde se očigledno vidi analogija sa spomenutim kriterijumom podele.

Po ovom kriterijumu imamo i neuronske mreže koje imaju povratnu spregu i nazivamo ih feedback neuronske mreže.. Feedback neuronske mreže koje imaju zatvorene petlje se nazivaju rekurentne neuronske mreže. Dva najčešća tipa feed forward neuronskih mreža su:

- Jednoslojna neuronska meža sa prostiranjem signala unapred
- Višeslojna neuronska mreža sa prostiranjem signala unapred

Prikazaćemo deo koda u kome upotrebljavamo Feed Forward NM.

```
public float[] FeedForward(float[] inputs)
{
    //add inputs to the neuron matrix
    for (int i = 0; i < inputs.Length; i++)
    {
        neurons[0][i] = inputs[i];
    }

    //itterate over all neurons and compute feedforward values
    for (int i = 1; i < layers.Length; i++)
    {
        for (int j = 0; j < neurons[i].Length; j++)
        {
            float value = 0.25f;

            for (int k = 0; k < neurons[i - 1].Length; k++)
            {
                //sum of all weights connections to this neuron with their values in previous layer
                value += weights[i - 1][j][k] * neurons[i - 1][k];
            }

            neurons[i][j] = (float)Math.Tanh(value);//hyperbolic tangent activation
        }
    }

    return neurons[neurons.Length - 1]; //return output layer
}
```

Mutacija je korišćena, i ovaj operator je jako bitan prilikom prolaženja iz generacije u generaciju. Koristi se pre svega da bi se generisale promene u vidu nekih odstupanja, koje svakako poboljšavaju dejstvo algoritma.

Operator **mutacije** unosi novi genetski materijal u populaciju. U slučaju da se operator mutacije ne bi koristio traženje rešenja bilo bi ograničeno genetskim materijalom koji se nalazi u početnoj populaciji. Uz vrlo malu verovatnost mutacije moguće je da se novi genetski materijal brže gubi nego stvara jer ga istiskuje kroz selekciju materijalom postojećih jedinki bolje kvalitete. To može dovesti do toga da pretraga prostora rešenja zaglavi u lokalnom ekstremu. Prevelik operator mutacije brže menja genetski materijal jedinki nego što se kvalitetan materijal može spojiti u jednoj jedinci kroz operator ukrštanja.

Pretraživanje sve više postaje nasumično traženje rešenja. Postoji mnogo raznih vrsta mutacijskih operatora, a neki od češće korišćenih su: promena bit-a - koristi se na kromosomima sa binarnim prikazom rešenja, granična mutacija - menja gen sa jednom od graničnih vrijednosti, uglavnom se koristi kod kodiranja pomoću celih ili brojeva sa pokretnim zarezom , uniformna - menja gen sa nasumičnom vrednošću po uniformnoj distribuciji, Gaussova mutacija - menja gen sa nasumičnom vrednošću po Gaussovoj distribuciji.

```
//mutate neural network weights based by chance
public void Mutate()
{
    for (int i = 0; i < weights.Length; i++)
    {
        for (int j = 0; j < weights[i].Length; j++)
        {
            for (int k = 0; k < weights[i][j].Length; k++)
            {
                float weight = weights[i][j][k];

                //mutate weights value
                float randomNumber = UnityEngine.Random.Range(0f, 1000f);

                if (randomNumber < 2f)
                {
                    //if <2 flip sign of weight
                    weight *= -1f;
                }
                else if (randomNumber < 4f)
                {
                    //if >2 and <4 pick random weight between -0.5 and 0.5
                    weight = UnityEngine.Random.Range(-0.5f, 0.5f);
                }
                else if (randomNumber < 6f)
                {
                    //if >4 and <6 randomly increse weight by 0% to 100%
                    float factor = UnityEngine.Random.Range(0f, 1f) + 1f;
                    if (weight * factor < 1f && weight * factor > -1f)
                    {
                        weight *= factor;
                    }
                }
                else if (randomNumber < 8f)
                {
                    //if >6 and <8 randomly decrese weight by 0% to 100%
                    float factor = UnityEngine.Random.Range(0f, 1f);
                    if (weight * factor < 1f && weight * factor > -1f)
                    {
                        weight *= factor;
                    }
                }

                weights[i][j][k] = weight;
            }
        }
    }
}
```

„Mozak“ programa se nalazi u **Manager.cs.** U tom delu je postavljeno upravljanje kretnje Maria, njegovoj početnoj poziciji na kojoj se nalazi i njegova koordinacija u prostoru, stvaranje pečurke i naravno timer koji je zapravo i najbitnije. U funkciji :

```
void Timer()
{
    isTraning = false;
    mario.transform.position = marioOriginalPosition;
    marioMoveLeft = true;
    marioMoveRight = true;
    marioJump = true;
    t = 0f;
}
```

se zapravo kontrolise preko boolean operatora kad je omoguceno treniranje, Mariovo kretanje u levo i desno i njegov skok. Gde se sve to dalje unapredjuje i kroz treniranje gradi u funkciji:

```
private void Update()
{
    InvokeTimer(time);//set a timer for activating Timer function

    if (!isTraning)
    {
        if (generationNumber == 0)
        {
            //if its the first generation initiate mushtoom neural network
            InitMushroomNeuralNetworks();
        }
        else
        {
            nets.Sort();
            for (int i = 0; i < populationSize / 2; i++)
            {
                nets[i] = new NeuralNetwork(nets[i + (populationSize / 2)]);
                nets[i].Mutate();
                //on restart create a deepcopy of neural network and set new neural network
                nets[i + (populationSize / 2)] = new NeuralNetwork(nets[i + (populationSize / 2)]);
            }

            for (int i = 0; i < populationSize; i++)
            {
                nets[i].SetFitness(0f);//set all fitness to 0
            }
        }


        generationNumber++;

        isTraning = true;
        CreateMushrooms();
    }
```

Radi optimizacije koda koristili smo funkciju cilja koja odredjuje pravac kretanja pečurke. Ulaz je udaljenost Maria od pečurke, a izlaz da li pečurka ide levo ili desno.

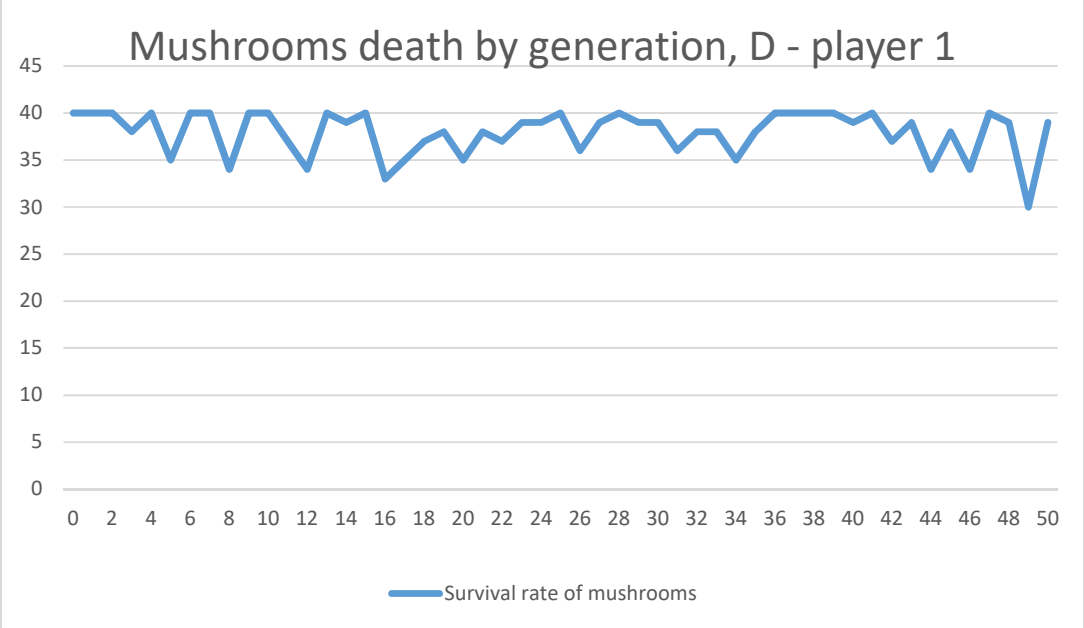
• Optimizacioni algoritmi pripadaju grupi algoritama pretrage
• Cilj je pronaći rešenje problema takvo da je:
1. Neka ciljna funkcija (funkcija cilja) maksimalna/minimalna
2. Neki skup ograničenja zadovoljen (opciono)
• Izazovi:

• Rešenje može biti predstavljeno kao kombinacija vrednosti iz različitih domena
• Ograničenja mogu biti nelinearna
• Karakteristike problema mogu varirati tokom vremena
• Funkcija cilja može biti u „konfliktu“ sa ograničenjima

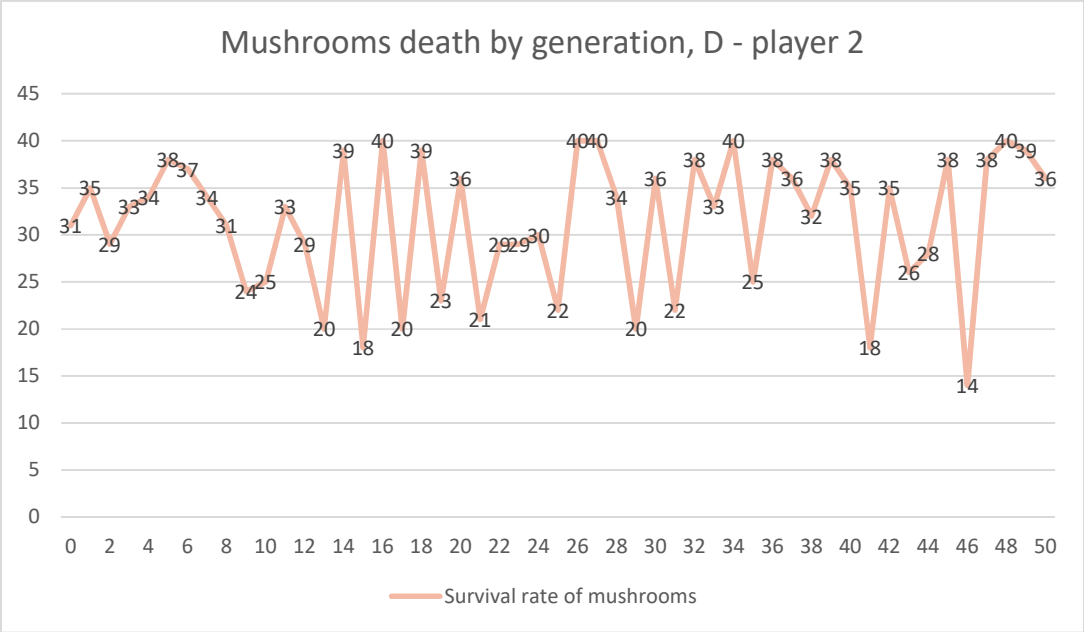
Eksperimentalni rezultati

Eksperimentalnim metodama smo testirali rad našeg projekta. To smo radili uz pomoć igrača koji su nam igranjem same igrice pomogli da dođemo do potrebnih zaključaka. Njihova iskustva i način igre su se menjale iz generacije u generaciju. Cilj je bio da ustanovimo i proverimo kako se AI s vremenom menja i unapređuje. Kroz mišljenje igrača i zapazanja posmatrača smo došli do odgovarajućih zaključaka.

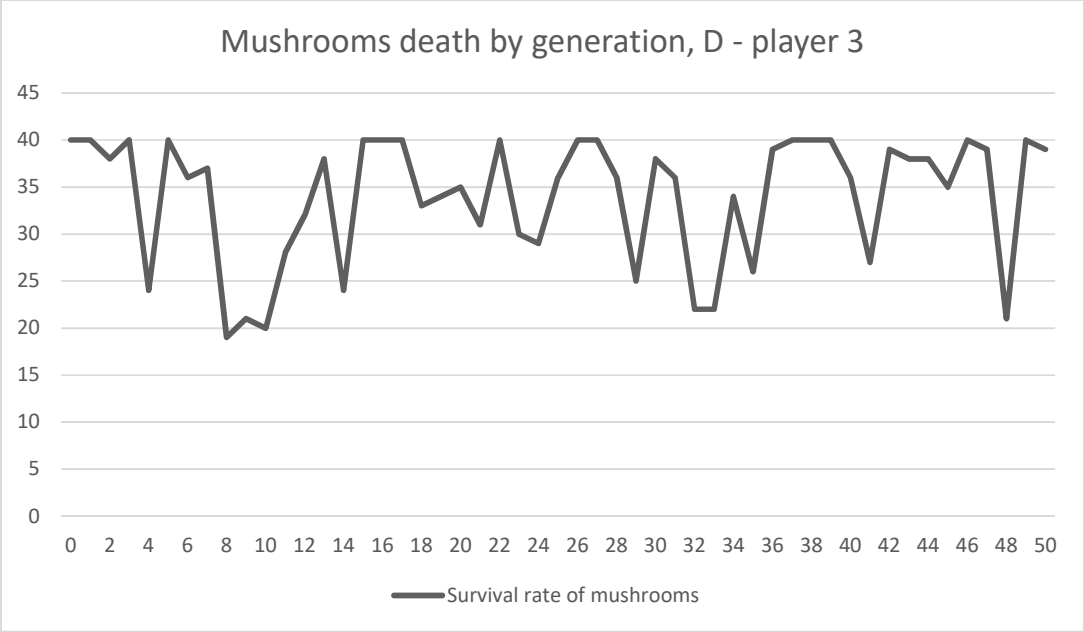
Narednim graficima smo predstavili način ponašanja pečuraka kroz igranje !igrača!. Time podrazumevamo njihovu promenu ponašanja iz generacije u generaciju, što zavisi i od iskustva igrača i njegovih sposobnosti, ali i od činjenice da pečurke postaju sve pametnije što vreme igranja odmiče. To smo sve odradili kroz 50 generacija. U ovim prvim graficima smo koristili neuronsku mrežu ciji neuroni imaju jedan ulazni sloj, dakle pečurke se kreću i opažaju okolinu po zdaljenosti Maria od pečuraka. Formula za to je $\sqrt{(x^2 + y^2)}$, gde je x razlika x koordinate pečurke i x koordinate Maria, a y je razlika y koordinate pečurke i y koordinate Maria. Dakle to je distanca pečurke od Maria u 2D prostoru. Koristili smo oznaku D.



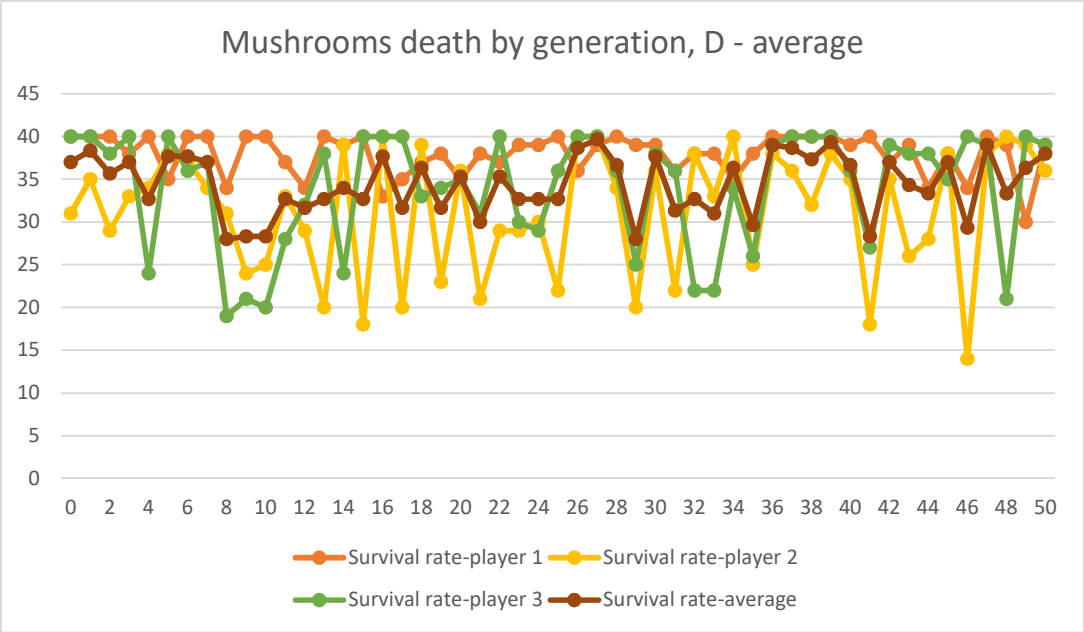
Na prvom grafiku primećujemo da ubijanje pečuraka varira. Pri čemu dolazimo do zaključka da igra vremenom postaje teža, jer pečurke dobijaju sposobnost da lakše izbegavaju napade i bolje ubijaju Maria. Pritom iskustvo igrača je malo lošije sto im povećava brzinu adaptiranja i uvežbavanja na određenom nivou igre.



Ovde već možemo primetiti da je sposobnost igrača znatno veća, što mu je dodatno omogućilo da brže uči i da se adaptira igri. Samim tim po podacima prikazanim vidimo da se i AI brže adaptirao i unapređivao sa njim.

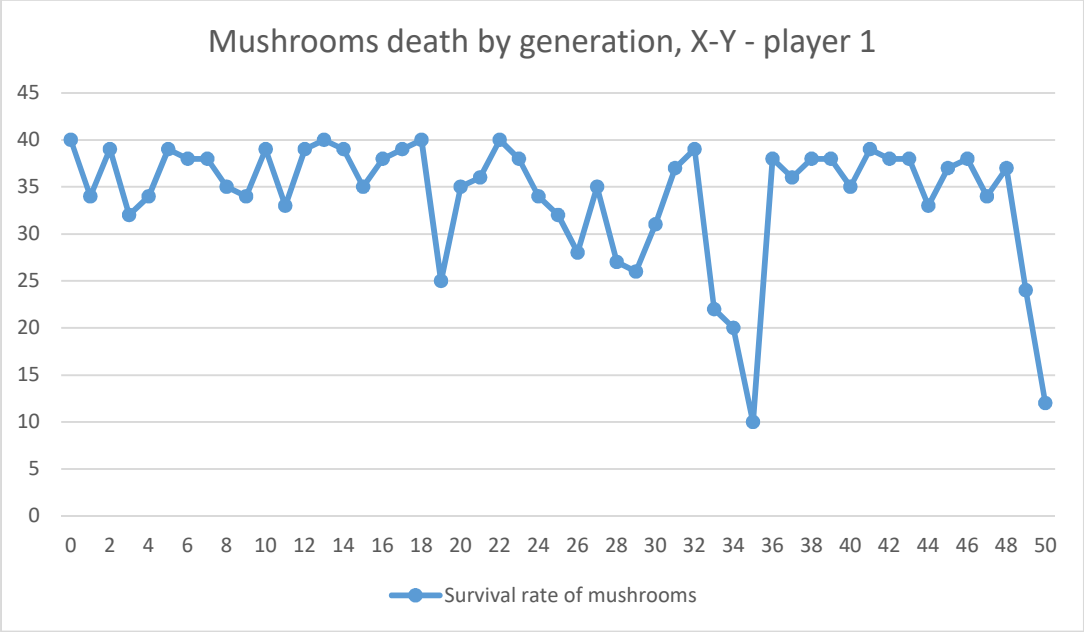


Isto kao i u prethodnom slučaju brzina učenja pečuraka je proporcionalan brzini učenja igrača, ovo je igrač nekog srednjeg nivoa.

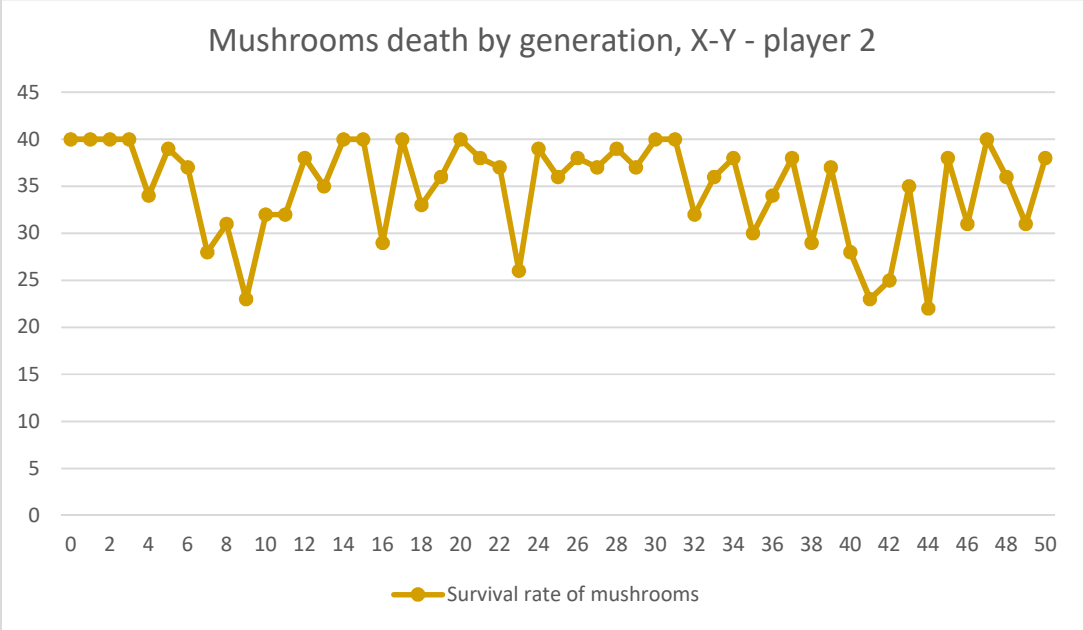


Nakon testiranja igrice na 3 različita igrača uradili smo prosek, jer sva tri prethodna igrača imaju različite nivoe veština, ali i da bismo se približili opštem stanju rezultata. Kao što se vidi, bez obzira što smo sjedinili grafike tri igrača sa različitim veštinama i iskustvima grafik (braon linija) pokazuje i padove i uzdizanja u skoro uniformnom redosledu. To nam dokazuje da unapređenje navikavanje pečuraka na igrača i zatim igrača na pečurke zaista jeste ciklus koji se ponavlja.

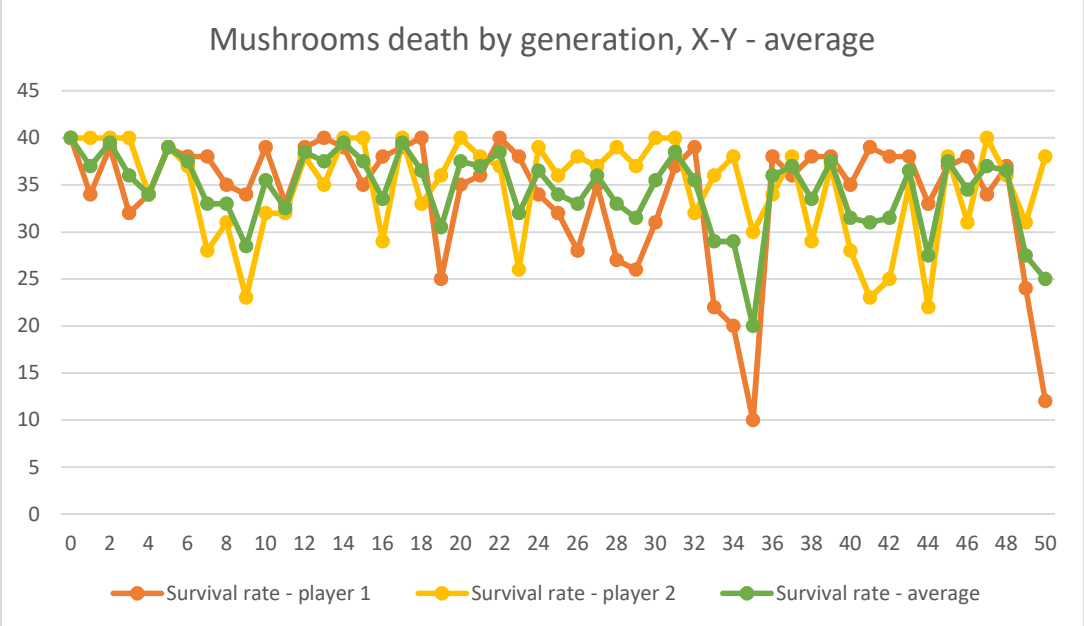
U narednim graficima smo koristili neuronsku mrežu ciji neuroni imaju dva ulazna sloja, dakle pečurke se kreću i opažaju poziciju Maria preko x i y udaljenosti. Gde je x razlika x koordinate pečurke i x koordinat Maria, a y razlika y koordinate pečurke i y koordinate Maria.



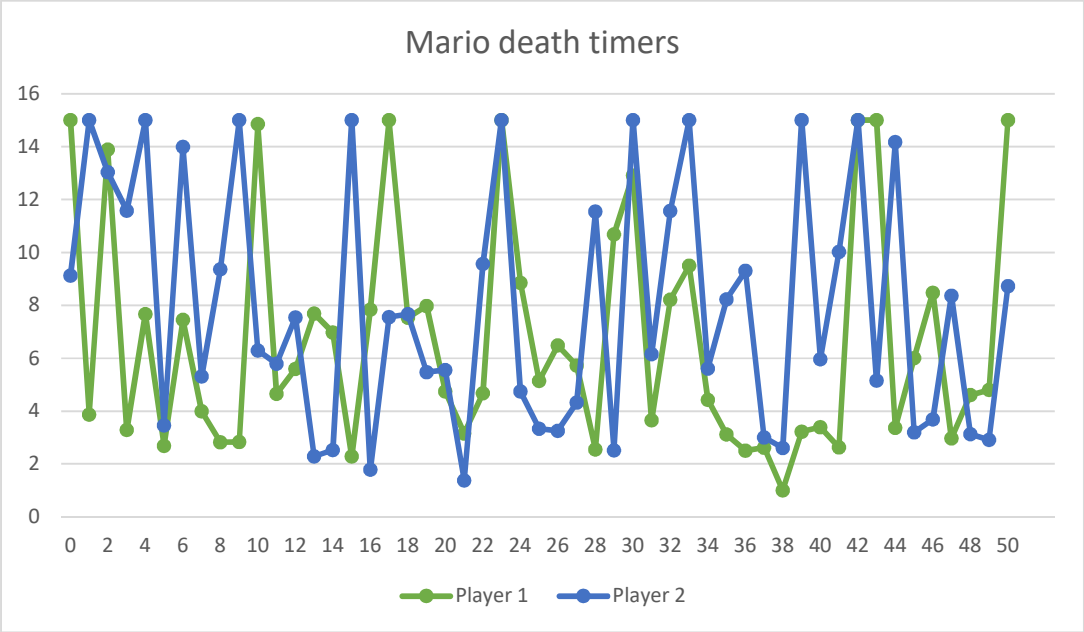
Kod ovakvog pristupa možemo primetiti da su pečurke steklu bolju sposobnost opažanja okoline oko sebe. S tim biva teže ubiti ih, jer iz generacije u generaciju opet postaju pametnije, ali ovaj put na višem nivou. Jer s tim boljim opažanjem okoline mogu lakše da se brane od potencijalnih napada Maria, a i lakše da ga ubiju. Što se ovog igrača tiče možemo primetiti da se u početku upoznaje sa igrom pa i rezultat ne varira toliko, gde nakon određenog broja pokušaja igračeve spospobnosti se povećavaju, pa je i rezultat bolji. Ali nakon toga, što možemo primetiti, to prati i poboljšanje pečurke.



Kod ovog igrača u početku imamo variranja boljih rezultata, gde pečurke nakog toga nadomešćuju poboljšanje igračevih sposobnosti boljim snalaženjem i izbegavanjem napada.



Nakon testiranja na 2 razlicita igrača smo izvadili prosek, koji je obeležen zelenom bojom na grafiku.



Na ovom grafiku prikazujemo kako se igrač navikava na pečurke i kako one podižu nivo težine da bi se prilagodile njemu. Zato postoje trenuci kada igrač preživljava dug period vremena, jer nauči kako se pečurke ponašaju i način na koji se kreću. Tada one promene način ponašanja te se na grafiku zavisnost od intenziteta pada vidi kolika je bila promena. Ono što je zanemarljivo su oni rezultati koji su oko 1-2 s, jer to predstavlja grešku pri padu koncentracije zbog novog početka runde. Poenta je da Mario radi isto sto je radio i u prvom testu i pokuša da ubije što više pečuraka, ali ovog puta umesto da gledamo koliko je pečuraka umrlo, gledamo koliko je dugo Mario ostao živ. Ovaj i prethodni eksperiment daju poprilično celovitu sliku o interakciji i procesu učenja pečurka-igrač i igrač-pečurka (gde stavljamo pečurka-igrač prvo jer u prvoj generaciji pečurke imaju nasumično generisano ponašanje).

Zaključak

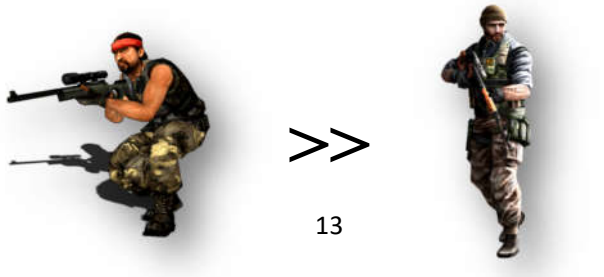
Kroz mnogobrojna testiranja i analizu algoritma za ponašanje određenih elemenata u igrici dosli smo do sledećih zaključaka. Efikasnost pečuraka raste s vremenom što se podiže nivo igre. Kako pečurke uče da se bore protiv igrača, tako će se njihove sposobnosti razvijati istom brzinom kao i igrač. Kada igrač nađe način da ih prestigne one će postepeno podizati težinu igre, dok opet ne postanu izazov za igrača. Što je iskusniji igrač, i što se brže adaptira na novonastale izazove, to su veće amplitude u rezultatima igre, jer što je igrač protiv koga igraju bolji, to će se one brže poboljsavati i postavljati nove izazove. Dakle, kada postanu izazov, dobar igrač će lakse i brze naći novi način da ga prevaziđe, a ekvivalentno tome će se pečurke brže adaptirati.

Kao i dodatnu potvrdu naše hipoteze možemo spomenuti Ellan Muskov OpenAI koji je naučio da igra Dotu, i baziran je na ANN. Koristili su isti pristup da nauče AI da igra igru GO, što je izuzetno impresivno. Princip na kome su ih stavili da uče je isti kao kod nas, nema pravila igre, samo „gledanje“ (ulazni/vizualni podaci koje bi igrači imali) i kroz pokušaje, i kontrolu fitnes funkcije, uči šta treba kad raditi. Dota je kompleksnija od igre GO iz sličnog razloga kao što je CS, Call of Duty, AC, ili bilo koja 3D igra kompleksnija od Maria, ime više varijabli, a jednaku slobodu za kreativnost (obe imaju pravila koja ih drže u određenim granicama, ali u okviru njih imamo veliki nivo slobode). Ovako kompleksna i raznovrsna ponašanja teško mogu da se opišu u linijama koda, linearnim algoritmom, stablom ponašanja, ili njima sličnim metodama. Korišćenje navedenih je dovelo do toga je izraz BOT postao sinonim za nekoga ko ne zna dobro da igra, bilo da je mehanicki los ili da ne zna da se adaptira u neočekivanim situacijama.

Odatle mozemo izvući korisnost ugradnje ANN kao trener ili kontroler botove i u složenijim 2D i 3D igricama. Ovo bi promenilo trenutno tržište dominirano multiplayer igricama. Igracima je potreban izazov, i na današnjem tržištu jedine igrice koje pružaju porporcionalan izazov veštini igraca jesu multyplayer igrice sa ranking sistemima. Problem ovih igrica je u tome sto postoje ljudi koji se namerno spuštaju sa visoko rangiranih mesta kako bi igrali protiv slabijih igraca, danas poznato pod nazivom „smurfing“. Pored toga zbog anonimnosti koju pruža alijas koji se koristi u većini igara, sve je veći broj ljudi koji su neprijatni i koji koriste igru da izbace svoje frustracije na drugima, poznatiji kao „troleri“. Zbog nedostatka singleplayer igirca koje mogu da adaptiraju nivo tezine da odgovara individualnom igraču, igracima ostaju samo dve opcije, da prestanu da igraju ili da trpe ovakve neprijatnosti.

To ne moraju da budu jedina rešenja, ubacivanjem ANN kao kontrolera za botove omogucava ostvarivanje baš tog efekta i primena je raznovrsna i u multiplayer i singleplayer, 3D i 2D igircama. Samo od nekih primera za primenu su: botovi koji mogu da zamene pravog igraca u slučaju da neko napusti meč, ili ako nije bilo moguće da se skupi dovoljno igraca istog nivoa veštine u odredjenom vremenskom periodu.

Drugi zaključak do kog smo došli je, da je vrlo bitan dizajn ANN-a, jer u zavisnosti od složenosti ANN-a, brzina učenja i generalno ponašanje botova je znatno drugačije, što mozemo videti iz D i X-Y eksperimenta. Te su botovi brže učili i adaptirali se u X-Y verziji.Oni koji su imali detaljnije informacije pokazivali su agresivnije ponašanje, dok su se botovi sa manje informacija, opreznije kretali kako bi nadmašili taj isti nedostatak informacija, baš zbog toga su učili i nalazili nove načine da interpretiraju date infromacije kako bi došli do cilja.



Korišćena Literatura

- Computational Intelligence - An Introduction, Andries Engelbrecht, John Willey & Sons, 2007
- https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7
- https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm
- http://www.maturskiradovi.net/forum/Thread-neuronske-mre%C5%BEe-feed-forward-tipa
- https://bib.irb.hr/datoteka/594673.mcrnic_diplomski.pdf